Introducing Datatypes in \textit{DL-Lite}

Ognjen Savković\textsuperscript{1} and Diego Calvanese\textsuperscript{1}

Abstract. In Description Logics (DLs) and in the ontology-based data access (OBDA) scenario, the use of actual datatypes (such as those adopted in DBMSs) has received only limited attention, although datatypes, with their predefined semantics, might have a substantial impact on the computational complexity of inference in ontology systems. In this work we aim at overcoming these limitations, and study the impact of adding datatypes to the OBDA scenario. To this aim, we introduce a language for datatypes and we define the notion of a datatype hierarchy, constituted by a set of datatypes that depend on each other. We classify hierarchies in three classes according to their distinguishing properties, and we establish a theoretical framework for datatypes in the OBDA scenario, based on three major components: a DL, a class of datatype hierarchies, and a query language. We establish the computational complexity of query answering for various significant instantiations of this framework, as ranging from FOL-rewritable to \textsc{coNP} in data complexity.

1 INTRODUCTION

Using ontologies for the conceptual modeling of a domain of interest is becoming increasingly popular, since ontologies have a formal semantics based on Description Logics (DLs), and since they inherit from modeling languages (like UML) intuitive constructors. Hence, also application scenario where ontologies are proposed as a conceptual view over data repositories, are becoming more and more popular. The idea is that the data underlying an application are encapsulated by an ontology interface, and all access to the data is done through the ontology [9]. This scenario is called ontology-based data access (OBDA), since the major concern is the efficient access to data through the ontology [10]. A key inference task in OBDA is answering (database like) queries by taking into account the ontology, and a key desiderata is that the ontology layer does not introduce significant overhead in dealing with the data, i.e., query answering is not harder (when measured in the size of the actual data) than it would be if the ontology layer was not present. This property is ensured by so-called FOL-rewritability, and several DLs have been proposed recently, grouped under the \textit{DL-Lite} family [7, 3] for which query answering is FOL-rewritable, and hence efficiently executable.

In DLs and in the OBDA scenario, the issues related to the use of actual datatypes (such as those adopted in database management systems) has received only limited attention. Recently, datatypes have been investigated more thoroughly in OWL [12]. OWL datatypes are based on the XML datatypes defined in the XML XSD W3C recommendation\textsuperscript{2}. The standardization of the XML datatypes is based on general recommendations and normatives for datatype declarations and usage presented in the General-Purpose Datatypes (GPD) ISO specification [1]. The XSD specification contains a comprehensive set of datatype generators that operate over already defined datatypes allowing for the creation of new datatypes using subtype operators (facets), unions, vectors, etc. However, the XSD recommendation does not provide a semantics, nor there is an obvious way to do it while keeping the intuitive relationship between numeric datatypes, for instance. For this and other reasons, the W3C recommendation for OWL\textsuperscript{3} adopts only some of the XSD datatypes, and introduces new ones, by overloaded already existing XSD datatypes [12]. For example, it recommends the use of \texttt{owl:real} instead of \texttt{xsd:double} or \texttt{xsd:float}\textsuperscript{4}. Additionally, in OWL 2 one can specify user-defined datatypes by constraining or combining the defined ones. Finally, OWL 2 defines various sub-languages (called profiles), such as OWL 2 QL, and specializes for them also the datatypes.

Interfacing a DL with datatypes. The simplest form of interface to datatypes adopted in DLs, e.g., in OWL 2 and in \textit{DL-Lite\textsubscript{A}}, is via binary relations (called \textit{attributes or data properties}), each from a set of objects to a set of datatype values. The range of such relations can be restricted to a datatype (i.e., a unary datatype predicate). For example, the range of the relation \texttt{hasName} could be restricted to the \texttt{xsd:string} datatype. Such a mechanism is a special case of the one adopted with \textit{concrete domains} [8], which come equipped with predicates of arbitrary arity (as opposed to unary predicates only) defined over a domain of values, and where the DL provides constructs for relating an object to a tuple of values that is required to belong to a predicate of the concrete domain.

Naively adding datatypes to a DL could negatively affect its computational properties. To prevent this, usually a DL provides additional restrictions over admissible datatypes. For example, \textit{EL} and \textit{EL++} require that: (i) satisfiability and implication of datatype predicates are polynomially decidable, and (ii) datatype predicates are convex, i.e., if any conjunction of datatype predicates implies a disjunction of datatypes, then it also implies at least one of the disjuncts [11].

In the \textit{DL-Lite} family, datatypes have been introduced in \textit{DL-Lite\textsubscript{A}}, with the strong condition that all datatypes are pairwise disjoint. The only kind of assertion in which datatypes can appear are range restrictions on attributes (e.g., \texttt{Rng(ssn) ⊆ String}). The DL \textit{DL-Lite\textsubscript{A}}\textsuperscript{[HNA]}\textsuperscript{4} extends \textit{DL-Lite\textsubscript{A}} by allowing for expressing local (i.e., concept dependent) attribute typing, and by allowing for inclusion and disjointness assertions between datatypes (thus relaxing the condition of pairwise disjointness). However, these kinds of assertions are in general not sufficient to properly take into account the semantics of predefined datatypes, which is given a priori (i.e., \textit{T} = \textit{val}(\textit{T}) ⊆ \Delta_{T}^\mathbb{V}, where \textit{val}(\textit{T}) is the given domain of values corresponding to datatype \textit{T}). Hence, the relationships between different datatypes are also predefined. As a consequence, the framework of [4] is only able to capture

\footnotesize
\textsuperscript{1}Free University of Bozen-Bolzano, Bolzano, Italy, lastname@inf.unibz.it
\textsuperscript{2}http://www.w3.org/TR/xmlschema-2/
\textsuperscript{3}http://www.w3.org/TR/owl2-direct-semantics/
\textsuperscript{4}http://www.w3.org/TR/owl2-syntax/#Datatype_Maps
uniform predicates with OWA semantics, but not (predefined) datatypes, given that the latter might satisfy dependencies that cannot be expressed by simple inclusion and disjointness assertions, and which might have an impact on (the complexity of) reasoning. This is shown in the following example.

