ECAI 2012
Luc De Raedt et al. (Eds.)

doi:10.3233/978-1-61499-098-7-720

Introducing Datatypes in *DL-Lite*

Ognjen Savković[1] and **Diego Calvanese**[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 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 an ontology. 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 *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[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[3] adopts only some of the XSD datatypes, and introduces new ones, by overloading already existing XSD datatypes [12]. For example, it recommends the use of *owl:real* instead of *xsd:double* or *xsd:float*[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 *DL-Lite*$_\mathcal{A}$, is via binary relations (called *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 $hasName$ could be restricted to the *xsd:string* datatype. Such a mechanism is a special case of the one adopted with *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 dataypes to a DL could negatively affect its computational properties. To prevent this, usually a DL provides additional restrictions over admissible datatypes. For example, $\mathcal{EL}$ and $\mathcal{EL}^{++}$ [6] require that: *(i)* satisfiability and implication of datatype predicates are polynomially decidable, and *(ii)* datatype predicates are convex, i.e., if any conjunction of datatypes implies a disjunction of datatypes, then it also implies at least one of the disjuncts [11].

In the *DL-Lite* family, datatypes have been introduced in *DL-Lite*$_\mathcal{A}$, 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., $\mathsf{Rng}(ssn) \sqsubseteq$ *String*). The DL *DL-Lite*$_{core}^{(\mathcal{HN}\mathcal{A})}$ [4] extends *DL-Lite*$_\mathcal{A}$ 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., $T^\mathcal{I} = val(T) \subseteq \Delta_V^\mathcal{I}$, where $val(T)$ is the given domain of values corresponding to datatype T). Hence, the relationships between different datatypes are also predefined. As a consequence, the framework of [4] is only able to capture

[1] Free University of Bozen-Bolzano, Bolzano, Italy, *lastname*@inf.unibz.it
[2] `http://www.w3.org/TR/xmlschema-2/`
[3] `http://www.w3.org/TR/owl2-direct-semantics/`
[4] `http://www.w3.org/TR/owl2-syntax/#Datatype_Maps`

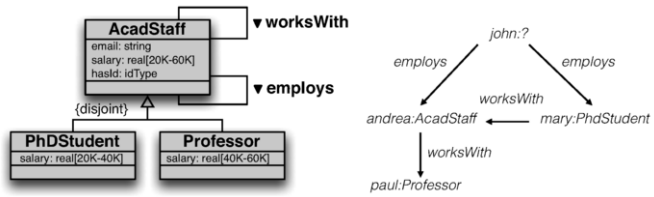

Figure 1. UML class diagram and ABox for the scenario of Example 1

unary 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.[5] 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:

$$\begin{aligned} \mathit{AcadStaff} &\sqsubseteq \forall \mathit{hasSalary}.\mathit{real}[\geq_{20K} \wedge \leq_{60K}] \\ \mathit{PhDStudent} &\sqsubseteq \forall \mathit{hasSalary}.\mathit{real}[\geq_{20K} \wedge \leq_{40K}] \\ \mathit{Professor} &\sqsubseteq \forall \mathit{hasSalary}.\mathit{real}[\geq_{40K} \wedge \leq_{60K}]. \end{aligned}$$

Assume we are given the ABox

$$\begin{aligned} &\mathit{Professor}(\mathit{paul}), \mathit{PhDStudent}(\mathit{mary}), \mathit{AcadStaff}(\mathit{andrea}), \\ &\mathit{worksWith}(\mathit{mary}, \mathit{andrea}), \mathit{worksWith}(\mathit{andrea}, \mathit{paul}), \\ &\mathit{employs}(\mathit{john}, \mathit{andrea}), \mathit{employs}(\mathit{john}, \mathit{mary}), \end{aligned}$$

and we are asking the query

$$\begin{aligned} q(x) \leftarrow\ & \mathit{employs}(x, y) \wedge \mathit{salary}(y, s) \wedge \mathit{real}[\geq_{20K} \wedge \leq_{40K}](s) \wedge \\ & \mathit{worksWith}(y, z) \wedge \mathit{salary}(z, t) \wedge \mathit{real}[\geq_{40K} \wedge \leq_{60K}](t). \end{aligned}$$

Considering that the range of salaries of academic staff covers the range of salaries of PhD students and professors, if we require via $\mathit{AcadStaff} \sqsubseteq \exists \mathit{salary}$ that each academic staff has a salary, then 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 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 $\mathcal{D}_0 \supset \mathcal{D}_1 \supset \mathcal{D}_2$.

