Freie Universität Bozen
Libera Università di Bolzano
Free University of Bozen - Bolzano

Technische Universität Wien
Vienna University of Technology

Fakultät für Informatik
Technische Universität Wien
Faculty of Computer Science
Free University of Bozen-Bolzano

European Master Program in Computational Logic
Master Thesis

Managing Datatypes in Ontology-Based Data Access

by

Ognjen Savković

born on July 12^{th} 1985, Belgrade, Serbia

Supervisor: **Prof. Diego Calvanese**

October, 2011

Acknowledgements

First of all, I would like to thank to my supervisor Prof. Diego Calvanese for his expert guidance, excellent comments and ideas that make this work possible.

Also, I wish to thank to Prof. Steffen Hölldobler for his effort on leading the EMCL program, and providing me the significant funding, that enables me to study and live two wonderful years in Vienna and Bolzano.

Most importantly, thanks to my family, to my wife Biljana and to my son Konstantin, for their great love and inspiration, and unrestricted support throughout my life. They unselfishly followed me on my study course and allowed me to share with them all my all goods and bads on my academic path.

Abstract

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 [10]. This scenario is called *ontology-based data access* (OBDA), since the major concern is the efficient access to data through an ontology. Two major inference tasks are of importance in OBDA, namely checking consistency of data wrt an ontology, and answering queries by taking into account the ontology. An important issue is whether the ontology layer introduces significant overhead in dealing with the data, and a key desiderata is that the major inference tasks are not harder (when measured in the size of the actual data) than they would be if the ontology layer were not present. This property is ensured in the presence of so-called *FOL-rewritability*. For this purpose, several DLs have been proposed, and they are grouped under the *DL-Lite* family of logics [7].

In the OBDA scenario, the issues related to the use of actual datatypes (such as those adopted in database management systems) essentially have been neglected. This thesis aims to overcome this restriction and studies 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 lattice, constituted by a set of datatypes that can be freely defined using the available constructors. We classify datatype lattices according to a hierarchy of three classes ($\mathcal{D}_0 \supset \mathcal{D}_1 \supset \mathcal{D}_2$), based on the conditions that are satisfied by the datatype lattice. 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 $\mathsf{UCQ}_{\mathcal{D}}$, which is obtained by extending standard UCQs with datatype constraints. In this theoretical framework we distinguish between three major components ($\mathcal{DL} + \mathcal{D} + \mathcal{Q}$). The first component is constituted by the ontology language $\mathcal{DL}$, and includes also the interface for combining ontologies with datatype lattices. The second component is the class $\mathcal{D}$ of datatype lattices considered. The last component is a query language $\mathcal{Q}$, over ontologies and datatype lattices ($\mathcal{DL} + \mathcal{D}$). The main technical contribution of our work consists in studying the properties of two important instances of the OBDA framework, namely, $\textit{DL-Lite}_{core}^{(\mathcal{HF})} + \mathcal{D}_1 + \mathsf{UCQ}$ and $\textit{DL-Lite}_{core}^{(\mathcal{HF})} + \mathcal{D}_2 + \mathsf{UCQ}_{\mathcal{D}}$. 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 lattices are necessary to preserve FOL-rewritability of query answering.

Dedicated to Biljana and Konstantin.

Contents

Chapter 1

Introduction

1.1 Description Logics

Description logics [7] (DLs) are a family of logic-based languages that are based, on the one hand, on the idea of formal knowledge representation that employs reasoning methods, and on the other hand, on the idea of describing a domain of interests using more intuitive languages like semantical networks or UML.

The formal face of DLs is rooted in the area of Knowledge Representation and Reasoning (KRR), a significant research field within Artificial Intelligence (AI). KRR explores logic based approaches for storing the knowledge about a given domain of discourse, and reasoning methods of inferring implicit consequences from the explicitly represented knowledge. Semantics of DLs are based on the FOL semantics. There is a great variety of different DL languages that are designed by taking into account the trade-off between reasoning complexity and description capabilities.

The 'humanistic' face of DLs is rooted in Computer Science (CS) and comes from a need to describe formal knowledge (machine readable) in a way intuitive for humans, often refereed as Ontologies. Ontologies are used to formally describe the domain of interest by means of concepts (classes), which are unary predicates that assemble objects of common properties, and roles (relationships), which are binary predicates that establish connections between objects. To describe object properties of objects, ontologies use attributes, which are binary predicates, that relate objects to datatype constants. Ontologies use standard constructors of conceptual modeling to describe dependencies between concepts, roles and attributes. Usually, the list of constructors includes is-a hierarchies (i.e., inclusions), disjointness for concepts, roles and attributes, domain and range constraints for roles and attributes, mandatory participation in roles and attributes, functionality constraints, general numerical restrictions for roles and attributes, etc.

An Ontology, also called as a Knowledge Base (KB), is described by a finite set of terminological assertions (inclusion assertions or axioms) called TBox and a finite set of facts, called ABox (extensional knowledge), about individual objects, such as whether an object is an instance of a concept, whether two objects are connected by a role, or whether an object is connected to a datatype constant by an attribute. Terminological assertions impose restrictions that all

object of the ontology have to obey. For example,

$$Student \sqsubseteq \exists hasSupervisor.Professor$$

states that each object of a type *Student* has a supervisor that is a professor. *Student* and *Professor* represent concepts that contain students and professors respectively. *hasSupervisor* represents a role that connects a student with his/her supervisor.

Similarly, one can state a fact about an object in an ontology,

$$hasEmail(\mathsf{John}, \text{"}\mathsf{john@inf.unibz.ac.at}\text{"})$$

where the attribute *hasSupervisor* is used to state the fact that the object John has an email "john@inf.unibz.ac.at".

Applications of the ontologies includs areas like: *Conceptual Modeling* [9, 3] (e.g. UML or ER diagrams), *information and data integration* [12, 11], *Ontology-based data access* [19, 21] and the Semantic Web [1]. OWL (Web Ontology Language) [2] is a W3C ontology language standard initially developed for use in the semantic web and now widely used both academically and commercially.

1.2 Motivation

1.2.1 Motivation for *DL-Lite*

The standard reasoning services over an ontology include checking its consistency (or ontology satisfiability), instance checking (whether a certain individual is an instance of a concept), and logic entailment (whether a certain constraint is logically implied by the ontology. There have been many proposed DL languages that successfully and efficiently implement these scenarios.

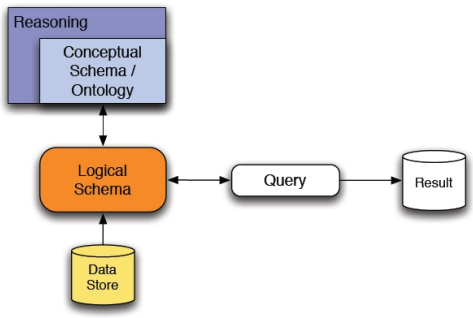


Figure 1.1: Conceptual schema used at a design phase

However, in most of those languages the size of the data (ABox) and the size of the constraints (TBox) are considered to be approximately the same, and they are not aimed at working with a large amounts of data.

Note that application like Information and Data Integration, the Semantic Web, and most importantly ontology-based data access (OBDA), can be very data intensive, which means that the size of data significantly dominates the size of the terminological part.

[1] http://www.w3.org/standards/semanticweb/
[2] http://www.w3.org/TR/owl2-overview/

The idea is to use ontologies as a high-level, conceptual view over data repositories, allowing users to access data items without the need to know how the data is actually organized and where it is stored. Besides providing the conceptual view, an ontology serves the role of imposing constraints, i.e., any legal data instance has to an conform with the ontology.

In the literature, those scenarios are named ontology-based data access (OBDA) scenario, since their major concern is the efficient access to data through an ontology.

The *DL-Lite* family of DLs is aimed to be used in describing conceptual models effectively balancing between expressive power on the one side and the computational complexity of the reasoning services on the other side. Notice that the more expressive a language is the higher is the computational complexity of the inference tasks. The reasoning services are inference methods that should be sound and complete wrt reasoned problem.

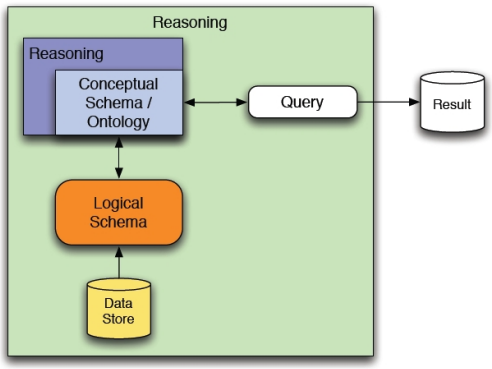


Figure 1.2: Ontology used in a run time

Conceptual models are formal conceptualization of a domain of interest. For example, in the context of data integration where a DL is used to describe dependences between sources and possible data incompleteness. The expressive power is reflected in the capability to formally capture the intensional meaning of the described domain. It was shown that *DL-Lite* family of DLs can be used to formalize modeling languages like UML class diagrams or Entity-Relationship diagrams [10].

In the common settings, conceptual schema are only used in the design phase. Once a logical schema is built, one can start querying the underlying data repository (Fig. 1.1), end, the conceptual schema is not considered any more.

The idea with OBDA, is to present a user a conceptual view of the data, that is more "human" readable and at the same time to wrap (hide) the real structure and organization of the data. One benefit is that OBDA can simplify maintenance of the information system, as the user only considers the conceptual model. In addition to, for example views in database systems, OBDA is semantically based, which facilitates formal integration with other systems. Finally, the OBDA successfully deals with incomplete information.