Motivating example. Suppose now that we constrain staff salaries as specified in Figure 1, using facets of the datatype real (see later). This can be formalized in DLs as follows:

\[
\text{AcadStaff} \subseteq \forall \text{hasSalary}.\text{real}[\geq 20k \land \leq 60k] \\
\text{PhDStudent} \subseteq \forall \text{hasSalary}.\text{real}[\geq 20k \land \leq 40k] \\
\text{Professor} \subseteq \forall \text{hasSalary}.\text{real}[\geq 40k \land \leq 60k].
\]

Assume we are given the ABox

\[
\text{Professor(paul)}, \text{PhDStudent(mary)}, \text{AcadStaff(andrea)}, \text{worksWith(mary, andrea)}, \text{worksWith(andrea, paul)}, \\
\text{employs(john, andrea)}, \text{employs(john, mary)},
\]

and we are asking the query

\[
q(x) \leftrightarrow \text{employs}(x, y) \land \text{salary}(y, s) \land \text{real}([\geq 20k \land \leq 40k](s)) \land \\
\text{worksWith}(y, z) \land \text{salary}(z, t) \land \text{real}([\geq 40k \land \leq 60k](t)).
\]

Considering that the range of salaries of academic staff covers the range of salaries of PhD students and professors, if we require via \text{AcadStaff} \not\subseteq \exists \text{salary} that each academic staff has a salary, then \text{andrea} will have either a student salary or a professor salary (although we don’t know which one). However, reasoning by cases we can conclude that in both cases the query answer should be \text{john}. This shows that predefined datatypes can relate to each other in a more complex way than our KB language is capable to express, and this needs to be taken into account in reasoning.

Contributions. In this work we aim at overcoming these restrictions and study the problem of introducing datatypes into the OBDA scenario. We introduce a formal language over datatypes, that enables creating new datatypes from existing ones using a comprehensive set of constructors. Additionally, we define the notion of datatype hierarchy, constituted by a set of datatypes that can be freely defined using the available constructors. Based on the conditions that are satisfied by the datatype hierarchies, we classify them in three classes \(D_0 \supset D_1 \supset D_2\).

Apart from the ontology language, we explore adding datatypes to other parts of the OBDA scenario, in particular the query language. Specifically, we introduce the language \(UCQ_\omega\), which is obtained by extending standard \(UCQ\)s with datatype constraints. In this framework we distinguish between three major components (\(L + D + Q\)). The first component is constituted by the ontology language \(L\), and

5. To overcome these difficulties, [5], which is the followup work of [4], adopted the conditions on datatypes that were first proposed in [13]. These are the ones on which also the present paper builds on (cf. Section 3).

includes also the interface for combining ontologies with datatype hierarchies. The second component is the class \(D\) of datatype hierarchies considered. The last component is a query language \(Q\) over ontologies and datatype hierarchies (\(L + D\)). The main technical contribution of our work consists in studying the properties of two important instances of the OBDA framework, namely, \(DL-Lite_{e+D_3} + UCQ\) and \(DL-Lite_{e+D_2} + UCQ\), both on an expressive member of the \(DL-Lite\) family of DLs. For both scenarios, we prove the property of FOL-rewritability of query answering and satisfiability. We also show for both cases that the given conditions over datatype hierarchies are indeed necessary to preserve FOL-rewritability of query answering. By relaxing any of the conditions, query answering becomes coNP-hard in data complexity, and hence not only non-FOL-rewritability but also intractable in the size of the data.

2 PRELIMINARIES

We present now the lightweight DLs of the \(DL-Lite\) family for which we develop our results. In DLs, the domain of interest is modeled using concepts (unary predicates), roles (binary predicates connecting two objects), and attributes (binary predicates connecting an object to a value), starting from atomic concepts \(A\), atomic roles \(P\), and atomic attributes \(U\). In \(DL-Lite_\omega\) [9], a basic role, denoted \(R\), is an expression of the form \(P \circ R\), and a basic concept, denoted \(B\), is an expression of the form \(A \circ \exists R\), or \(A \circ \forall R\). Additionally, we make use of datatype names, denoted \(T\). A knowledge base (KB) is a pair \(\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle\) where \(\mathcal{T}\) is a TBox describing intensional knowledge, and \(\mathcal{A}\) an ABox maintaining extensional knowledge. A TBox is a finite set of (positive) inclusions (Pls) \(B_1 \subseteq B_2\) or \(R_1 \subseteq R_2\), negative inclusions (NIs) \(B_1 \not\subseteq B_2\) or \(R_1 \not\subseteq R_2\), and functionality assertions (funct \(R\)) or (funct \(U\)). The DL \(DL-Lite_{e+D_3}\) extends \(DL-Lite\) with role constraints (symm \(R\)), (asym \(R\)), (refl \(R\)) or (irref \(R\)) [3], local attribute restrictions \(B \subseteq \forall U.T\), and global attribute restrictions \(Rng(U) \subseteq T\) [4]. Given a TBox \(\mathcal{T}\), we denote with \(\subseteq\) the reflexive and transitive closure of the \(\subseteq\) relation. Instead, \(DL-Lite_{e+D_2}\) is obtained from \(DL-Lite_{e+D_3}\) by disallowing (positive and negative) role inclusions and functionality assertions.