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 $UCQ_{\mathcal{D}}$, which is obtained by extending standard UCQs with datatype constraints. In this framework we distinguish between three major components ($\mathcal{L}$+$\mathcal{D}$+$\mathcal{Q}$). The first component is constituted by the ontology language $\mathcal{L}$, and includes also the interface for combining ontologies with datatype hierarchies. The second component is the class $\mathcal{D}$ of datatype hierarchies considered. The last component is a query language $\mathcal{Q}$, over ontologies and datatype hierarchies ($\mathcal{L}$+$\mathcal{D}$). The main technical contribution of our work consists in studying the properties of two important instances of the OBDA framework, namely, *DL-Lite*$_d$+$\mathcal{D}_1$+UCQ and *DL-Lite*$_d$+$\mathcal{D}_2$+$UCQ_{\mathcal{D}}$, both based 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-rewritable but also intractable in the size of the data.

[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).

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*$_{\mathcal{A}}$ [9], a *basic role*, denoted R, is an expression of the form P or P^-, and a *basic concept*, denoted B, is an expression of the form A, $\exists R$, or $\exists U$. 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* (PIs) $B_1 \sqsubseteq B_2$ or $R_1 \sqsubseteq R_2$, *negative inclusions* (NIs) $B_1 \sqsubseteq \neg B_2$ or $R_1 \sqsubseteq \neg R_2$, $U_1 \sqsubseteq \neg U_2$, and *functionality assertions* ($\mathsf{funct}\ R$) or ($\mathsf{funct}\ U$). The DL *DL-Lite*$_d$ extends *DL-Lite*$_{\mathcal{A}}$ with *role constraints* ($\mathsf{symm}\ R$), ($\mathsf{asym}\ R$), ($\mathsf{refl}\ R$) or ($\mathsf{iref}\ R$) [3], *local attribute restrictions* $B \sqsubseteq \forall U.T$, and *global attribute restrictions* $\mathsf{Rng}(U) \sqsubseteq T$ [4]. Given a TBox $\mathcal{T}$, we denote with $\sqsubseteq^*_{\mathcal{T}}$ the reflexive and transitive closure of the $\sqsubseteq$ relation. Instead, *DL-Lite*$_{core}$ is obtained from *DL-Lite*$_{\mathcal{A}}$ by disallowing (positive and negative) role inclusions and functionality assertions.

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

Queries. An *atom* 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 $\Gamma_O \cup \Gamma_V$. A *conjunctive query* (CQ) q over a KB $\mathcal{K}$ is an expression of the form $q(\vec{x}) \leftarrow \beta(\vec{x}, \vec{y})$, where $\vec{x}$ is a tuple of distinct variables, called *distinguished*, $\vec{y}$ is a tuple of distinct variables not occurring in $\vec{x}$, called *non-distinguished*, and $\beta(\vec{x}, \vec{y})$ is a *conjunction* of atoms with variables in $\vec{x}$ and $\vec{y}$, whose predicates are atomic concepts, roles, and attributes from $\mathcal{K}$. A *conjunctive query with datatypes* ($CQ_{\mathcal{D}}$) is like a CQ, but it also admits atoms whose predicates are datatype names. A *union of CQs* (UCQs) (resp., union of $CQ_{\mathcal{D}}$s ($UCQ_{\mathcal{D}}$s)) is a set of CQs (resp., $CQ_{\mathcal{D}}$s) with the same head. Additionally we assume that $CQ_{\mathcal{D}}$s ($UCQ_{\mathcal{D}}$s) are *safe*, i.e., for every datatype atom $T_i(x)$ in a $CQ_{\mathcal{D}}$ there exists at least one attribute atom $U(y, x)$ in the same $CQ_{\mathcal{D}}$. Finally a First-Order Logic (FOL) query q over a KB is a, possibly open, FOL formula $\phi(x)$ whose predicate symbols are from the KB.