The goal is to allow answering queries to the ABox with the constraints in the TBox taken into account (Fig. 1.2) .The queries that are considered are First-order Logic (FOL) conjunctive queries (CQs), which correspond to the commonly used `Select-Project-Join` SQL queries. The idea is to benefit from already existing highly optimized relational technology for query answering. That is achieved by rewriting a query into SQL considering only the ter-

minological part (TBox), and evaluating the rewritten SQL query against the extensional part stored in a relational database.

The notion of FOL-rewritability of CQ answering ensures the desired property, that by rewriting into SQL are obtains a query that when evaluated over the extensional data only, returns the certain answers to the original query. Unfortunately, not every DL will enjoys FOL-re and hence is not suitable OBDA. Most commonly a DL will need to rewrite a CQ query into a language of higher expressibility than FOL queries (f.e. recursive datalog). Consequently, we can't benefit from existing relational database technology.

The DLs that allows for FOL-rewritability of CQs are logics from the *DL-Lite* family. Moreover, the property will hold even for Union of Conjunctive Queries (UCQs).

Recently, the W3C consortium had defined OWL 2 QL [3] , as a one of three profiles in OWL 2, that is designed for applications that are data extensive and for which query answering is the major task. The OWL 2 QL profile is a syntactical restriction on the full OWL 2 language and corresponds to $DL\text{-}Lite_{\mathcal{R}}$, a *DL-Lite* family member.

1.2.2 Datatypes in the *DL-Lite* family

DL-Lite family members have been widely investigated [4] and more than 40 different logics of this family have been introduced. They are structured in the taxonomy, considering expressibility and their computational properties.

However, no significant attention has been made towards introducing datatypes in the *DL-Lite* family. We distinguish three issues relating datatypes in DLs

- The first one in an interface that a DL language provides to establish the connection between datatypes and other ontology constructors. This includes for example, constructor to a range restriction over an attribute:

  $$\mathsf{Rng}(hasName) \sqsubseteq \mathsf{xsd} : \mathsf{string}$$

  it constrains an attribute *hasName*, so that the second component can only by of type xsd : string, and not xsd : integer for example.

  Seeing more widely, a DL interface towards datatypes includes addressing datatypes in all reasoning methods related to the DL. For instance, we can consider CQ answering with numerical restrictions:

  $$q(x) : -\exists y.Student(x), hasAge(x, y), \geq_{26} (y)$$

  A CQ q should return students that are twenty-six years old or older.

- Secondly, an ontology language in addition to a language for modeling objects, should provide a language for modeling datatypes. For example, creating new datatypes using the defined datatypes or by enumerating the datatype elements. Additionally, most of the standard datatypes, like integers and strings, contain predefined constructors (*facets*) which defines restrictions over the original datatypes. For example, numerical datatypes provide facets like $\{<_n, \leq_n, \geq_n, >_n, \neq_n\}$, $maxlen(n)$ (the maximal number

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

of digits), $minlen(n)$ (the minimal number of digits), while string datatype is commonly restricted using regular expressions. An example of facets application could be:

$$\mathsf{dt : adultAge} ::= \mathsf{xsd : integer}[\geq_{18}],$$

$$\mathsf{dt : emailAddress} := \mathsf{xsd : string}[\mathsf{RegExpr}([0-9A-Z] + @[0-9A-Z.]+)],$$

where dt : adultAge is a new datatype defined as a restriction on integers that are greater or equal 18, and dt : emailAddress is a new datatype that admits only correctly formated email addresses.

The goal is to utilize existing facets functions and to introduce other datatype constructors, to obtain a comprehensive datatype modeling language, that adopts common standards in creating datatypes.

- Datatypes assigned to a DL will possess certain characteristics. For example, values from datatype xsd : decimal are contained in datatype owl : real, while owl : real and xsd : string will have no common values.

  We realize that internal relations between datatypes have consequences on reasoning methods over the ontology. For example,

  $$Professor \sqsubseteq \exists officeNum.\mathsf{xsd : string} \quad \mathsf{Rng}(officeNum) \sqsubseteq \mathsf{xsd : nonNegativeInteger}$$

  will imply that $Professor \sqsubseteq \bot$ considering that, $\mathsf{xsd : string} \sqcap \mathsf{xsd : nonNegativeInteger} \sqsubseteq \bot$. Depending on the reasoning method, datatypes can cause more sophisticated issues. For instance, if we have a finite datatype $\mathsf{dt : colors} := \{\mathsf{red}, \mathsf{blue}, \mathsf{green}\}$, then using inclusions

  $$Vartex \sqsubseteq \exists hasColor, \quad \mathsf{Rng}(hasColor) \sqsubseteq \mathsf{dt : colors}, \quad (\mathsf{funct}\ hasColor)$$

  we can simulate graph coloring. Then we pose a CQ

  $$q() : -\exists x_1, x_2, y.Edge(x_1, x_2), hasColor(x_1, y), hasColor(x_2, y)$$

  over the ontology that contains graph edges and vertices. If the answer is *true* it means that for every positive extension of the ABox (that includes color assignments as well) we have adjustment vertices with the same color. In other words, the query answering problem is at least as complex as 3-colorability, a computationally hard problem.

  In this sense, merging datatypes with a certain DL has to be done carefully. In the best case, we would like to characterize necessary and sufficient conditions over datatypes wrt reasoning methods so that the complexity remains in the assumed boundaries.

In the *DL-Lite* family datatypes were introduced with *DL-Lite*$_\mathcal{A}$, with a strong condition that datatypes assigned to a *DL-Lite*$_\mathcal{A}$ logic are pairwise disjoint (over the values). Practically, it means that we can't have owl : real and xsd : integer in a *DL-Lite*$_\mathcal{A}$ ontology, because xsd : integer $\sqsubseteq$ owl : real. The attribute interfaces allow range restriction for an attribute (e.g., $\mathsf{Rng}(hasSalary) \sqsubseteq \mathsf{xsd : nonNegativeInteger}$), mandatory attribute participation (e.g., $Professor \sqsubseteq \exists hasSalary$), and typed attribute participation (e.g., $Professor \sqsubseteq \exists hasSalary.\mathsf{xsd : nonNegativeInteger}$).

Recently a *DL-Lite* logic named $DL\text{-}Lite_{core}^{\mathcal{HNA}}$ [5] has been proposed, with the aim of extending interface that connects datatypes and the abstract components of the logic. In addition to the $DL\text{-}Lite_{\mathcal{A}}$ interface, it provides constructors for expressing that two datatypes are disjoint (e.g., $T_1 \sqcap T_2 \sqsubseteq \bot$), that one is a subtype of another one (e.g., $T_1 \sqsubseteq T_2$), that an attribute is universally restricted for a datatype (e.g., $B \sqsubseteq \forall U.T_i$) and that a datatype constant belongs to a datatype (e.g., $T_i(v_j)$).

However, datatypes in $DL\text{-}Lite_{core}^{\mathcal{HNA}}$ can not be considered predefined , and from the logic point of view can be treated as special concepts, with the restriction that only attributes can range over them (more discussion on this in the section 3.1)

We argue, that there is an essential semantical difference between datatype constant and objects constants. The meaning (interpretation) of a datatype constant is fixed and given in advance (predefined), while object constants can change their semantics depending on the interpretation. For example, an assertion $hasAge(\mathsf{John}, \text{"25"}\hat{}\hat{}\mathsf{xsd:int})$ [4] states that object John is 25 years old. While an object John has potential to be considered (interpreted) as any person with name John (e.g., as a professor or as a student), "25"$\hat{}\hat{}$xsd : int has to be considered (interpreted) as the integer number twenty-five.

1.3 Aims and Results

The main aim of the thesis is to propose a language for describing datatypes and to introduce such a language in the *DL-Lite* family, while exploring the computational effects on the desired properties. The main computational properties of our concern are FOL-rewritability of ontology satisfiability and FOL-rewritability of query answering. Concerning the query language we are not binding ourselves to UCQs only, and we consider more expressive query languages that includes datatypes restrictions.

The contributions of the thesis include:

- We introduce a language over datatypes, aimed to define new datatypes from the already existing ones. The language defines the syntax and semantics of a datatype, that in a such a way that is compatible with FOL semantics. New datatypes can be defined as sub-types by constraining functions (facets) defined over the original datatypes. Additionally, a new datatype can be built combining different datatypes using predefined constructors.

  Finally, we define the notion of datatype lattice, a set of datatypes, that can be freely defined using the available constructors. that is self contained wrt the definitions of datatypes. We classify datatype lattice, in three classes named $\mathcal{D}_0$, $\mathcal{D}_1$, and $\mathcal{D}_2$ ($\mathcal{D}_0 \supset \mathcal{D}_1 \supset \mathcal{D}_2$), based on the conditions that a datatype lattice satisfies or not.

- We define a theoretical framework for OBDA and identify three major components ($\mathcal{DL} + \mathcal{D} + \mathcal{Q}$). The first component is a language $\mathcal{DL}$, that

[4] "*literal_sting*"$\hat{}\hat{}$datatype_name is a RDF format for presenting typed constants (literals) in a RDF ontology.

describes the ontology language as well as an interface for combing ontologies with datatype lattices. The second component is the class $\mathcal{D}$ of datatype lattices assigned to the $\mathcal{DL}$. The Last component is a query language $\mathcal{Q}$.