To define ABoxes, we make use of an alphabet \(\Gamma_{\mathcal{O}}\) of object constants and an alphabet \(\Gamma_{\mathcal{V}}\) of datatype constants. An ABox is a finite set of membership assertions \(A(o), P(o, o'), U(o, d)\), where \(o, o' \in \Gamma_{\mathcal{O}}\) and \(d \in \Gamma_{\mathcal{V}}\). Similar to \(DL-Lite_{e+D_3}\), we impose the syntactic restriction that a role \(P\) declared functional, via (funct \(P\)), or inverse functional, via (funct \(P^{-}\)), cannot be specialized, i.e., cannot appear in the rhs of a role inclusion assertion \(R \subseteq P\) or \(R \subseteq P^{-}\). The same applies to attributes.

Queries. An atom \(A(t)\) is an expression of the form \(A(t)\), \(P(t, t')\), \(U(t, t')\), or \(T(t)\) where \(t\) and \(t'\) are atom terms, i.e., variables or constants from \(\Gamma_{\mathcal{T}} \cup \Gamma_{\mathcal{V}}\). A conjunctive query (CQ) \(q\) over a KB \(K\) is an expression of the form \(q(x) \leftrightarrow \beta(x, y)\), where \(x\) is a tuple of distinct variables, called distinguished, \(y\) is a tuple of distinct variables not occurring in \(x\), called non-distinguished, and \(\beta(x, y)\) is a conjunction of atoms with variables in \(x\) and \(y\), whose predicates are atomic concepts, roles, and attributes from \(K\). A conjunctive query with datatypes \(CQ_{\omega}\) is like a CQ, but it also admits atoms whose predicates are datatype names. A union of CQs (UCQs) (resp., union of CQs (UCQs) (resp., \(CQ_{\omega}\)s) \((UCQ_{\omega}\)s)) is a set of CQs (resp., \(CQ_{\omega}\)s) with the same head. Additionally we assume that \(CQ_{\omega}\)s (\(UCQ_{\omega}\)s) are safe, i.e., for every datatype atom \(T(x)\) in a \(CQ_{\omega}\) there exists at least one attribute atom \(U(y, x)\) in the same \(CQ_{\omega}\). Finally a First-Order Logic (FOL) query \(q\) over a KB is a possibly open, FOL formula \(\phi(x)\) whose predicates are symbols from the KB.
Semantics. The semantics of DL-Lite is based as usual on the notion of FOL interpretation $I = (\Delta_I^{\cdot}, \cdot^I)$, where $\Delta_I^{\cdot}$ is the interpretation domain, the union of two non-empty disjoint sets, $\Delta_I^{\cdot} = \Delta_1^{\cdot} \cup \Delta_2^{\cdot}$, and $\cdot^I$ is the interpretation function. We refer, e.g., to [9] for the semantics of the DL constructs. We just remark that we adopt the unique name assumption (UNA) over the object constants, i.e., for every two object constant $o_1$ and $o_2$, if $o_1 \neq o_2$, then $o_1^I \neq o_2^I$. UNA is not enforced over the datatype constants. When convenient, we may view an interpretation as a set of facts. We also make use of the standard notions of satisfaction, model, and entailment. Given an interpretation $I = (\Delta_I^{\cdot}, \cdot^I)$, the FOL query $q = \phi(x)$ is interpreted in $I$ as the set of tuples $\delta \in \Delta_I^{\cdot} \times \cdots \times \Delta_I^{\cdot}$ such that $I \models q(\delta)$.

A tuple $\vec{c}$ of constants appearing in $K$ is a certain answer to $q$ over $K$, written $\vec{c} \in cert(q, K)$, if for every model $I$ of $K$, we have that $\vec{c}^I \in q^I$. Given an ABox $A$, we have that $\vec{c}^I \in q^I$ for every non-empty ABox $A$ and every tuple $\vec{c}$ of constants occurring in $A$, we have that $\vec{c} \in cert(q(T, A), I)$, if and only if $I^{\cdot}_{DB}(A) \in q_{FOL}^{\cdot}(A)$. We remind that the data complexity of FOL query evaluation is in AC$^0$ [2], which is strictly contained in PTM$\exists$ and hence in coNP.

3 EXTENDING DL-LITE WITH DATATYPES

In this section we present our proposal for extending with datatypes the logics of the DL-Lite family, with the aim of preserving their good computational properties. Inspired by XML and RDF datatype schemas, and by [12], we introduce a language for defining datatypes by using constructors starting from a set of predefined datatypes.

A datatype $T$ is a quadruple $(LS_T, VS_T, FS_T, \cdot^T)$, where $LS_T$ is a countable (possibly infinite) set of (datatype) constants called lexical space, $VS_T$ is a (possibly infinite) set of datatype values, $FS_T$ is a (possibly infinite) set of facets, each being a restriction of the value space, and $\cdot^T : LS_T \rightarrow V_S(T)$ is an interpretation function. For example, the datatype int can be defined as $(\{\ldots, -1, 0, 1, \ldots\}, \mathbb{Z}, \bigcup_{n \in \mathbb{Z}} \{\langle n, >, n, 1, n, 0, \ldots \rangle \} )$.

Each facet of $T$ may depend on one or more parameters, and corresponds to a unary predicate that restricts the value space of $T$. Most of the standard XML datatypes possess facets, like inequalities over numeric datatypes or regular expressions over strings. Moreover, general facets are defined over every datatype (e.g., equality). A facet expression $\phi$ for $T$ is a boolean formula over the facets of $T$ for which the parameters, if present, have been instantiated (we call these instantiated facets). Examples are ranges of reals, such as $\geq 0.001 \land \leq 0.002$ used in Example 1. The semantics of a facet expression is defined in the obvious way, considering the semantics of its parameterized facet components and of the boolean operators.