Semantics. The semantics of *DL-Lite* is based as usual on the notion of FOL interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, where $\Delta^{\mathcal{I}}$ is the *interpretation domain*, the union of two non-empty disjoint sets, $\Delta^{\mathcal{I}} = \Delta^{\mathcal{I}}_O \cup \Delta^{\mathcal{I}}_V$, and $\cdot^{\mathcal{I}}$ 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 $o_1^{\mathcal{I}} \neq o_2^{\mathcal{I}}$. 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 $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, the FOL query $q = \phi(x)$ is interpreted in $\mathcal{I}$ as the set $q^{\mathcal{I}}$ of tuples $\vec{o} \in \Delta^{\mathcal{I}} \times \cdots \times \Delta^{\mathcal{I}}$ such that $\mathcal{I} \models q(\vec{o})$.

A tuple $\vec{c}$ of constants appearing in $\mathcal{K}$ is a *certain answer* to q over $\mathcal{K}$, written $\vec{c} \in cert(q, \mathcal{K})$, if for every model $\mathcal{I}$ of $\mathcal{K}$, we have that $c^{\mathcal{I}} \in q^{\mathcal{I}}$. Given an ABox $\mathcal{A}$, the interpretation $DB(\mathcal{A}) = \langle \Delta^{DB(\mathcal{A})}, \cdot^{DB(\mathcal{A})} \rangle$ is obtained from $\mathcal{A}$ by taking $\Delta^{DB(\mathcal{A})}$ as the set of constants in $\mathcal{A}$, and interpreting each concept, role, and attribute with the set of facts asserted in $\mathcal{A}$.

Given a TBox $\mathcal{T}$ and a query q, a FOL query q_{FOL} defined over the alphabet of $\mathcal{T}$, is called *perfect reformulation* of q w.r.t. $\mathcal{T}$, if for every non-empty ABox $\mathcal{A}$ and every tuple $\vec{t}$ of constants occurring in $\mathcal{A}$, we have that $\vec{t} \in cert(q, (\mathcal{T}, \mathcal{A}))$ if and only if $\vec{t}^{\,DB(\mathcal{A})} \in q_{FOL}^{DB(\mathcal{A})}$.

We remind that the data complexity of FOL query evaluation is in AC^0 [2], which is strictly contained in PTIME 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 $\cdot^T : LS_{T_i} \rightarrow VS_{T_i}$ is an *interpretation function*. For example, the datatype *int* can be defined as $(\{\ldots, -1, 0, 1, \ldots\}, \mathbb{Z}, \bigcup_{n \in \mathbb{Z}} \{<_n, >_n, \leq_n, \geq_n, =_n, \ldots\}, \cdot^{int})$.

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* ϕ 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 $\geq_{40K} \wedge \leq_{60K}$ used in Example 1. The semantics of a facet expression is defined in the obvious way, considering the semantics of its parametrized 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 φ built according to the syntax

$$\varphi \longrightarrow \bot_d \mid \top_d \mid T_i \mid T_i[\phi] \mid \{d_1, \ldots, d_m\} \mid \varphi_1 \circ \varphi_2,$$

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), $T_i \in \{T_1, \ldots, T_n\}$, ϕ denotes a facet expression, $\{d_1, \ldots, d_m\}$ is a set of datatype constants from the T_i's, and $\circ$ is a set operator in $\{\sqcap, \sqcup, \setminus, \times\}$. To derive a new datatype T using a datatype range φ, 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 φ. 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 dataype as follows:

$$regMailUnibz := string[RegExpr(\texttt{[w]+@inf\textbackslash.unibz\textbackslash.it})].$$

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., $\bigcup_i \cdot^{T_i} \subseteq \cdot^D$. 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., $u_i^D \neq u_j^D$ when $i \neq j$), and that $u_i^D \notin T_j^D$, 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 Δ_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

$$\Delta_D = \bigcup_i T_i^D \cup \bigcup_j \{u_j^D\}$$

and $\top_d^D = \Delta_D$. 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:

- $(infinite)$ There do not exist finitely many T_i's such that $2 \leq |\bigcap_i T_i^D| < \infty$.
- $(infinitediff)$ There do not exist finitely many T_i's and T_j's such that $\bigcap_i T_i^D \not\subseteq T_j^D$, for every j, and $|\bigcap_i T_i^D \setminus \bigcup_j T_j^D| < \infty$.
- $(opendomain)$ There do not exist finitely many T_i's such that $|\Delta_D \setminus \bigcup_i T_i^D| < \infty$.