Eligible candidates for the OBDA framework will be triples $\mathcal{DL} + \mathcal{D} + \mathcal{Q}$ that are FOL-rewritable, i.e., where given a query from $\mathcal{Q}$ we can rewrite it in a FOL query, called perfect reformulation, using only the terminological part of an ontology. Answers of the perfect reformulation evaluated against ABox should match the answers of the original query evaluated over the ontology (model theoretical approach, Def., 12).

- Finally we present two significant instances of the OBDA framework. The first instance is $\mathit{DL\text{-}Lite}_{core}^{(\mathcal{HF})} + \mathcal{D}_1 + \mathsf{UCQ}$. For the second instance, $\mathit{DL\text{-}Lite}_{core}^{(\mathcal{HF})} + \mathcal{D}_2 + \mathsf{UCQ}_{\mathcal{D}}$, we explored adding datatypes in a query language. Specifically, we extend the standard UCQs with datatype constraints, and obtain a query language $\mathsf{UCQ}_{\mathcal{D}}$. Most importantly, for both instances we proved FOL-rewritability of query answering. Conversely, we show that in both cases the conditions over datatype lattices are necessary to preserve FOL-rewritability.

  In other words, if we allow datatype lattices of type $\mathcal{D}_0$ in the case of $\mathit{DL\text{-}Lite}_{core}^{(\mathcal{HF})} + \mathcal{D}_1 + \mathsf{UCQ}$ then the OBDA framework is not FOL-rewritable any more. The same holds in the case of $\mathit{DL\text{-}Lite}_{core}^{(\mathcal{HF})} + \mathcal{D}_2 + \mathsf{UCQ}_{\mathcal{D}}$ if the datatype lattices are of type $\mathcal{D}_0$ or $\mathcal{D}_1$ instead of $\mathcal{D}_2$.

1.4 Structure of the thesis

In the following we describe the structure of the rest of the thesis.

Chapter 2: Preliminaries. In this chapter, we introduce basic definitions regarding computational complexity. Also we present the syntax and semantics of FOL queries. Afterwards, we introduce the most common constructors in the *DL-Lite* family and we define their semantics. Then, we present in more detail CQs and UCQs, together with their basic properties. Finally, we formally define the notion of query answering over ontologies by means of certain answers.

Chapter 3: Datatypes. This chapter contains two parts. In the first part we are discussing about datatypes from two points of view. One is a brief history of datatypes in DLs in general. The other one, is a brief review of datatypes in semistructured languages like XML, RDF and OWL. In the second part, we introduce a formal language over datatypes, that enables creating new datatypes from existing ones using a comprehensive set of constructors. We also provide a semantics for the language constructors. Finally, we introduce the notion of datatype lattice, a set of datatypes, that contains declarations of primitive (ab initio) datatypes together with the definitions of datatypes derived from them. Also we classify datatype lattices according to certain restrictions over them.

Chapter 4: OBDA framework. This chapter identifies the theoretical bases for the OBDA scenario. Firstly, we present a *DL-Lite* common interface for connecting ontology elements (in particular attributes) with datatypes (denoting

it $\mathcal{DL} + \mathcal{D}$), and we define its semantics. Secondly, we extend the definitions of certain answers and FOL-rewritability to a three component framework, denoted with $\mathcal{DL} + \mathcal{D} + \mathcal{Q}$. Here, $\mathcal{DL}$ stands for a DL language, $\mathcal{D}$ stands for a datatype class, and $\mathcal{Q}$ stands for a query language. Lastly, we introduce new query language, called Union of Conjunctive Queries with datatypes ($\mathsf{UCQ}_{\mathcal{D}}$), that extends UCQs. We provide a formal semantics for it.

Chapter 5: OBDA framework *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1 + \mathsf{UCQ}$. This chapter contains two parts. In the first part, we are proving that *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1 + \mathsf{UCQ}$ is FOL-rewritable. This is done in two steps. Firstly, by showing that satisfiability is FOL-rewritable in *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$. And secondly, by constructing the perfect reformulation of a UCQ for a satisfiable ontology. In the second part, we are proving that the conditions over datatype lattices in *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1 + \mathsf{UCQ}$ are necessary. In other words, FOL-rewritability of *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1\mathsf{UCQ}$ will be lost if we violate them.

Chapter 6: OBDA framework *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_2 + \mathsf{UCQ}_{\mathcal{D}}$. In this chapter we take an approach that is similar to the one in the previous chapter. FOL-rewritability of satisfiability of *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ ontologies directly follows more general case of *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ satisfiability. Then we extend the algorithm , that for a given $\mathsf{UCQ}_{\mathcal{D}}$ constructs a perfect reformulation over a satisfiable *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ ontology. Finally, we show that the conditions over the datatype lattices in $\mathcal{D}_2$ are necessary in order to have FOL-rewritability of query answering in *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_2 + \mathsf{UCQ}_{\mathcal{D}}$.

Chapter 7: Conclusions and Future work. This last chapter presents some conclusions of the work carried out in this thesis. It includes also some future paths of research on the topics concerning the thesis.

Chapter 2

Preliminaries

In the following chapter we introduce the syntax and semantics of FOL queries, together with the common constructors in *DL-Lite* family. Then we present in a more detail CQs and UCQs. Finally, we define the notion of query answering over ontologies by means of certain answers. We start with computational complexity that appears in all notions listed above.

2.1 Computational Complexity

In this thesis, we will make use of some complexity classes and their properties. We will assume that a reader is familiar with the basic notions of computational complexity. For the formal definitions of computational complexity classes we refer to some standards textbooks on computational complexity, like [18].

Complexity classes that are important for the *DL-Lite* family are presented in the relationship chain:

$$\mathrm{AC}^0 \subsetneq \textsc{LogSpace} \subseteq \textsc{NLogSpace} \subseteq \textsc{PTime} \subseteq \mathrm{NP} \subseteq \textsc{ExpTime}$$

where $\subseteq$ states that it is not known whether an inclusion is strict, while $\subsetneq$ states that an inclusion is strict. In addition, it is known that $\textsc{PTime} \subsetneq \textsc{ExpTime}$. For *DL-Lite*, also important class is coNP, the class of problems that are complement to the problems in NP.

Classes that are not often referenced are AC^0, LogSpace, and NLogSpace. We will try to introduce them in an informal way.

A (decision) problem belongs to LogSpace if it can be decided by a two-tape (deterministic) Turing machine that receives its input on the read-only input tape and uses a number of cells of the read/write work tape that is at most logarithmic in the length of the input. The complexity class NLogSpace is defined analogously, except that a nondeterministic Turing machine is used instead of a deterministic one. Two most prominent problem that are in NLogSpace-complete but not in AC^0 are directed graph reachability (or *st-reachability*) and 2-SAT.

A LogSpace reduction is a reduction computable by a three-tape Turing machine that receives its input on the read-only input tape, leaves its output on the write-only output tape, and uses a number of cells of the read/write work tape that is at most logarithmic in the length of the input.We observe that

most reductions among decision problems presented in the computer science literature, including all reductions that we present here, are actually LogSpace reductions.

For the formal definition of AC^0 we refer the reader to [23]. Here we will describe basic intuition. A problem is in AC^0 if it can be decided in a constant time using a number of processors that is polynomial in the size of the input. In order words, a problem can be efficiently parallelized. The most prominent AC^0 member, is the evaluation of First-Order Logic (FOL) queries over relational databases, where only the database is considered to be the input, and the query is considered to be of a fixed size.

This is the main assumption of relational database effectiveness, because SQL queries (without aggregation) correspond to FOL queries. The key of the assumption is that the SQL query size is neglectable comparing to the size of the database. Moreover, the same assumption is applied for Ontology-Based Data Access framework.

On the other hand, we will often exploit the fact, whenever a problem is shown to be a hard problem for a complexity class that strictly contains AC^0 (e.g. NLogSpace), then it cannot be reduced to the evaluation of First-Order Logic queries.

2.2 Query answering in databases

Nowadays, relational databases are de facto standard for storing and querying large amounts of data. In this sense, we would like to define a simple logic formalism for relational representation. For a moment we will assume that all constants and values that appears in a database instance (shorter instance) are from a common domain (a universe). In addition we need to define a *query language*, a language for declaration of *queries*. A query should be a deceleration that specifies what to extract from a database, but not and how. [1] For our purposes, First Order Logic (FOL) queries will be the most expressive query language that we consider. We define basic notations.

First-order logic queries. We assume a fixed vocabulary $\mathcal{V}$ that contains infinite sets of variables $Vars = \{x_1, x_2, \ldots\}$, constants $Const = \{c_1, c_2, \ldots\}$, functional symbols $Fun = \{f_1, f_2, \ldots\}$, and predicate symbols $Pred = \{P_1, P_2, \ldots\}$. For each natural number n we have infinitely many functional symbols and predicate symbols of arity n. In addition we have logical connectors $\neg$ ("negation"), $\wedge$ ("and"), $\vee$ ("or"), $\rightarrow$ ("imply") and $\leftrightarrow$ ("equal") and two quantifiers $\forall$ ("for all") and $\exists$ ("exists").

We define sets of *Terms* and *Formulas* recursively. *Terms* contains all variables and constants. If $t_1, \ldots, t_n \in Terms$ then also $f(t_1, \ldots, t_n) \in Terms$. Let $t_1, \ldots, t_n \in Terms$ and P a predicate symbol of arity n. Then $P(t_1, \ldots, t_n) \in Formulas$. If $\phi, \psi \in Formulas$ then also $\neg\phi$, $\exists x.\phi$, $\forall x.\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \rightarrow \psi$, $\phi \leftrightarrow \psi \in Formulas$, where x is a variable that appears in ϕ and it is not quantified (no $\exists x$ nor $\forall x$ appears in ϕ).