A datatype can be either primitive, i.e., defined axiomatically (a priory) and not in terms of other datatypes, or derived from other datatypes using datatype ranges. A datatype range over a set $\{T_1, \ldots, T_n\}$ of (primitive or derived) datatypes is an expression $\varphi$ built according to the syntax

$$\varphi \rightarrow \bot_d \mid \top_d \mid T_d \mid T_i[\phi] \mid \{d_1, \ldots, d_n\} \mid \varphi_1 \circ \varphi_2,$$

where $\bot_d$ denotes the bottom datatype (interpreted as the empty set), $\top_d$ denotes the top datatype (interpreted as the set of all possible datatype values, see below), $T_i$ denotes a set of datatypes (interpreted as the set of all possible datatype values), $\varphi$ denotes a facet expression, $\{d_1, \ldots, d_n\}$ is a set of datatype constants from the $T_i$'s, and $\circ$ is a set operator in $\{\land, \lor, \\times, \\cdot\}$. To derive a new datatype $T$ using a datatype range $\varphi$, we can assert $T := \varphi$.

The value space $VS_T$ is obtained from the value spaces of $T_1, \ldots, T_n$, by considering the semantics of the facet expressions and set operators used in $\varphi$. The lexical space $LS_T$ is a set of fresh constants that are in one-to-one correspondence with the constants of the types $T_1, \ldots, T_n$, whose value is in $VS_T$, and that are interpreted as the corresponding constant, while the set $FS_T$ is empty. Notice that defining new datatypes using facets is in line with the idea of user-defined datatypes implemented in OWL 2. For example, a new datatype that represents valid email addresses of staff of the University of Bolzano can be obtained by restricting the string datatype as follows:

$$\text{regMailUnibz} := \text{string}[\text{RegExpr}\{[w]@\inf\.unibz\.it}\}].$$

A datatype hierarchy $D$ is a pair constituted by a countable (possible infinite) set $\{T_1, T_2, \ldots\}$ of datatypes whose lexical spaces are pairwise disjoint, and a countable set $\{\{u_1, u_2, \ldots\}\}$ of untyped constants. The interpretation function $\cdot^D$ of $D$ complies with the semantics of the datatypes, i.e., $\bigcup_{i=1}^{\infty} \cdot^D_i \subseteq \cdot^D_0$. All datatypes in $D$ are primitive or derived from the existing ones in $D$, as specified above, where the derivation relation is assumed to be acyclic. Additionally we impose the UNA over untyped constants (i.e., $u_0^D \neq u_1^D$ when $i \neq j$), and that $u_0^D \neq T_0^D$ for each $i$ and $j$. Untyped constants (which are similar to plain literals in RDF) are introduced mainly for two reasons. First, it is not always the case that the datatype of a constant is known, and second, it naturally fits to the idea that the interpretation domain $\Delta_D$ of a datatype hierarchy, called the datatype domain, is an infinite set that is not restricted to the value spaces of its component datatypes [12]. So, for a given datatype hierarchy $D$, we define

$$\Delta_D = \bigcup_{i=0}^{\infty} \cdot^D_i \cup \bigcup_{j=1}^{\infty} \{u_j^D\}$$

and $\Delta_D^D = \Delta_D$. We point out that unary datatype predicates over a concrete domain play the same role as datatypes in a hierarchy restricting the datatype corresponding to the concrete domain.

Given a datatype hierarchy $D$ with datatypes $\{T_1, T_2, \ldots\}$, we have identified the following conditions, stating important properties of the hierarchy that have an impact on the computational complexity of inference, as we will show later:

(\text{infinite}) There do not exist finitely many $T_i$'s such that $2 \leq |\{T_i^D\}| < \infty$.

(\text{infinite}\text{df}) There do not exist finitely many $T_i$'s and $T_j$'s such that $\bigcap_{i=1}^n T_i^D \not\subseteq T_j^D$, for every $j$, and that $|\bigcap_{i=1}^n T_i^D \setminus \bigcup_{j=1}^m T_j^D| < \infty$.

(\text{opendomain}) There do not exist finitely many $T_i$'s such that $|\Delta_D \setminus \bigcup_{j=1}^m T_j^D| < \infty$.

Given these properties, we classify datatype hierarchies as follows:

$D_0$ is the set of all datatype hierarchies without restrictions.

$D_1$ is the set of datatype hierarchies satisfying the (infinite) restr.

$D_2$ is the set of datatype hierarchies satisfying the (infinite), (infinite\text{df}), and (opendomain) restrictions.

For example, the following three datatype hierarchies where the primitive datatypes and facets are interpreted in the expected way are classified as specified on the right:

$D_0 = \{\text{int}, \text{positiveInt}, \text{int}[\leq 10]\}$

$D_1 = \{\text{int}, \text{positiveInt}, \text{nonNegativeInt}\}$

$D_2 = \{\text{int}, \text{positiveInt}, \text{string}\} \cup \{u_1\}^\infty$.
3.1 Datatype Frameworks

Given the above classification of datatype hierarchies, we are now ready to discuss how the combination of a DL with a datatype hierarchy of a certain class might affect the computational properties of the DL. To this aim, for a DL $\mathcal{L}$ and a class $D$ of datatype hierarchies, we denote with $\mathcal{L} + D$ the class of logics obtained by extending $\mathcal{L}$ with a datatype hierarchy $D$ from $\mathcal{L}$. An $\mathcal{L} + D$ KB (or TBox, ABox) is a KB (resp., TBox, ABox) in $\mathcal{L}$ such that all datatypes appearing in it belong to $D$. From now on we require that every interpretation $I$ of the KB (resp., TBox, ABox) complies with the semantics of $D$, i.e., $\Delta_I^T = \Delta_D$ and $D \subseteq \mathcal{I}^T$.