Given these properties, we classify datatype hierarchies as follows:

- $\mathcal{D}_0$ is the set of all datatype hierarchies without restrictions.
- $\mathcal{D}_1$ is the set of datatype hierarchies satisfying the $(infinite)$ restr.
- $\mathcal{D}_2$ is the set of datatype hierarchies satisfying the $(infinite)$, $(infinitediff)$, 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:

$D_0 = \{int, positiveInt, int[\leq_{10}]\}$ $\quad D_0 \in \mathcal{D}_0, \quad D_0 \notin \mathcal{D}_1,$
$D_1 = \{int, positiveInt, nonNegativeInt\}$ $\quad D_1 \in \mathcal{D}_1, \quad D_1 \notin \mathcal{D}_2,$
$D_2 = \{int, positiveInt, string\} \cup \{u_i\}_i^{\infty}$ $\quad D_2 \in \mathcal{D}_2.$

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 $\mathcal{D}$ of datatype hierarchies, we denote with $\mathcal{L}{+}\mathcal{D}$ the class of logics obtained by extending $\mathcal{L}$ with a datatype hierarchy D from $\mathcal{D}$. 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 $\mathcal{I}$ of the KB (resp., TBox, ABox) complies with the semantics of D, i.e., $\Delta_V^{\mathcal{I}} = \Delta_D$ and $\cdot^D \subseteq \cdot^{\mathcal{I}}$.

The characterizing property of OBDA scenarios is FOL rewritability of query answering. In our setting, we can identify three major elements that might affect FOL rewritability: *(i)* the DL $\mathcal{L}$, *(ii)* the datatype class $\mathcal{D}$, and *(iii)* the query language $\mathcal{Q}$ 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 $\mathcal{L}{+}\mathcal{D}{+}\mathcal{Q}$. For such frameworks, we are interested in FOL-rewritability.

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

Our aim is to investigate potentially useful cases of datatype frameworks that are FOL-rewritable, and identify the conditions under which FOL-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*$_d{+}\mathcal{D}_1{+}UCQ$

We now investigate the properties of the framework *DL-Lite*$_d{+}\mathcal{D}_1{+}UCQ$. Notice that *DL-Lite*$_{\mathcal{A}}$ [9] together with UCQs is a special case of this framework. Indeed, for OWL 2 QL[6], which is based on the *DL-Lite*$_{\mathcal{R}}$ sub-language of *DL-Lite*$_{\mathcal{A}}$, 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 $\mathcal{D}_1$ is necessary for this framework to be FOL-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 $\{T_1, \ldots, T_m\}$ such that $|\bigcap_{i=1}^m T_i^D| = k$. 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., $B_1 \sqsubseteq B_2$) and global attribute restrictions (i.e., $\mathsf{Rng}(U) \sqsubseteq T_i$), 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 $\mathcal{T}$ such that answering CQs over $\langle \mathcal{T}, \mathcal{A}\rangle$ is coNP-hard in $|\mathcal{A}|$, i.e., in data complexity.*

[6] http://www.w3.org/TR/owl2-profiles/#OWL_2_QL

We now prove that *DL-Lite*$_d{+}\mathcal{D}_1{+}UCQ$ is FOL-rewritable. We proceed in four steps: *(1)* Normalize the KB. *(2)* Define a *canonical model*. *(3)* Show that KB satisfiability is FOL-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 $D \in \mathcal{D}_1$ and a *DL-Lite*$_d{+}D$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$, 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 $\mathcal{K}$ a new datatype $T^{\mathcal{T}}_{max}(U)$, called *maximal range* for U, derived as follows:

$$T^{\mathcal{T}}_{max}(U) := \bigsqcap_{\text{exists } U' s.t.\ U \sqsubseteq^*_{\mathcal{T}} U' \text{ and } \mathsf{Rng}(U') \sqsubseteq T \in \mathcal{T}} T.$$

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

The normalization rules are shown in Table 1, where $\rightsquigarrow$ means that the assertion(s) on the lhs are replaced by those on the rhs, where D' is extended with a new datatype $T \sqcap T^{\mathcal{T}}_{max}(U)$. 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 linear in size.