A FOL *interpretation* over a vocabulary $\mathcal{V}$ is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is an *interpretation domain* (a set of objects, the universe) and $\cdot^{\mathcal{I}}$ is an *interpretation function* that interprets predicates and function symbols s.t. for

[1] The most of the material in this section is adopted from the prof. Calvanese lecture notes on the course Knowledge Representation and Ontologies., academic year 2010/2011

each constant $c : c^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, for each function f with arity n, $f^{\mathcal{I}} : (\Delta^{\mathcal{I}})^n \to \Delta^{\mathcal{I}}$, and for each predicate P with arity n, $P^{\mathcal{I}} \subseteq (\Delta^{\mathcal{I}})^n$.

We define a *variable assignment* $\alpha : Vars \to \Delta^{\mathcal{I}}$ that maps all variables into the universe. Naturally, α can be extended to an arbitrary term with $\alpha(c) = c^{\mathcal{I}}$ and $\alpha(f(t_1, \ldots, t_n)) = f(\alpha(t_1), \ldots, \alpha(t_n))^{\mathcal{I}}$.

Finally we say that a FOL formula ϕ is *true* in an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ wrt an assignment α (we write $\mathcal{I}, \alpha \models \phi$) if:

$\mathcal{I}, \alpha \models P(t_1, \ldots, t_n)$	if	$(\alpha(t_1), \ldots, \alpha(t_n)) \in P^{\mathcal{I}}$
$\mathcal{I}, \alpha \models t_1 = t_2$	if	$\alpha(t_1) = \alpha(t_2)$
$\mathcal{I}, \alpha \models \neg\phi$	if	$\mathcal{I}, \alpha \not\models \phi$
$\mathcal{I}, \alpha \models \phi \wedge \psi$	if	$\mathcal{I}, \alpha \models \phi$ and $\mathcal{I}, \alpha \models \psi$
$\mathcal{I}, \alpha \models \phi \vee \psi$	if	$\mathcal{I}, \alpha \models \phi$ or $\mathcal{I}, \alpha \models \psi$
$\mathcal{I}, \alpha \models \phi \to \psi$	if	$\mathcal{I}, \alpha \not\models \phi$ or $\mathcal{I}, \alpha \models \psi$
$\mathcal{I}, \alpha \models \phi \leftrightarrow \psi$	if	$\mathcal{I}, \alpha \models \phi$ iff $\mathcal{I}, \alpha \models \psi$
$\mathcal{I}, \alpha \models \exists x.\phi$	if	exists $a \in \Delta^{\mathcal{I}} : \mathcal{I}, \alpha[x \mapsto a] \models \phi$
$\mathcal{I}, \alpha \models \forall x.\phi$	if	for all $a \in \Delta^{\mathcal{I}} : \mathcal{I}, \alpha[x \mapsto a] \models \phi$

where $\alpha[x \mapsto a]$ is the same as α except it maps x to a.

Also we say that a variable x is *free* in a formula ϕ if $\exists x$ or $\forall x$ doesn't appear in ϕ. A formula ϕ is *open* if it contains some free variables, otherwise way say it is a *closed* formula.

Finally, we say that a ***FOL query*** is an (open) FOL formula. Now let $x_1, \ldots, x_n$ be free variables in ϕ then we denote an assignment α that is "relevant" for ϕ as $\langle a_1, \ldots, a_n \rangle$ which means that for each $i : \alpha(x_i) = a_i$. Given an interpretation $\mathcal{I}$, ***the answer to a query*** ϕ , denoted as $\phi^{\mathcal{I}}$, is:

$$\phi^{\mathcal{I}} := \{(a_1, \ldots, a_n) |\ \mathcal{I}, \langle a_1, \ldots, a_n \rangle \models \phi\}$$

Query evaluation. Assume that a DB instance is a finite set of tuples. In addition, a query ϕ contains only finitely many functional symbols and predicates. So for a moment we consider finite interpretation $\mathcal{I}$ (finite domain $\Delta^{\mathcal{I}}$). We define complexity of a problem $\mathcal{I}, \alpha \models \phi$ based on the input:

- *The combined complexity*, where interpretation, tuple, and query are considered as the input: $\{\langle \mathcal{I}, \alpha, \phi \rangle | \mathcal{I}, \alpha \models \phi\}$.
- *The data complexity*, where an interpretation and a tuple are considered as the input (query is fixed): $\{\langle \mathcal{I}, \alpha \rangle | \mathcal{I}, \alpha \models \phi\}$.
- *The query complexity*, where a tuple and a query are considered as the input (interpretation is fixed): $\{\langle \alpha, \phi \rangle | \mathcal{I}, \alpha \models \phi\}$.

2.3 Description Logic family *DL-Lite*

In the following we present common constructors of DL family *DL-Lite*, together with their semantics. Also we present CQs and UCQs in a more detail, together with their properties. After that we introduce common inference tasks in *DL-Lite*. Finally, we define the notion of FOL rewritability.

2.3.1 Syntax and Semantics

Logical expressions in *DL-Lite* family are carefully selected in order to express the main features of conceptual modeling while keeping low the computational complexity of the main reasoning tasks. The authors have defined more then forty dialect of *DL-Lite* family[4] and the they are still introducing new ones [5].

Syntax

We adopt the definitions from [10].

A ontology vocabulary $\mathcal{V}$ is a set of symbols that contains:

- Infinite set of concept and role symbols: $\Gamma_A^{\mathcal{V}} : A_1, A_2, \ldots$, $\Gamma_P^{\mathcal{V}} : P_1, P_2, \ldots$ respectfully.
- Infinite set $\Gamma_O^{\mathcal{V}}$ of object constants: $o_1, o_2, \ldots$

Using vocabulary elements we define more complex expressions. The notations are:

- A denotes an atomic concept, B a basic concept, C a general concept, and $\top_c$ the universal concept. An atomic concept is a concept denoted by a name. Basic and general concepts are concept expressions whose syntax is given at point 1 below.
- P denotes an atomic role, R a basic role, and Q a general role. An atomic role is simply a role denoted by a name. Basic and general roles are role expressions whose syntax is given at point 3 below.

1. Concept expressions are built according to the following syntax:

$$\begin{array}{lllllllll} B & \longrightarrow & A & | & \exists Q & & & & \\ C & \longrightarrow & \top_c & | & B & | & \neg B & | & \exists Q.C \end{array}$$

2. Role expressions are built according to the following syntax:

$$\begin{array}{lllll} R & \longrightarrow & P & | & P^- \\ Q & \longrightarrow & R & | & \neg R \end{array}$$

3. Attribute expressions are built according to the following syntax:

$$V \longrightarrow U \quad | \quad \neg U$$

Inclusion assertion allowed in *DL-Lite*$_{core}^{(\mathcal{HF})}$ are:

$$B \sqsubseteq C \qquad E \sqsubseteq F \qquad R \sqsubseteq Q \qquad U \sqsubseteq V$$

where *positive inclusions (PIs)* do not contain symbol '$\neg$'. Otherwise we call them *negative inclusions (NIs)*.

Functionality assertion in *DL-Lite*$_{core}^{(\mathcal{HF})}$ can be:

$$(\mathsf{funct}\ R)$$

Finally, *membership assertions* in *DL-Lite*$_{core}^{(\mathcal{HF})}$ that declares the instances of concepts, roles and attributes have the form:

$$A(o) \qquad\qquad P(o_1, o_2)$$

where A is an atomic concept, P is an atomic role, o, o_1 and o_2 are from $\Gamma_O^{\mathcal{V}}$. Inclusion assertions and functional assertion are also called *intensional assertions*, while membership assertions are called *extensional assertions*.

Definition 1. *A DL-Lite*$_{core}^{(\mathcal{HF})}$ *TBox is* $\mathcal{T}$ *is finite set of inclusion assertions and functional assertions, with condition that a role that appears in a functional assertion can not appear on the right hind side of some role assertion (can not be specialized). A DL-Lite*$_{core}^{(\mathcal{HF})}$ *ontology* $\mathcal{O}$ *is a pair* $(\mathcal{T}, \mathcal{A})$ *where* $\mathcal{T}$ *is DL-Lite*$_{core}^{(\mathcal{HF})}$ *TBox and* $\mathcal{A}$ *is a final set of membership assertions that each concept and role that appears in* $\mathcal{A}$*, appears in* $\mathcal{T}$ *as well.* $\mathcal{A}$ *is called DL-Lite*$_{core}^{(\mathcal{HF})}$ *ABox.*

Semantics

We define semantics in the term of First Order Logic interpretations.