The characterizing property of OBDA scenarios is FO-rewritability of query answering. In our setting, we can identify three major elements that might affect FO-rewritability: (i) the DL $\mathcal{L}$, (ii) the datatype class $D$, and (iii) the query language $Q$ of queries that can be posed over a KB. We call a combination of choices for these three elements a (datatype) framework, and denote it with $\mathcal{L} + D + Q$. For such frameworks, we are interested in FO-rewritability.

Definition 1 Let $\mathcal{L} + D + Q$ be a datatype framework. Then $\mathcal{L} + D + Q$ is FO-rewritable, if for every datatype hierarchy $D \subseteq \mathcal{L}$, every $\mathcal{L} + D$ TBox $T$, and every query $q$ over $T$ in $Q$, there exists a perfect reformulation of $q$ w.r.t. $T$.

Our aim is to investigate potentially useful cases of datatype frameworks that are FO-rewritable, and identify the conditions under which FO-rewritability for a given framework is lost. Notice that, not surprisingly, adding more expressive power to a certain component of a framework might constrain us in selecting the other two components. In the following two sections we illustrate this with two significant framework instances.

4 FRAMEWORK 1: DL-Lite$_{d+}$ + D$_1$ + UCQ

We now investigate the properties of the framework DL-Lite$_{d+}$ + D$_1$ + UCQ. Notice that DL-Lite$_A$ [9] together with UCQs is a special case of this framework. Indeed, for OWL 2 QL$^6$, which is based on the DL-Lite$_A$ sub-language of DL-Lite$_A$, the set of allowed datatypes is such “… that the intersection of the value spaces of any set of these datatypes is either empty or infinite, which is necessary to obtain the desired computational properties”.

We first show that condition (infinite) of D$_1$ is necessary for this framework to be FO-rewritable (providing a formal justification of the datatype restriction in OWL 2 QL). Indeed, we establish the much stronger negative result that, if this condition is relaxed, then query answering becomes coNP-hard in data complexity. To do so, we reduce a coNP-hard problem to CQ answering over a DL-Lite$_{core} + D$ KB, where D is any datatype lattice that violates the (infinite) condition.

Let $D$ contain datatypes $\{T_1, \ldots, T_n\}$ such that $\bigcap_{i=1}^n T_i^D = k$. For the reduction, we distinguish two cases. (i) When $k \geq 3$, we reduce the complement of the $k$-colorability problem to CQ query answering. (ii) Instead, when $k = 2$, we reduce the 2+2-CNF unsatisfiability problem, in a way similar to the proof in [10]. The TBoxes used in the reductions contain only concept inclusions (i.e., $B_1 \subseteq B_2$) and global attribute restrictions (i.e., $\text{Rng}(U) \subseteq T_1$), hence are expressible in DL-Lite$_{core}$ [9].

Theorem 1 Let $D$ be a datatype hierarchy that violates the (infinite) condition. Then there is a DL-Lite$_{core} + D$ TBox $T$ such that answering CQs over $(T, A)$ is coNP-hard in $|A|$, i.e., in data complexity.

We now prove that DL-Lite$_{d+}$ + D$_1$ + UCQ is FO-rewritable. We proceed in four steps: (1) Normalize the KB. (2) Define a canonical model. (3) Show that KB satisfiability is FO-rewritable. (4) Develop an algorithm for the perfect reformulation of a UCQ.

1. Normalization. In order to simplify the subsequent steps, given a datatype hierarchy $D \subseteq D_1$ and a DL-Lite$_{d+}$ + D KB $\mathcal{K} = (T, A)$, we apply to it a set of normalization rules. We first extend $D$ to a new datatype hierarchy $D'$ by adding to $D$ for each attribute $U$ of $K$ a new datatype $T_{max}(U)$, called maximal range for $U$, derived as follows:

$$T_{max}(U) := \bigcap_{U' \in \mathcal{U}, U \subseteq U' \text{ and } \text{Rng}(U') \subseteq T} T.$$

If for an attribute $U$ there exists no $U'$ and $T$ s.t. $U \subseteq T$ and $\text{Rng}(U') \subseteq T$, then $T_{max}(U) := T$. Then, if $T_{max}(U)$ turns out to be inconsistent in $D'$, i.e., $(T_{max}(U))^{D'} = \emptyset$, we add $U \subseteq \neg U$ to the TBox. To deal with (a)symmetry, we introduce a special role $id$ that is always interpreted as $\text{Id}^T = \{(o, o) \mid o \in \Delta^T\}$ [3].

The normalization rules are shown in Table 1, where $\rightarrow$ means that the assertion(s) on the lhs are replaced by those on the rhs, where $D'$ is extended with a new datatype $T \cap T_{max}(U)$. It is easy to see that the rewriting rules are sound and that their application produces a TBox that is equivalent to the original one and in size.