Table 1. Normalization rules for *DL-Lite*$_d{+}\mathcal{D}_1$

$\{(\mathsf{symm}\ R)\} \rightsquigarrow \{R \sqsubseteq R^-\}$	$\{(\mathsf{asym}\ R)\} \rightsquigarrow \{R \sqsubseteq \neg R^-\}$
$\{(\mathsf{refl}\ R)\} \rightsquigarrow \{Id \sqsubseteq R\}$	$\{(\mathsf{iref}\ R)\} \rightsquigarrow \{R \sqsubseteq \neg Id\}$
$\{B \sqsubseteq \forall U.T\} \rightsquigarrow \{B \sqsubseteq \forall U.(T \sqcap T^{\mathcal{T}}_{max}(U))\}$	
$\{B \sqsubseteq \forall U_1.T,\ U_2 \sqsubseteq^*_{\mathcal{T}} U_1\} \rightsquigarrow \{B \sqsubseteq \forall U_1.T,\ U_2 \sqsubseteq U_1,\ B \sqsubseteq \forall U_2.T\}$	

2. Canonical model. The canonical interpretation $can(\mathcal{K})$ of a *DL-Lite*$_d$ + D' KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$ is constructed using a chase-like procedure [2], starting from $\mathcal{A}$ and applying expansion rules considering the PIs in $\mathcal{T}$. The domain $\Delta^{can(\mathcal{K})}$ of $can(\mathcal{K})$ contains all object constants in $\mathcal{A}$ and a possibly infinite set of fresh objects. For a set $\mathcal{B}$ of facts, let $ob(\mathcal{B})$ denote the set of objects appearing in the facts of $\mathcal{B}$. Starting from $can_0(\mathcal{K}) = \mathcal{A}$, we inductively define a possibly infinite set $can(\mathcal{K}) = \bigcup_i can_i(\mathcal{K})$ of facts, where for $i \in \mathbb{N}$:

$$can_{2i+1}(\mathcal{K}) := \mathsf{Close}(can_{2i}(\mathcal{K})) \quad can_{2i}(\mathcal{K}) := \mathsf{Gen}(can_{2i-1}(\mathcal{K}))$$

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

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

- if α is $B \sqsubseteq \exists R$, then $\mathsf{Gen}(\mathcal{B}) = \mathcal{B} \cup \{R(o, o_*)\}$, for a fresh object o_*,
- if α is $B \sqsubseteq \exists U$, then we proceed as follows. We say that an inclusion $B_i \sqsubseteq \forall U.T_i \in \mathcal{T}$ is *active* if $B_i' \sqsubseteq^*_{\mathcal{T}} B_i$ and $B_i'(o) \in \mathcal{B}$. We select a fresh datatype value d_* from D' such that $d_* \in \bigcap_{B_i \sqsubseteq \forall U.T_i \text{ is active}} T_i^{D'}$, if the intersection is not finite. If the intersection is a singleton we choose that value for d_*. Otherwise $\mathcal{K}$ is unsatisfiable, and d_* is an arbitrarily chosen datatype value. In any case we define $\mathsf{Gen}(\mathcal{B}) = \mathcal{B} \cup \{U(o, d_*)\}$.

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 parametrizing it with a set of predicates.

Definition 2 *Given two interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ and $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ over a common set of predicates including the set $\mathcal{P}$, a* homomorphism μ from $\mathcal{I}$ to $\mathcal{J}$ over $\mathcal{P}$ *is a mapping $\mu : \Delta^{\mathcal{I}} \rightarrow \Delta^{\mathcal{J}}$ such that for each predicate $P \in \mathcal{P}$ of arity n and each tuple $(o_1, \ldots, o_n) \in P^{\mathcal{I}}$, we have that $(\mu(o_1), \ldots, \mu(o_n)) \in P^{\mathcal{J}}$.*

We state the crucial property of the canonical model.

Lemma 2 *Let D be a datatype hierarchy in $\mathcal{D}_1$, $\mathcal{K}$ a satisfiable DL-Lite$_d$+D KB, and $\mathcal{M}$ a model of $\mathcal{K}$. Then, there exists a homomorphism from $can(\mathcal{K})$ to $\mathcal{M}$ over the set of concept, role, and attribute symbols of $\mathcal{K}$.*

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 *$can(\mathcal{K}) \models \mathcal{K}$ iff $\mathcal{K}$ is satisfiable.*