Definition 2. *An DL-Lite*$_{core}^{(\mathcal{HF})}$ intepretation $\mathcal{I}$ *is a pair* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *where :*

- *As usual the interpretation domain* $\Delta^{\mathcal{I}}$ *is non empty set, called domain of objects.*
  - *for each object constant* $o_i \in \Gamma_O^{\mathcal{V}}$, $o_i^{\mathcal{I}} \in \Delta_O^{\mathcal{I}}$.
  - *for each two objects constants* $o_i, o_j \in \Gamma_O^{\mathcal{V}}$,

$$o_i \neq o_j \longrightarrow o_i^{\mathcal{I}} \neq o_j^{\mathcal{I}} \qquad \text{(UNA)}$$

  - $\cdot^{\mathcal{I}}$ *assigns domain elements to concepts and roles from vocabulary* $\mathcal{V}$ *s.t.*

$$A^{\mathcal{I}} \subseteq \Delta_O^{\mathcal{I}}, \qquad P^{\mathcal{I}} \subseteq \Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}}$$

  Based on that we evaluate other DL-Lite$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ *expressions in the following way:*

$$(\neg B)^{\mathcal{I}} = \Delta_O^{\mathcal{I}} \setminus B^{\mathcal{I}} \qquad (\exists Q)^{\mathcal{I}} = \{o | \exists o'.(o', o) \in R^{\mathcal{I}}\}$$
$$(\neg R)^{\mathcal{I}} = (\Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}}) \setminus R^{\mathcal{I}} \qquad (\exists R.C)^{\mathcal{I}} = \{o | \exists o'.(o', o) \in R^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\}$$
$$(P^-)^{\mathcal{I}} = \{(o, o') | (o', o) \in P^{\mathcal{I}}\}$$

Definition 3. *Let* $\mathcal{I}$ *be a DL-Lite*$_{core}^{(\mathcal{HN})}$ *interpretation:*

- *We define semantics for the intensional assertions*

$$\mathcal{I} \text{ satisfies } B \sqsubseteq C \qquad \text{if} \quad B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$$
$$\mathcal{I} \text{ satisfies } Q \sqsubseteq R \qquad \text{if} \quad Q^{\mathcal{I}} \subseteq R^{\mathcal{I}}$$

  and

$\mathcal{I}$ *satisfies* (funct P) *if* $\forall o^{\mathcal{I}}, o_1^{\mathcal{I}}, o_2^{\mathcal{I}} \in \Delta_O^{\mathcal{I}}.(o^{\mathcal{I}}, o_1^{\mathcal{I}}), (o^{\mathcal{I}}, o_2^{\mathcal{I}}) \in P^{\mathcal{I}} \rightarrow o_1^{\mathcal{I}} = o_2^{\mathcal{I}}$

- *For membership assertions* $\mathcal{I}$

$$\begin{array}{lll} \mathcal{I} \text{ satisfies } A(o) & \text{if} & a^{\mathcal{I}} \in A^{\mathcal{I}} \\ \mathcal{I} \text{ satisfies } P(o_1, o_2) & \text{if} & (o_1^{\mathcal{I}}, o_2^{\mathcal{I}}) \in P^{\mathcal{I}} \end{array}$$

- *Finally, we say that* $\mathcal{I}$ *satisfies (or is a model of it) DL-Lite*$_{core}^{(\mathcal{HF})}$ *ontology* $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ *if it satisfies all assertions in* $\mathcal{T}$ *and* $\mathcal{A}$.

We denote with $\mathcal{I} \models \alpha$ *for an assertion* α *(intensional or extensional) if* $\mathcal{I}$ *satisfies* α. *Similarly,* $\mathcal{I} \models \mathcal{T}$ *(resp.* $\mathcal{I} \models \mathcal{A}$*) when* $\mathcal{I}$ *satisfies all assertions in* $\mathcal{T}$ *(resp.*$\mathcal{A}$*). Finally,* $\mathcal{I} \models \mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ *if* $\mathcal{I} \models \mathcal{T}$ *and* $\mathcal{I} \models \mathcal{A}$.

We point out two important syntactical restrictions of *DL-Lite*$_{core}^{(\mathcal{HF})}$. Namely, *DL-Lite*$_{core}^{\mathcal{H}}$ and *DL-Lite*$_{core}^{\mathcal{F}}$. The former does not contain functional assertions, while the later does not allow role inclusions.

Unrestricted merging of those two logic gives a DL *DL-Lite*$_{core}^{\mathcal{HF}}$. *DL-Lite*$_{core}^{\mathcal{HF}}$, has no conditions on potential interplay between role inclusions and functional constraints. Since, it is more expressive logic then *DL-Lite*$_{core}^{(\mathcal{HF})}$. In particular, satisfiability of *DL-Lite*$_{core}^{\mathcal{HF}}$ is not FOL-rewritable. More on this we elaborate in the following.

2.3.2 Conjunctive Queries in *DL-Lite*

Conjunctive queries (CQs) are queries constructed as FOL queries ϕ using conjunction ($\wedge$), existential quantifier ($\exists x$), atomic concepts ($A(t_1)$) , atomic roles ($P(t_1, t_2)$) where t_1 and t_2 are terms. Formally:

$$\begin{array}{l} t \longrightarrow x \mid o_i \\ \varphi \longrightarrow \exists x.\varphi \mid \varphi_1 \wedge \varphi_2 \mid A(t) \mid P(t_1, t_2) \end{array}$$

where ϕ is a proper FOL formula, [2] x is a variable and o_i is a object constant.

To denote, CQ query we will use datalog notation:

$$q(\vec{x}) : - conj(\vec{x}, \vec{y})$$

where q names a query, called the head of query, and *conj* denotes the conjunction described above, called the body of a query. As usual, $\vec{x}$ represents a vector of distinguishable variables $\langle x_1, \ldots, x_n \rangle$, and $\vec{y}$ a vector of non-distinguishable variables $\langle y_1, \ldots, y_m \rangle$. Arity of a CQ query is the arity of $\vec{x}$.

Union of Conjunctive queries with datatypes(UCQ), is a FOL query that is constructed as a disjunction of CQs of the same arity:

$$q(\vec{x}) : - \bigvee_{1 \leq i \leq k} conj_i(\vec{x}, \vec{y_i})$$

We denote with $q(\vec{o})$ a formula obtained from $q(x)$ by replacing $\vec{x}$ with $\vec{o}$.

The complexities of the decision problems of interest for UCQs and FOL queries are presented at table 2.1.

[2] in particular $\exists x\varphi$ can be written only if x exists in φ and it is unbounded. Additionally, we assume that $\exists x$ can appear at mostly once in a CQ.

	Combined complexity	Data complexity	Query complexity
UCQ	NP	AC^0	NP
FOL query	PSPACE	AC^0	PSPACE

Table 2.1: The decision procedures complexities for UCQ and FOL queries

Definition 4 (Homomorphism between a model and a CQ). *Given a CQs $q(x) = \exists y.conj(x, y)$ over interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, and a tuple $o = (o_1, \ldots, o_n)$ of objects of $\Delta^{\mathcal{I}}$ of the same arity as $x = (x_1, \ldots, x_n)$, a homomorphism from $q(o)$ to $\mathcal{I}$ is a mapping μ from the variables and constants in $q(x)$ to $\Delta^{\mathcal{I}}$ such that:*

- *$\mu(c) = c^{\mathcal{I}}$, for each constant c in $conj(x, y)$,*
- *$\mu(x_i) = o_i$, for $i \in \{1, \ldots, n\}$, and*
- *$(\mu(t_1), \ldots, \mu(t_n)) \in P^{\mathcal{I}}$, for each atom $P(t_1, \ldots, t_n)$ that appears in $conj(x, y)$.*

Theorem 1. *Given a CQ $q(x) = \exists y.conj(x, y)$ over an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, and a tuple $o = (o_1, \ldots, o_n)$ of objects of $\Delta^{\mathcal{I}}$ of the same arity as $x = (x_1, \ldots, x_n)$, we have that $o \in q^{\mathcal{I}}$ if and only if there is a homomorphism from $q(o)$ to $\mathcal{I}$.*

Definition 5 (Certain answers of UCQs). *Let O be a DL-Lite$_{core}^{(\mathcal{HF})}$ ontology and q a UCQ over $\mathcal{O}$. A tuple $\vec{c}$ of constants appearing in O is a certain answer to q over $\mathcal{O}$, written $\vec{c} \in cert(q, \mathcal{O})$, if for every model $\mathcal{I}$ of $\mathcal{O}$, we have that $c^{\mathcal{I}} \in q^{\mathcal{I}}$.*

2.3.3 Inferencing

Traditionally, in the most of DLs there are some reasoning services of interest. In addition in *DL-Lite* family we are interest also in checking (funct f) and answering (U)CQ [10]. So the problems include:

- *Ontology satisfiability*, i.e., to check whether for a given ontology exists at least one model.
- *Concept or role satisfiability*, i.e., given a TBox $\mathcal{T}$ and a concept C (resp., a role R or an attribute U), verify whether $\mathcal{T}$ admits a model $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$ (resp., $R^{\mathcal{I}} \neq \emptyset$).
- *Logical implication*, i.e. to check whether an ontology $\mathcal{O}$(resp. $\mathcal{T}$) logically implies assertion α, is to check if for every model $\mathcal{I}$ of $\mathcal{O}$(resp. $\mathcal{T}$) is also a model for α. We write shortly $\mathcal{O} \models \alpha$ (resp. $\mathcal{O} \models \alpha$). In particular we intresting in following sub-problems:
  - *instance checking*, i.e., given an ontology $\mathcal{O}$, a concept $\mathcal{C}$ and a constant o (resp., a role R and a pair of object constants o_1 and o_2), verify whether $\mathcal{O} \models C(o)$ (resp. $O \models R(o_1, o_2)$);
  - *subsumption of concepts and roles* i.e., given a TBox $\mathcal{T}$ and two general concepts C_1 and C_2 (resp., two general roles R_1 and R_2), verify whether $\mathcal{T} \models C_1 \sqsubseteq C_2$ (resp.,$\mathcal{T} \models R_1 \sqsubset R_2$);

– *checking functionality*, i.e., given a TBox $\mathcal{T}$ and a basic role R, verify whether $\mathcal{T} \models (\mathsf{funct}\ R)$.