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Normalization rules for DL-Lite$_{d+}$ + D$_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${\text{symm} R} \rightarrow {\text{R} \subseteq \neg \neg \text{R}}$</td>
<td>$\text{assym} R \rightarrow {\text{R} \subseteq \neg \neg \text{R}}$</td>
</tr>
<tr>
<td>${\text{refl} R} \rightarrow {\text{Id}}$</td>
<td>${\text{irref} R} \rightarrow {\text{R} \subseteq \neg \text{Id}}$</td>
</tr>
<tr>
<td>${B \subseteq \forall U.T} \rightarrow {B \subseteq \forall \mathcal{U}(T \cap T_{max}(U))}$</td>
<td>${B \subseteq \forall U_1.T, U_2 \subseteq \forall U_1} \rightarrow {B \subseteq \forall U_1.T, U_2 \subseteq U_1, B \subseteq \forall U_2.T}$</td>
</tr>
</tbody>
</table>

2. Canonical model. The canonical interpretation $\text{can}(K)$ of a DL-Lite$_{d+}$ + D$_1$ KB $\mathcal{K} = (T, A)$ is constructed using a chase-like procedure [2], starting from $A$ and applying expansion rules considering the PIs in $T$. The domain $\Delta^\text{can}(K)$ of $\text{can}(K)$ contains all object constants in $A$ and a possibly infinite set of fresh objects. For a set $B$ of facts, let $\text{ob}(B)$ denote the set of objects appearing in the facts of $B$. Starting from $\text{can}_0(K) = A$, we inductively define a possibly infinite set $\text{can}(K) = \bigcup_i \text{can}_i(K)$ of facts, where for $i \in \mathbb{N}$:

$$\text{can}_{2i+1}(K) := \text{Close}(\text{can}_{2i}(K)) \quad \text{can}_{2i}(K) := \text{Gen}(\text{can}_{2i-1}(K)).$$

Close$(B)$ contains $B$ and for every $B(o) \in B$ (resp., $R(o_1, o_2) \in B$, $U(o, d) \in B$, and $o \in \text{ob}(B)$) such that $B \subseteq T$ (resp., $R \subseteq T$, $U \subseteq T$, and $Id \subseteq T$), it also contains $A(o)$ (resp., $R'(o_1, o_2)$, $U'(o, d)$, and $R(o, o)$), and nothing else.

To define $\text{Gen}(B)$, we say that a PI $B \subseteq \exists R$ (resp., $B \subseteq \exists U$) is applicable in $B$ if there exists $B(o) \in B$ and no $R(o', o) \in B$ for some $o'$ (resp., no $U(o, d) \in B$ for some $d$). Let $\alpha$ be lexicographically the first applicable PI in $B$. Then,

- if $\alpha$ is $B \subseteq \exists R$, then $\text{Gen}(B) = B \cup \{R(o, o)\}$, for a fresh object $o$,
- if $\alpha$ is $B \subseteq \exists U$, then we proceed as follows. We say that an inclusion $B_i \subseteq \forall U.T \in T$ is active if $B_i \subseteq T$ and $B(o) \in B$. We select a fresh datatype value $d_i$ from $D'$ such that $d_i \in B_i \subseteq \forall U.T$. Is active $B'$ and $B(o) \in B$. A crucial property of the canonical model is that it can be homomorphically mapped to any model of the KB. Formally, we have to adapt the standard definition of homomorphism by parameterizing it with a set of predicates.
Definition 2 Given two interpretations $I = (\Delta^I, \cdot^I)$ and $J = (\Delta^J, \cdot^J)$ over a common set of predicates including the set $P$, a homomorphism $\mu$ from $I$ to $J$ over $P$ is a mapping $\mu : \Delta^I \rightarrow \Delta^J$ such that for each predicate $P \in P$ of arity $n$ and each tuple $(o_1, \ldots, o_n) \in P^I$, we have that $(\mu(o_1), \ldots, \mu(o_n)) \in P^J$.

We state the crucial property of the canonical model.

Lemma 2 Let $D$ be a datatype hierarchy in $D_1$, $K$ a satisfiable DL-Lite+$+D$ KB, and $M$ a model of $K$. Then, there exists a homomorphism from $\text{can}(K)$ to $M$ over the set of concept, role, and attribute symbols of $K$.

Notice that in the above lemma, the homomorphism does not include datatype predicates. Indeed, we can devise an example where the homomorphism does not exist if datatype predicates are considered.

3. Satisfiability. Exploiting Lemma 2, we can prove that

Lemma 3 $\text{can}(K) \models K$ iff $K$ is satisfiable.

However, $\text{can}(K)$ can be infinite and therefore this does not provide an effective method for checking KB satisfiability. To do so, we extend instead the technique of $\cdot^I$ which is based on constructing a FOL query that checks for violations of NIs ($T_1$) and functionality assertion ($T_2$) of the TBox. Namely, to every NI or functionality assertion $\alpha$ in $T$, we associate a Boolean query $q_\alpha$, s.t. $I \models q_\alpha$ iff $I \not\models \alpha$, for every interpretation $I$. For example, the query associated to $R_{o_1} \subseteq \neg R_{o_2}$ is $q_{\alpha}(\cdot) \leftarrow R_1(x, y), R_2(x, y)$. We combine such queries with those resulting from violations of attribute typing, defined below:

$$Q_{\text{assert}} := \bigcup_{\text{Reg}(U) \subseteq T} (U(x, v) \land \neg T(v)) \cup \bigcup_{B \subseteq U \cup T} (B(x) \land \bigcup_{U \cup T} U(x, v) \land \neg T(v)) \cup \bigcup_{q_\alpha} \bigcup_{\alpha \in \text{TBox}} q_\alpha \cdot$$

To check for satisfiability, we can compute the perfect reformulation of $Q_{\text{assert}}$ with respect to $T$, as detailed in Step 4, and evaluate the resulting query over the ABox considered as a database.

Lemma 4 $\text{PerfectRef}(Q_{\text{assert}}, T)[\text{DL}(A)] = \text{false}$ iff $\text{can}(K) \models K$.