However, $can(\mathcal{K})$ 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 [9] that is based on constructing a FOL query that checks for violations of NIs ($\mathcal{T}_n$) and functionality assertion ($\mathcal{T}_f$) of the TBox. Namely, to every NI or functionality assertion α in $\mathcal{T}$, we associate a Boolean query q_α, s.t. $\mathcal{I} \models q_\alpha$ iff $\mathcal{I} \not\models \alpha$, for every interpretation $\mathcal{I}$. For example, the query associated to $R_1 \sqsubseteq \neg R_2$ is $q_n() \leftarrow R_1(x, y), R_2(x, y)$. We combine such queries with those resulting from violations of attribute typing, defined below:

$$Q_{unsat} := \Big(\bigcup_{\mathsf{Rng}(U)\sqsubseteq T} U(x,v) \wedge \neg T(v)\Big) \cup \Big(\bigcup_{B\sqsubseteq\forall U.T} B(x) \wedge U(x,v) \wedge \neg T(v)\Big)$$
$$\cup \Big(\bigcup_{\substack{\{B_i\sqsubseteq\forall U.T_i\}_{i=1}^{l} \text{ s.t.} \\ T_1^{D'}\cap\cdots\cap T_l^{D'}=\emptyset}} B_1(x) \wedge \cdots \wedge B_l(x) \wedge U(x,v)\Big) \cup \bigcup_{\alpha\in\mathcal{T}_n\cup\mathcal{T}_f} q_\alpha.$$

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

Lemma 4 $(\mathsf{PerfectRef}(Q_{unsat}, \mathcal{T}))^{DB(\mathcal{A})} =$ *false iff* $can(\mathcal{K}) \models \mathcal{K}$.

From Lemmas 3 and 4, we can conclude

Theorem 5 *The satisfiability problem for DL-Lite$_d$+$\mathcal{D}_1$ is FOL rewritable.*

4. Perfect rewriting. Lemma 2 and the fact that homomorphisms are closed under composition, imply that $cert(q, \mathcal{K}) = q^{can(\mathcal{K})}$, 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 reformulation. The PerfectRef algorithm we adopt for *DL-Lite$_d$+$\mathcal{D}_1$+UCQ*, which rewrites a UCQ q by considering the TBox of $\mathcal{K}$, is directly derived from the one for *DL-Lite$_\mathcal{A}$* 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 θ **then**
    $q'_{pr} \leftarrow q'_{pr} \cup anon(reduceId(q, Id(t_1, t_2)))$;
  end if
end for

Here $reduceId(q, Id(x_1, x_2))$ returns a new CQ $\theta(q) \setminus \{\theta(Id)\}$ in which the Id atom is discarded, and $anon(\cdot)$ replaces each variable occurring only once in q with the symbol '_', 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 $\{B(x), T^{\mathcal{T}}_{max}(U)\}$ if $\{\exists B \sqsubseteq U\} \in \mathcal{T}$ and $T^{\mathcal{T}}_{max}(U) = \{d\}$.

Theorem 6 *Let D be a datatype hierarchy in $\mathcal{D}_1$, $\mathcal{K} = \langle\mathcal{T}, \mathcal{A}\rangle$ a satisfiable DL-Lite$_d$+D KB, and q a UCQ over $\mathcal{K}$. Then*

$$(\mathsf{PerfectRef}(q, \mathcal{T}))^{DB(\mathcal{A})} = cert(q, \mathcal{K}).$$

Taking into account that $\mathsf{PerfectRef}(q, \mathcal{T})$ depends only on the TBox $\mathcal{T}$ and the input query q, we conclude that the *DL-Lite$_d$+$\mathcal{D}_1$+UCQ* framework is FOL-rewritable.

The satisfaction of the $\mathcal{D}_1$ 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 $\mathcal{D}_1$.

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

$$D_1^Q := Q \cup \{Q[\geq_q], Q[\leq_q], Q[>_q], Q[<_q], Q[\neq_q] \mid q \in \mathbb{Q}\}, \text{ and}$$
$$D_1^{\circ_n} := Z \cup \{Z[\circ_n] \mid n \in \mathbb{Z}\}, \text{ for } \circ_n \in \{\leq_n, \geq_n, <_n, >_n, \neq_n\}$$

are the families of datatype hierarchies that are good candidates for the OBDA framework, where Q[facet] (resp., Z[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$_d$+$\mathcal{D}_2$+$UCQ_\mathcal{D}$*

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 $UCQ_\mathcal{D}$ imposes additional conditions for FOL-rewritability. Specifically, we show that we additionally need to assume the $(infinitediff)$ and $(opendomain)$ conditions.

Similarly to the previous framework, the proof of necessity of these conditions is based on reductions of coNP-hard problems to $UCQ_\mathcal{D}$ answering in *DL-Lite$_d$+D*, where D violates at least one of the conditions in $\mathcal{D}_2$.