In addition we are interested in:

- *Query answering*, i.e., given an ontology $\mathcal{O}$ and a query q (either a CQ or a UCQ) over $\mathcal{O}$, compute the set $cert(q, \mathcal{O})$.

The following decision problem, called *recognition problem*, is associated to the query answering problem: given an ontology O, a query q (either a CQ or a UCQ), and a tuple of constants $\mathbf{o}$ of $\mathcal{O}$, check whether $\mathbf{o} \in cert(q, \mathcal{O})$. When we talk about the computational complexity of query answering, in fact we implicitly refer to the associated recognition problem.

We point out that the problems of concepts and roles satisfiability , as well as the problems related to logical implication, can be reduced to the problem of ontology satisfiability in *DL-Lite*$_{core}^{(\mathcal{HF})}$. Additionally, the problem of ontology satisfiability can be reduced to the problem of CQ answering in *DL-Lite*$_{core}^{(\mathcal{HF})}$. The proves of the reductions can be found in [10]. Since, in the thesis we will focus only on the problems of ontology satisfiability and query answering over the ontologies.

2.3.4 FOL-Rewritability notion

We define the notion of FOL-Rewritability as it is presented in [10].

Assume we are given a DL ABox $\mathcal{A}$. Based on that we create an interpretation $\mathcal{DB}(\mathcal{A}) = (\Delta^{DB(\mathcal{A})}, \cdot^{DB(\mathcal{A})})$ on the following way:

- $\Delta^{DB(\mathcal{A})}$ is a union of all objects constants that appears in $\mathcal{A}$
- $o^{DB(\mathcal{A})} = o$ for each object constant
- $A^{DB(\mathcal{A})} = \{o | A(o) \in \mathcal{A}\}$, for each atomic concept A,
- $P^{DB(\mathcal{A})} = \{(o_1, o_2) | P(o_1, o_2) \in \mathcal{A}\}$, for each atomic role P

Definition 6. *Satisfiability in a DL $\mathcal{DL}$ is FOL-rewritable, if for every TBox $\mathcal{T}$ expressed in $\mathcal{L}$, there exists a boolean FOL query q, over the alphabet of $\mathcal{T}$, such that for every non-empty ABox $\mathcal{A}$, the ontology $(\mathcal{T}, \mathcal{A})$ is satisfiable if and only if q evaluates to false in $DB(\mathcal{A})$.*

Definition 7. *Answering UCQs in a DL $\mathcal{DL}$ is FOL-rewritable, if for every UCQ q and every TBox $\mathcal{T}$ expressed over $\mathcal{L}$, there exists a FOL query q_1, over the alphabet of $\mathcal{T}$, such that for every non-empty ABox $\mathcal{A}$ and every tuple of constants $\mathbf{t}$ occurring in $\mathcal{A}$, we have that $t \in cert(q, (\mathcal{T}, \mathcal{A}))$ if and only if $t^{DB(\mathcal{A})} \in q_1^{DB(\mathcal{A})}$*

If a DL $\mathcal{DL}$ has a property of FOL-rewritability of satisfiability (resp., query answering) it means that checking satisfiability (resp. query answering) can be done by evaluating a FOL query over the ABox $\mathcal{A}$. Practically this means that if our ABox is stored in a relational database, we don't have to import the data to the main memory to check those tasks. This has a crucial effect on the applicability if we imagine that we are working with a big ABox.

In theory this also reflected on the computational complexity issue. According to the definition a FOL query depends only on the TBox. Evaluating

First-Order Logic queries (i.e. an SQL queries without aggregation) over a DB is in AC^0 complexity class, this means that our problem will be in AC^0 as well.

On the other hand, to showing that a certain problem in a DL $\mathcal{DL}$ is not FOL-rewritable, it is enough to show that data complexity of the problem is above AC^0, for example LogSpace-hard, NLogSpace-hard, PTime-hard, or even coNP-hard.

We state the known result for *DL-Lite*$_{core}^{(\mathcal{HF})}$.

Theorem 2. *Ontology satisfiability in DL-Lite*$_{core}^{(\mathcal{HF})}$ *is FOL-rewritable. Answering* UCQs *in DL-Lite*$_{core}^{(\mathcal{HF})}$ *is FOL-rewritable.*

Chapter 3

Datatypes

In this chapter we are defining a formal language for describing datatypes, together with the language for constructing new datatypes. Also we characterize the sets of datatypes, throughout the notion of datatype lattice, based on the certain restriction.

Before the formal definitions, we motivate a reader, with two overviews on datatypes appearances in the DLs as well as in the semistructured languages like XML.

3.1 Datatypes in DLs and wider

We distinguish two axes of our concern that should guide us in the selecting proper datatypes language. One is the applications of datatypes, and de facto standards that have been defined for datatypes is Computer Science in languages like XML, RDF, OWL and so forth, a practical side of datatypes. And the second axis, a theoretical view, is the story of introducing datatypes in DLs so far. We explain each, in a more detail.

Datatypes in Computer Science

Standardization of XML datatypes is based on general recommendations and normatives for datatype declerations and usage presented by ISO specification [1] , so called *General-Purpose Datatypes (GPD)* paper. In essence GPD specifies the nomenclature (or terminology) and shared semantics for a collection of datatypes commonly occurring in programming languages and software interfaces (like HTML, XML).

XML Schema Definition (XSD) of XML Datatypes [2] (XSD datatypes, or XML datatypes), is a language specification for XML datatypes, that complies with GPD recommendations. GDP provides very broad view on datatypes in CS, and XSD adopts only a part of it. Nevertheless, XSD specification is a very comprehensive set of datatype generators that operate over defined datatypes (allowing creation of unions, vectors, tables,etc). The subtype (generator) is

[1] ISO 11404: Information technology - General-Purpose Datatypes (GPD) (2007)
[2] defined in document: http://www.w3.org/TR/xmlschema-2/

most basic one as operates only over one datatype and it is used for decelerations of other (common) datatypes. For example, XSD defines datatype integer as a subtype of decimal. However, the presented intuition is not consistently followed. For instance, the real numbers are presented with doubles and floats, which are treated as disjoint sets, plus they are disjoint with decimals (and so with integers). Despite this, XML datatypes are assumed as underling datatype schema for the widely accepted Semantic Web languages, RDF and OWL.

According the W3C recommendation a XML datatype is characterized by a triple (VS, LS, FS) where:

- LS denotes a **lexical space**, which is the set of valid literals (constants) for a datatype
- VS denotes a **value space**. Each value in the value space of a datatype is denoted by one or more literals in its LS.
- Finally FS is a set of **facets**. A facet is a single defining aspect of a VS. Generally speaking, each facet characterizes a value space along independent axes or dimensions.

Example 1 For example, an XML integer can be described with:

$$\mathsf{xsd:int} := (\{..., "-1"\text{^^}\mathsf{xsd:int}, "0"\text{^^}\mathsf{xsd:int}, "1"\text{^^}\mathsf{xsd:int}, ...\}, \{... -1, 0, 1, ...\}, \{<, >, ...\})$$

Δ

In order to visualize an idea of a datatype langugege, we present some of the features of XML datatype language. According XML Schema Datatypes Specification each datatype can be assigned to one of the three classes:

Atomic datatypes are those having values which are regarded by this specification as being indivisible. They can be either *primitive*, like $\mathsf{xsd:decimal}$ or *derived*, like $\mathsf{xsd:nonNegativInteger}$.

List datatypes are those having values each of which consists of a finite-length (possibly empty) sequence of values of an atomic datatype. For example, if an order in a list is not relevant, a list datatype can be $\mathcal{P}(\mathbb{N})$, a partial set of natural numbers, where underling atomic datatype is $\mathsf{xsd:nonNegativInteger}$.

Union datatypes are those whose value spaces and lexical spaces are the union of the value spaces and lexical spaces of one or more other datatypes. For example: $\mathsf{number} := \mathsf{xsd:float} \cup \mathsf{xsd:double} \cup \mathsf{xsd:decimal}$.

Another distinction can be made on:

Primitive datatypes are those that are not defined in terms of other datatypes; they exist ab initio.

Derived datatypes are those that are defined in terms of other datatypes. For example, derivation can be defined by restriction using facets:
$\mathsf{xsd:nonNegativInteger} := \mathsf{xsd:integer}[\geq_0]$.

Finally we can distinguish between:

Built-in datatypes are those which are defined by XML specification [3] , and can be either primitive or derived. XML Schema Definition Language hierarchy of built-in datatypes that is W3C recommended (at the moment of the writing), is given on the Figure [3.1].

User-derived datatypes are those derived datatypes that are defined by individual schema designers.

Additionally, the schema impose that each dataype should be characterized by a unique **namespace**, so a full name of datatype is unique. It is a good practice that a namespace and the URI address where a datatype is specified is exactly the same word. For example, the datatypes from the XML schema are referred with the namespace: `http://www.w3.org/2001/XMLSchema-datatypes`.