From Lemmas 3 and 4, we can conclude

Theorem 5 The satisfiability problem for DL-Lite+$+D_1$ is FOL rewritable.

4. Perfect rewriting. Lemma 2 and the fact that homomorphisms are closed under composition, imply that $\text{cert}(q, K) = q^{\text{can}(K)}$, for every UCQ $q$. Notice that for this to hold it is crucial that $q$ does not contain any datatype predicates, as is the case for the framework that we are considering. To actually compute certain answers, we resort again to perfect rewriting. The PerfectRef algorithm we adopt for DL-Lite+$+D_1$ and UCQ, which rewrites a UCQ $q$ by considering the TBox of $K$, is directly derived from the one for DL-Lite+$+D_1$ as presented in [9], but is extended to take into account the special Id role, needed for reflexivity. Specifically, we add to the iteration over all queries $q$ produced so far, the following step:

for all $Id(t_1, t_2) \in q$ do
  if $t_1$ and $t_2$ are unifiable with $\emptyset$ then
    $q_{\text{ref}} \leftarrow q_{\text{ref}} \cup \text{reduceId}(q, Id(t_1, t_2));$
  end if
end for

Here $\text{reduceId}(q, Id(x_1, x_2))$ returns a new UCQ $\theta(q) \{ \{\theta(Id)\} \}$ in which the Id atom is discarded, and $\text{anon}(\cdot)$ replaces each variable occurring only once in $q$ with the symbol '$\cdot$', so that new rewriting rules in PerfectRef may become applicable. In addition, we extend the rewriting rules so that $\{U(x, y)\}$ can be rewritten into $\{B(x), T_{\text{max}}^{\text{answ}}(U)\}$ if $\exists B \subseteq U \in T$ and $T_{\text{max}}^{\text{answ}}(U) = \{d\}$.

Theorem 6 Let $D$ be a datatype hierarchy in $D_1$, $K = &\langle T, A \rangle$ a satisfiable DL-Lite+$+D$ KB, and $q$ a UCQ over $K$. Then

$$\text{PerfectRef}(q, T)[\text{DL}(A)] = \text{cert}(q, K).$$

Taking into account that $\text{PerfectRef}(q, T)$ depends only on the TBox $T$ and the input query $q$, we conclude that the DL-Lite+$+D_1$+UCQ framework is FOL-rewritable.

The satisfaction of the $D_1$ conditions can be directly checked, given a KB together with a datatype hierarchy. Here, we would like to exemplify some meaningful families of datatype hierarchies in $D_1$.

Example 1 The most prominent datatype restrictions are inequalities (i.e., $\{\leq, \geq, <, >, \neq\}$) over numeric datatypes. For example,

$$D^2_q := Q \cup \{Q[\geq], Q[\leq], Q[>, Q[<], Q[\neq], q \in Q]\},$$

and $D^2_n := Z \cup \{Z[\geq], Z[\leq], \text{for } n \in Z\}$, for $n \in \{n <, n >, n =\}$. These are the families of datatype hierarchies that are good candidates for the OBDA framework, where $Q[\text{facet}]$ (resp., $Z[\text{facet}]$) are datatypes obtained from rationals (resp., integers) by restricting the value spaces. Another example is the set of non-finite XML datatypes.

5 FRAMEWORK 2: DL-Lite+$+D_2$+UCQ$_D$

With respect to the framework studied in Section 4, we extend now the query language with the possibility of using datatype predicates. Clearly, condition (infinite) will continue to be necessary for FOL-rewritability, but we show now that the extension of the query language from UCQ to UCQ$_D$ imposes additional conditions for FOL-rewritability. Specifically, we show that we additionally need to assume the (infinite) and (opendomain) conditions.

Similarly to the previous framework, the proof of necessity of these conditions is based on reductions of coNP-hard problems to UCQ$_D$ answering in DL-Lite+$+D$, where $D$ violates at least one of the conditions in $D_2$.

Again we distinguish two cases. In the first case we reduce 242-CNF unsatisfiability to the cases of UCQ$_D$ answering over KBs with datatype hierarchies respectively satisfying $\bigcap_{i \geq k} T_i^{D} \subseteq T^D \cup T_2$, $|\bigcap_{i \geq k} T_i^{D} \cup T_2| = m$, $D = \{(T), \cup \{u_i\}\}$, or $D = \{(T_1, T_2), \emptyset\}$ (for some $m \geq 1$). In the second case we reduce $k$-coloring (for $k \geq 3$) to UCQ$_D$ answering over KBs with datatypes satisfying $\text{resp. } D = \{(T_1, \ldots, T_k), \emptyset\}, D = \{(T_1, \ldots, T_{k-1}), \cup \{u_i\}\},$ $\bigcap_{i \geq k} T_i^{D} \subseteq \bigcup_{j \neq k} T_j$, or $\bigcap_{i \geq k} T_i^{D} \cup \cup_{j \neq k} T_j = m$.

Theorem 7 Let $D$ be a datatype hierarchy that violates at least one of the (infinite), (opendomain), or (infinite) conditions. Then there is a DL-Lite+$+D$ TBox $T$ such that answering UCQ$_D$s over $\langle T, A \rangle$ is coNP-hard in $\{A\}$, i.e., in data complexity.

We now turn to FOL-rewritability of the framework DL-Lite+$+D_2$+UCQ$_D$, which we prove again in four steps.

1. Normalization. This step is as in Section 4.
2. Canonical model. The construction of the canonical model is analogous to the one in Section 4, except that the fresh datatype value \( d_s \) is selected such that \( d_s \in \cap_{1 \leq i \leq |T|} T_i \) and for any \( m \) datatypes \( T_1, \ldots, T_m \) in \( D' \) s.t. \( \bigcup_{j=1}^{m} T_j \subseteq \cap_{1 \leq i \leq |T|} T_i \) it holds that \( d_s \notin \bigcup_{j=1}^{m} T_j \). Notice that such selection is possible because of the \((\infdiff)\) condition.