Again we distinguish two cases. In the first case we reduce 2+2-CNF unsatisfiability to the cases of $CQ_\mathcal{D}$ answering over KBs with datatype hierarchies respectively satisfying $\bigcap_i T_i^D \subseteq T_1^D \cup T_2^D$, $|\bigcap_i T_i^D \setminus T_1^D| = m$, $D = \langle\{T_1\}, \bigcup_{i=1}^m \{u_i\}\rangle$, or $D = \langle\{T_1, T_2\}, \emptyset\rangle$ (for some $m \geq 1$). In the second case we reduce k-coloring (for $k \geq 3$) to $UCQ_\mathcal{D}$ answering over KBs with datatypes satisfying resp. $D = \langle\{T_1, \ldots, T_k\}, \emptyset\rangle$, $D = \langle\{T_1, \ldots, T_{k-1}\}, \bigcup_{i=1}^m \{u_i\}\rangle$, $\bigcap_i T_i^D \subseteq \bigcup_{j=1}^k T_j^D$, or $|\bigcap_i T_i^D \setminus \bigcup_{j=1}^{k-1} T_j^D| = m$.

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

We now turn to FOL-rewritability of the framework *DL-Lite$_d$+$\mathcal{D}_2$+$UCQ_\mathcal{D}$*, 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_* is selected such that $d_* \in \bigcap_{B_i \sqsubseteq \forall U.T_i \text{ is activated}} T_i^{D'}$ and for any m datatypes $T_1, \ldots, T_m$ in D' s.t. $\bigcup_{j=1}^{m} T_j^{D'} \subsetneq \bigcap_{B_i \sqsubseteq \forall U.T_i \text{ is activate}} T_i^{D'}$ it holds that $d_* \notin \bigcup_{j=1}^{m} T_j^{D'}$. Notice that such selection is possible because of the $(infinitediff)$ 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 $\mathcal{D}_2$, $\mathcal{K}$ a satisfiable DL-Lite$_d$+D KB, and $\mathcal{M}$ a model of $\mathcal{K}$. Then, there exists a homomorphism from $can(\mathcal{K})$ to $\mathcal{M}$ over the set of concept, role, attribute, and datatype symbols of $\mathcal{K}$.*

3. Satisfiability. Considering that *DL-Lite*$_d$+$\mathcal{D}_2$ is a special case of *DL-Lite*$_d$+$\mathcal{D}_1$ (the second framework is more general only in the query language), it follows that satisfiability of *DL-Lite*$_d$+$\mathcal{D}_2$ is FOL-rewritable.

4. Perfect rewriting. Given that the datatype predicates are taken into account in the homormophism from $can(\mathcal{K})$ to a model $\mathcal{M}$, we can follow the same line of reasoning as in the previous framework and show that $cert(q, \mathcal{K}) = q^{can(\mathcal{K})}$, for an arbitrary $UCQ_{\mathcal{D}}$ q.

We recall that we consider only safe $UCQ_{\mathcal{D}}$s. In order to deal with datatype predicates in the input $UCQ_{\mathcal{D}}$, we need to further extend the PerfectRef algorithm presented in Section 4, with the following part:

for all $U(x, \__i) \in q$ **do**
  if $T^{\mathcal{T}}_{max}(U) \sqsubseteq \mathsf{Rng}^q(U(x, \__i))$ **then**
    $q'_{pr} \leftarrow q'_{pr} \cup \mathsf{deleteRng}(q, \__i)$;
  end if
end for

Here, we have denoted with '$\__i$' (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, $\mathsf{Rng}^q(U(x, \__i))$ denotes the conjunction of datatype predicates containing $\__i$, and the method $\mathsf{deleteRng}(q, \__i)$ creates a new $CQ_{\mathcal{D}}$ from q by discarding all datatype atoms containing $\__i$.