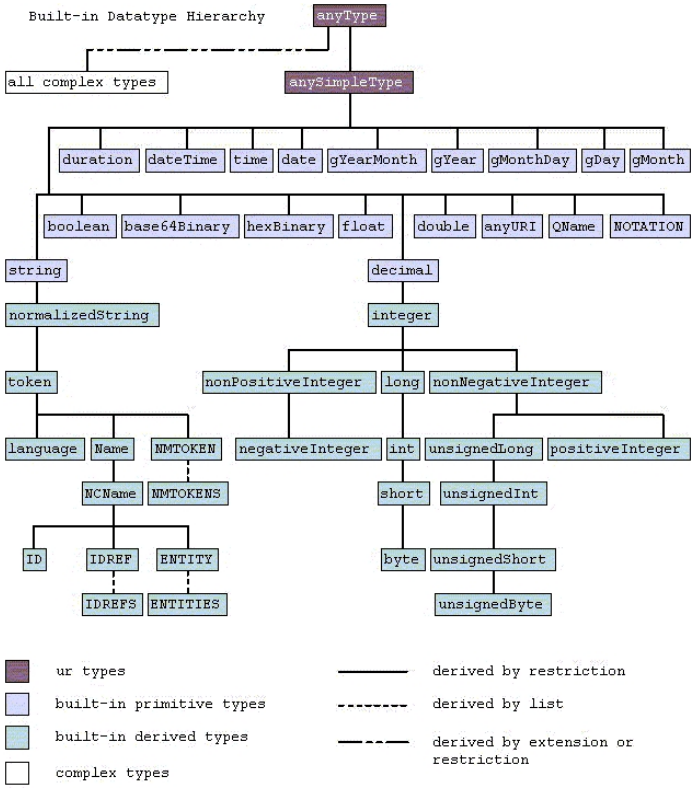


Figure 3.1: XML Schema Datatype Hierarchy of built-in datatypes

Still XSD recommendations doesn't provide any semantics.

W3C recommendation for RDF datatypes [4] adopts XSD with "natural" semantics. In addition, RDF defines untyped literals, datatype constants that are not assigned to any datatype.

On the other hand, W3C recommendation for OWL datatypes [5] has combined approach. It partially adopts some XSD datatypes, but also introduce

[3] http://www.w3.org/TR/xmlschema-2/
[4] http://www.w3.org/TR/rdf-concepts/#section-Datatypes
[5] http://www.w3.org/TR/owl2-direct-semantics/

some new, by overloading already existing XSD datatypes. For example, it recommends to use owl : real instead of xsd : double or xsd : float [6].

Enhancing the OWL1 datatypes support in OWL2, the authors propose a richer language for defining new subtypes over existing one using facets [7]. Also, there is a support to express dependences between values on different attributes using datatype equations.

Finally, OWL2 defines a sub-language (profile) OWL-QL, that guarantees FOL-rewritability of so called `select-project-join` queries (CQs). OWL2-QL is based on a *DL-Lite*-family member *DL-Lite*$_{\mathcal{R}}$. It is a syntactical restriction of full OWL2, and so it allows fewer datatypes than OWL2. The set of allowed datatypes for OWL2-QL is selected *"...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"* [8]

Datatypes in DLs

Datatypes in DLs were firstly introduced by KL-One [8] and so called *Concrete domains*. Concrete domain were more explored in connection with $\mathcal{ALC}$ in [15]. Finally, they were introduced in a DL $\mathcal{EL}$ ($\mathcal{EL}^{++}$) [6], that similarly to *DL-Lite* DLs has weak expressiveness and tractable computational properties. In the mentioned papers the philosophy of adding datatypes in DLs is the same. Datatypes should be accessed throughout the interface provided by a DL. The DL should impose sufficient restrictions to preserve desired computational properties. Reasoning over datatypes is not a part of DL reasoning and should be done by external reasoner.

For example, $\mathcal{EL}^{++}$ condition over the set of admissible datatypes is called p-admissibility, which essentially constraint datatypes, (i) that satisfiability of conjunctions of datatype restriction can be solved in polynomial time (e.g., $(>, 10) \wedge (\leq, 14)$), and (ii) that if any conjunction of datatype constraints implies disjunction of datatype constrains than it implies at least one disjunct (e.g., $(>, 10) \wedge (\leq, 11) \rightarrow (=, 10) \vee (\geq, 15)$ then $(>, 10) \wedge (\leq, 11) \rightarrow (=, 10)$). Datatypes are accessed by predicates (of different arity $n \geq 1$). Although such an approach allows us to have very complex datatype predicates (for example, like "+" where $(x, y, z) \in +$ iff $x + y = z$), the condition of p-admissibility is strong concerning more common unary datatypes, for example integers.

As a consequence, OWL2 profile OWL2-EL, which underlying logic is $\mathcal{EL}^{++}$, allows only equality as an integer constraint (f.e. $=_n (x)$ and no comparisons like $\{<, >, \geq, \leq\}$). In the paper [16] authors relaxed conditions (i) and (ii), allowing inequalities in $\mathcal{EL}^{++}$ like $\{<, \leq, \geq, >\}$ over the arithmetic datatypes, but restricting based on the cases whether inequalities is appeared on the rhs of an axiom (positive occurrence) or on lhs of an axiom (negative occurrence).

In *DL-Lite* family datatypes are introduced with *DL-Lite*$_{\mathcal{A}}$, with strong condition that datatypes assigned to a *DL-Lite*$_{\mathcal{A}}$ logic are pairwise disjoint (over the values). The only type of assertion in which datatype can appear is range restriction for an attribute (e.g. $\mathsf{Rng}(U) \sqsubseteq T_i$). Additionally, datatype constant can appear in a CQ only throughout an attribute (e.g. in a CQ $q(\ldots) \leftarrow \ldots U(x, v) \ldots$).

[6] http://www.w3.org/TR/owl2-syntax/#Datatype_Maps
[7] http://www.w3.org/TR/owl2-primer/#Advanced_Use_of_Datatypes
[8] http://www.w3.org/TR/owl2-profiles/#OWL_2_QL

A new *DL-Lite* family member named *DL-Lite*$_{core}^{\mathcal{HN}\mathcal{A}}$ [5] relaxes the condition of datatype disjointness. In addition to *DL-Lite*$_{\mathcal{A}}$ interface, we can express that two datatypes are disjoint (e.g. $T_1 \sqcap T_2 \sqsubseteq \bot$), that one is a sub-type of the other (e.g. $T_1 \sqsubseteq T_2$), that an attribute is universally quantified (e.g. $B \sqsubseteq \forall U.T_i$) or that a datatype constant belongs to a datatype (e.g. $T_i(v_j)$). Finally, *DL-Lite*$_{core}^{\mathcal{HN}\mathcal{A}}$ assumes that the semantics of datatypes is predefined, i.e. that each interpretation $\mathcal{I}$ obeys the semantics defined by a datatype ($T_i^{\mathcal{I}} = val(T_i) \subseteq \Delta_V^{\mathcal{I}}$).

Then two questions emerge. Firstly, if datatypes are predefined why the modeling language defines assertion of the type $T(v_j)$ (e.g. $\mathsf{xsd:integer}(5)$), or $T_1 \sqsubseteq T_2$ (e.g. $\mathsf{xsd:integer} \sqsubseteq \mathsf{xsd:decimal}$) or $T_1 \sqcap T_2 \sqsubseteq \bot$. Secondly, the reasoning methods presented in the proves considers only dependences between datatypes provided in the TBox of the reasoning ontology. So, it doesn't take into account predefined datatypes dependences that are not specified in the TBox, which also have influence on the reasoning outcome. Even more, the modeling datatype language is not sufficient to capture arbitrary datatype dependences
(f.e. $\mathsf{xsd:integer} \equiv \mathsf{xsd:nonNegativeInteger} \cup \mathsf{xsd:nonPositiveInteger}$).

OWL QL vs. *DL-Lite*$_{\mathcal{R}}$

OWL 2 QL profile is a syntactical restriction of OWL 2 full. Since, it doesn't adopt (UNA), although it is based on a *DL-Lite* member *DL-Lite*$_{\mathcal{R}}$, . Dropping (UNA) assumption in the case of *DL-Lite*$_{\mathcal{R}}$ doesn't increase the complexity of CQ answering (AC^0). However, *DL-Lite* members like *DL-Lite*$_{\mathcal{R}}$ or *DL-Lite*$_{\mathcal{A}}$ where constructor ($\mathsf{funct}\ P$) is allowed, dropping (UNA) will increase the data complexity of CQ answering to NLogSpace. Consequently, CQ answering will not be FOL-rewritable[10].

According W3C recommendation *...The OWL 2 QL profile is designed so that sound and complete query answering is in LogSpace (more precisely, in AC^0) with respect to the size of the data (assertions), while providing many of the main features necessary to express conceptual models such as UML class diagrams and ER diagrams.*

The main differences between OWL-QL and *DL-Lite*$_{\mathcal{A}}$ are [10]:

- *Unique name assumption*(UNA)
- no ($\mathsf{funct}\ P$) nor ($id\ B \ldots$) operators in OWL 2 QL
- OWL 2 QL posses roles restrictions like ($\mathsf{symm}\ P$) and ($\mathsf{asym}\ P$) that we can not express directly using *DL-Lite*$_{\mathcal{A}}$ syntax. On the other hand OWL 2 QL doesn't have ($\mathsf{funct}\ P$) and ($\mathsf{id}B\ \ldots$) restrictions.
- Finally OWL 2 QL is binded to predefined datatypes. In particular rdf : PlainLiteral, rdf : XMLLiteral, rdfs : Literal, owl : real, owl : rational, xsd : decimal, xsd : integer, xsd : nonNegativeInteger, xsd : string, xsd : normalizedString, xsd : token, xsd : Name, xsd : NCName, xsd : NMTOKEN, xsd : hexBinary, xsd : base64Binary, xsd : anyURI, xsd : dateTime, xsd : dateTimeStamp, while in *DL-Lite*$_{\mathcal{A}}$ datatypes are not specified and they assumed to be disjoint. In addition, OWL-QL contains constructor DataIntersectionOf.