Moreover, it is necessary in order to be able to extend the homomorphism from the canonical model to any other model also to the datatype predicates of the knowledge base.

**Lemma 8** Let \( D \) be a datatype hierarchy in \( D_2 \), \( K \) a satisfiable \( DL-Lite_+D \) KB, and \( M \) a model of \( K \). Then, there exists a homomorphism from \( \text{can}(K) \) to \( M \) over the set of concept, role, attribute, and datatype symbols of \( K \).

3. Satisfiability. Considering that \( DL-Lite_+D_2 \) is a special case of \( DL-Lite_+D_1 \) (the second framework is more general only in the query language), it follows that satisfiability of \( DL-Lite_+D_2 \) is \( \mathcal{FOL} \)-rewritable.

4. Perfect rewriting. Given that the datatype predicates are taken into account in the homomorphism from \( \text{can}(K) \) to \( M \), we can follow the same line of reasoning as in the previous framework and show that \( \text{cert}(q, K) = q_{\text{can}(K)} \) for an arbitrary \( UCQ_D \).

We recall that we consider only safe \( UCQ_D \).s. In order to deal with datatype predicates in the input \( UCQ_D \), we need to further extend the PerfectRef algorithm presented in Section 4, with the following part:

```
for all \( U(x_\omega) \in q \) do
    if \( T_{\max}(U) \subseteq \text{Rng}(U(x_\omega)) \) then
        \( q_p \leftarrow q_p \cup \text{deleteRng}(q(x_\omega)) \);
    end if
end for
```

Here, we have denoted with \( x_\omega \) (where the subscript is needed to distinguish different such variables) a variable that appears only once in the \( CQ \), counting the head but ignoring the datatype atoms. Then, \( \text{Rng}(U(x_\omega)) \) denotes the conjunction of datatype predicates containing \( x_\omega \), and the method deleteRng eliminates a new \( CQ_D \) from \( q \) by discarding all datatype atoms containing \( x_\omega \).

**Theorem 9** Let \( D \) be a datatype hierarchy in \( D_2 \), \( K = (\mathcal{T}, A) \) a satisfiable \( DL-Lite_+D \) KB, and \( q \) a \( UCQ_D \) over \( K \). Then

\[
(\text{PerfectRef}(q, T))^{\mathcal{D}(\mathcal{A})} = \text{cert}(q, K).
\]

As observed, there is no significant difference between datatype predicates and datatypes themselves. Now, we can ask whether we can relax the conditions on the datatype hierarchy while preserving \( \mathcal{FOL} \)-rewritability, if we assume that only datatype predicates (facts, constraints) but no datatypes appear in the query. Unfortunately, the answer is negative. Indeed, our coNP-hardness reductions (in the cases of \((\infdiff)\) and \((\opendomain)\)) essentially use only super type(s) \( \bigcup_{i=1}^{m} T_i \) in the terminological part, while subtuples \( \bigcup_{j=1}^{m} T_j \) appear exclusively in the query. This proves that a special treatment of datatype restrictions cannot bring additional expressive power on modeling datatype language, and the \( D_2 \) conditions need to be verified over both datatypes and their restricting predicates.

**Example 2** We continue Example 1 and propose suitable families of numerical datatype hierarchies for our second framework:

\[
\begin{align*}
D_2^Q &:= \{ Q \cup \{ Q_{o_q} \mid q \in \mathcal{Q} \}, \{ u_{i}^{\mathcal{Q}} \} \}, \\
D_2^Z &:= \{ Z \cup \{ Z_{o_n} \}, \{ u_{i}^{\mathcal{Z}} \} \},
\end{align*}
\]

for \( o_q \in \{ \leq, \geq, <, > \} \), \( o_n \in \{ \leq, \geq, <, n, > \} \), and \( n \in \mathbb{Z} \).

6 CONCLUSION

| Table 2. Data complexity of query answering in \( DL-Lite_\mathcal{A} \) framework. |
|-------------------------------|---|---|---|
| \( UCQ \) | \( D_1 \) | \( D_2 \) |
| coNP | AC\({}_{\mathcal{A}}\) | AC\({}_{\mathcal{A}}\) |

We summarize in Table 2 our results on the data complexity of query answering in the two variant of our datatype framework presented in Sections 4 (Column 2) and 5 (Column 3).

Column 1 shows the case of \( DL-Lite_\mathcal{A} \), in which datatype hierarchies satisfy the \((\infinitessim)\) condition and where in addition the considered datatypes are pairwise disjoint. The coNP lower-bound for \( UCQ_D \) answering is a consequence of the violation of \((\opendomain)\). The set of datatypes supported in OWL 2 QL satisfy the conditions of the \( D_1 \) datatype class (in fact, much stricter conditions, and we have shown here that such conditions are necessary and sufficient to ensure \( \mathcal{FOL} \)-rewritability of \( CQ \) answering (both in OWL 2 QL and in \( DL-Lite_\mathcal{A} \)). Moreover, by considering queries with datatype predicates \(( UCQ_D )\), the conditions for the stricter \( D_2 \) datatype class are required.

We are interested in extending the proposed framework to \( n \)-ary datatypes (c.f., concrete domains \([8]\)), aiming at finding suitable conditions for \( \mathcal{FOL} \)-rewritability. We are currently working on implementing an extension of the presented framework in a prototype OBDA reasoner, providing support for XML and user-defined datatypes.

REFERENCES