Theorem 9 *Let D be a datatype hierarchy in $\mathcal{D}_2$, $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ a satisfiable DL-Lite$_d$+D KB, and q a $UCQ_{\mathcal{D}}$ over $\mathcal{K}$. Then*

$$(\mathsf{PerfectRef}(q, \mathcal{T}))^{DB(\mathcal{A})} = cert(q, \mathcal{K}).$$

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 FOL-rewritability, if we assume that only datatype predicates (facets, constraints) but no datatypes appear in the query. Unfortunately, the answer is negative. Indeed, our coNP-hardness reductions (in the cases of $(infinitediff)$ and $(opendomain)$) essentially use only super type(s) ($\bigcap_i T_i$) in the terminological part, while subtypes ($\bigcup_{j=1}^{k} T_j$) 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 $\mathcal{D}_2$ 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:

$D_2^{\circ_q} := \langle Q \cup \{Q[\circ_q] \mid q \in \mathbb{Q}\}, \{u_i\}_i^{\infty} \rangle$, for $\circ_q \in \{\leq_q, \geq_q, <_q, >_q\}$,
$D_2^{\circ_n} := \langle Z \cup \{Z[\circ_n]\}, \{u_i\}_i^{\infty} \rangle$, for $\circ_n \in \{\leq_n, \geq_n, <_n, >_n \mid n \in \mathbb{Z}\}$.

■

6 CONCLUSION

Table 2. Data complexity of query answering in *DL-Lite* datatype frameworks.

	DL-Lite$_{\mathcal{A}}$	*DL-Lite*$_d$ $\mathcal{D}_1$	*DL-Lite*$_d$ $\mathcal{D}_2$
UCQ	AC^0	AC^0	AC^0
$UCQ_{\mathcal{D}}$	coNP	coNP	AC^0

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 $(infinite)$ condition and where in addition the considered datatypes are pairwise disjoint. The coNP lower-bound for $UCQ_{\mathcal{D}}$ answering is a consequence of the violation of $(opendomain)$. The set of datatypes supported in OWL 2 QL satisfy the conditions of the $\mathcal{D}_1$ datatype class (in fact, much stricter conditions), and we have shown here that such conditions are necessary and sufficient to ensure FOL rewritability of CQ answering (both in OWL 2 QL and in *DL-Lite*$_{\mathcal{A}}$). Moreover, by considering queries with datatype predicates ($UCQ_{\mathcal{D}}$), the conditions for the stricter $\mathcal{D}_2$ 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 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

[1] ISO/IEC 11404:2007, 'Information technology: General-Purpose Datatypes (GPD)', Technical report, ISO/IEC, CH-1211 Geneva 20, Switzerland, (2007).

[2] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*, Addison Wesley Publ. Co., 1995.

[3] A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev, 'The *DL-Lite* family and relations', *J. of Artificial Intelligence Research*, **36**, 1–69, (2009).

[4] A. Artale, Y. A. Ibanez-Garcia, R. Kontchakov, and V. Ryzhikov, '*DL-Lite* with attributes and sub-roles (extended abstract)', in *Proc. of DL 2011*, volume 745 of *CEUR*, `ceur-ws.org`, (2011).

[5] A. Artale, R. Kontchakov, and V. Ryzhikov, '*DL-Lite* with attributes and datatypes', in *Proc. of ECAI 2012*, (2012).

[6] F. Baader, S. Brandt, and C. Lutz, 'Pushing the $\mathcal{EL}$ envelope', in *Proc. of IJCAI 2005*, pp. 364–369, (2005).

[7] *The Description Logic Handbook: Theory, Implementation and Applications*, eds., F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, Cambridge University Press, 2nd edn., 2007.

[8] F. Baader and P. Hanschke, 'A schema for integrating concrete domains into concept languages', in *Proc. of IJCAI'91*, pp. 452–457, (1991).

[9] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodríguez-Muro, and R. Rosati, 'Ontologies and databases: The *DL-Lite* approach', in *5th Int. Reasoning Web Summer School (RW 2009)*, volume 5689 of *LNCS*, 255–356, Springer, (2009).

[10] F. M. Donini, M Lenzerini, D. Nardi, and A. Schaerf, 'Deduction in concept languages: From subsumption to instance checking', *J. of Logic and Computation*, **4**(4), 423–452, (1994).

[11] D. Magka, Y. Kazakov, and I. Horrocks, 'Tractable extensions of the description logic $\mathcal{EL}$ with numerical datatypes', *J. of Automated Reasoning*, **47**(4), 427–450, (2011).

[12] B. Motik and I. Horrocks, 'OWL datatypes: Design and implementation.', in *Proc. of ISWC 2008*, pp. 307–322, (2008).

[13] O. Savkovic, *Managing Data Types in Ontology-based Data Access*, Master's thesis, KRDB Res. Centre, Free Univ. of Bozen-Bolzano, 2011.