3.2 Datatypes language

In the following section we provide a language for describing datatypes and for defining new datatypes using predefined constructors. Mostly inspired with XML and RDF datatype schemas we will define notions of a datatype and datatype lattice. Finally we define a language for constructing new datatypes, and we classify datatype lattices.

In our vocabulary we use $T_1, T_2, \ldots, T_i, \ldots$ to denote datatypes, $LS_{T_1}, LS_{T_2}, \ldots, LS_{T_i}, \ldots$ to denote their lexical spaces, $VS_{T_1}, VS_{T_2}, \ldots, VS_{T_i}, \ldots$ to denote their values spaces, $FS_{T_1}, FS_{T_2}, \ldots, FS_{T_i}, \ldots$ to denote their set of facets, respectfully. Additionally we have infinite set of untyped literals $u_1, u_2, \ldots$.

Definition 8. *A **datatype** T_i is defined by a quadruple $(LS_{T_i}, VS_{T_i}, FS_{T_i}, \cdot^{T_i})$*

- *LS_{T_i} denotes set of constants that is called **lexical space** which is finite or countably infinite.*
- *VS_{T_i} denotes set of values that is called **value space** that is disjoint from LS_{T_i}*
- *FS_{T_i} denotes set of facets constructors that is called **facet space**. A facet constructor is a function that depending of the type on the input accepts either constants from LS_{T_i}, positive integers, or string as parameters and returns a subset of VS_{T_i}.*
- *$\cdot^{T_i}$ is an interpretation function.*

*We assume that a datatype is normalized, i.e. $\cdot^{T_i} : LS_{T_i}$ and VS_{T_i} is a bijection. Additionally for each facet $F_{T_i} \in FS_{T_i}$ and it's acceptable parameters $p_1, \ldots, p_n$ ($n = 0$ if a facet arity is 0), the datatype defines $(F_{T_i}(p_1, \ldots, p_n))^{T_i} \subseteq VS_{T_i}$. An expression $F_{T_i}(p_1, \ldots, p_n)$ we call **parametrized facet** of datatype T_i.*

*A **facet expression** $T_i[\phi]$ over a datatype T_i is a boolean expression ϕ where ϕ is a recursively defined boolean formula using formulas of the form:*

$$\phi \longrightarrow \top_{T_i} | \bot_{T_i} | F_{T_i} | \neg\phi_1 | \phi_1 \wedge \phi_2 | \phi_1 \vee \phi_2$$

where F_{T_i} is a parametrized expression of T_i. Finally the formula $T_i[\phi] = (\phi)^{T_i} \subseteq VS_T$ is interpreted recursively using:

$$(\top_{T_i})^{T_i} = VS_{T_i}, \qquad (\bot_{T_i})^{T_i} = \emptyset, \qquad (\neg\phi_1)^{T_i} = VS_{T_i} \setminus (\phi_1)^{T_i},$$
$$(\phi_1 \wedge \phi_2)^{T_i} = (\phi_1)^{T_i} \cap (\phi_2)^{T_i}, \qquad (\phi_1 \vee \phi_2)^{T_i} = (\phi_1)^{T_i} \cup (\phi_2)^{T_i}$$

To facilitate explanations, we define DL-like expressions to express some of the possible relations between datatypes:

$T_i \sqsubseteq T_j$	if	$VS_{T_i} \subseteq VS_{T_j}$
$T_i \sqsubseteq \neg T_j$	if	$VS_{T_i} \cap VS_{T_j} = \emptyset$
$T_i \equiv T_j$	if	$VS_{T_i} = VS_{T_j}$
$T_k \equiv T_i \sqcap T_j$	if	$VS_{T_k} = VS_{T_i} \cap VS_{T_j}$
$T_k \equiv T_i \sqcup T_j$	if	$VS_{T_k} = VS_{T_i} \cup VS_{T_j}$
$\lvert T_k \rvert = \lvert VS_{T_k} \rvert$		

(Notice that $T_i \sqsubseteq \neg T_j$ iff $T_j \sqsubseteq \neg T_i$)

Definition 9. *A datatype can be either* ***primitive*** *or* ***derived****. Primitive datatypes are those that are defined axiomatically and not in terms of other datatypes. Derived datatypes are all others that are defined using primitive datatypes.*

A datatype T_i is a ***subtype*** *of a datatype T_j if it is defined using expression:*

$$T_i := T_j[\phi]$$

where ϕ is a facet expression over T_j. Additionally, the value space of T_i is defined by $VS_{T_i} = T_j[\phi]$, the lexical space is defined by set of new constants $LS_{T_i} = \{l^v_{T_i} |\ v \in T_j[\phi]\}$ and the facet set $FS_{T_i} = FS_{T_j}$.

A ***datatype range*** *over datatypes $\{T_1, \ldots, T_n\}$ is an expression φ recursively defined using formulas of the form:*

$$\varphi \longrightarrow T_i \mid T_i[\phi] \mid \{d_1, \ldots, d_m\} \mid \varphi_1 \sqcup \varphi_2 \mid \varphi_1 \sqcap \varphi_2 \mid \varphi_1 \setminus \varphi_2 \mid \varphi_1 \times \varphi_2$$

where $i \in \{1, \ldots, n\}$, $\{d_1, \ldots, d_m\}$ is a set of datatype constants from $T_1, \ldots, T_n$, ϕ is a facet expression over T_i, $\times$ is a Cartesian product, and φ_1 and φ_2 are data ranges.

A datatype T' is defined by a datatype range φ over datatypes $\{T_1, \ldots, T_n\}$ if it defined by expression:

$$T' := \varphi$$

Additionally, the value space of T' is defined by $VS_{T'} = \varphi^{T'}$, where $\varphi^{T'}$ is determined recursively:

$$T_i^{T'} = VS_{T_i}\ ,\ (T_i[\phi])^{T'} = (T_i[\phi])^{T_i},$$

$$(\varphi_1 \sqcup \varphi_2)^{T'} = (\varphi_1)^{T'} \cup (\varphi_2)^{T'}\ ,\ (\varphi_1 \sqcap \varphi_2)^{T'} = (\varphi_1)^{T'} \cap (\varphi_2)^{T'},$$

$$(\varphi_1 \setminus \varphi_2)^{T'} = (\varphi_1)^{T'} \setminus (\varphi_2)^{T'}\ ,\ (\varphi_1 \times \varphi_2)^{T'} = ((\varphi_1)^{T'}, (\varphi_2)^{T'})$$

The lexical space is a set of fresh constants:

$$LS_{T'} = \{l^v_{T'} | v \in (\varphi)^{T'}\}$$

and $FS_{T_i} = \emptyset$.

A derived datatype T_i directly depends *on a datatype T_j if it is a subtype of T_j or T_i it is defined by a data range where T_j occurs.*

Definition 10. *A* ***Datatype lattice*** *$\mathcal{D}$ is defined with a finite set of datatypes $T_i = (LS_i, VS_i, FS_i)$ $(1 \leq i \leq n)$ and infinite set of untyped constants $\{u_1, u_2, \ldots\}$. Additionally, $\mathcal{D}$ determines a function $\cdot^{\mathcal{D}}$ that complies with the semantics of the datatypes, i.e. $\cdot^{\mathcal{D}} = \bigcup_i \cdot^{T_i}$ Then $\cdot^{\mathcal{D}}$ naturally extends to sets of literals, facet expressions and range expressions.*

Additionally we put constraint on $\mathcal{D}$:

1. *Each derived datatype in $\mathcal{D}$ is defined by datatypes from $\mathcal{D}$, either by data range or as a sub-type. In addition, dependency relation over datatypes in $\mathcal{D}$ is acyclic.*

2. *for each two datatypes T_i and T_j in $\mathcal{D}$ their lexical spaces are disjoint: $LS_{T_i} \cap LS_{T_j} = \emptyset$ (strict typing).*

3. *$\mathcal{D}$ contains an empty datatype $\bot_{\mathcal{D}} = (\emptyset, \emptyset, \emptyset)$*

4. *each untyped constant u_i does not appear in any of lexical spaces in $\mathcal{D}$ ($u_i \notin \bigcup_{1 \leq i \leq n} LS_{T_i}$). Also we define $(u_i)^{\mathcal{D}} = u_i \notin \bigcup_{1 \leq i \leq n} VS_{T_i}$. Lastly for each two untyped constants u_i and u_j ($i \neq j$), $(u_i)^{\mathcal{D}} \neq (u_j)^{\mathcal{D}}$.*

Each datatype lattice $\mathcal{D}'$ defines a datatype domain $\Delta_{\mathcal{D}}$:

$$\Delta_{\mathcal{D}} = \bigcup_{T_i \text{ from } \mathcal{D}'} T_i^{\mathcal{D}'} \cup \bigcup_{u_i \text{ from } \mathcal{D}'} u_i^{\mathcal{D}'}$$

Comment 1. Not every datatype lattice $\mathcal{D}$ is a mathematical lattice, wrt $\sqsubseteq$ as a partial order. Nevertheless, we can using presented datatype language "complete" $\mathcal{D}$ to become a mathematical lattice, by defining new datatypes. The construction of a datatype lattice can be done in the following way: At the beginning we are given only primitive datatypes that are pairwise disjoint and independent. Then we enrich the lattice with new datatypes using datatype restrictions (facets and intersection), to obtain new "sub-datatypes". Finally, using union and difference operators over datatypes we shape the lattice to obtain desired lattice structure.

Comment 2. Untyped constants are involved mainly because of the two reasons. The first is to avoid closed domain issue and the second is to enhance language functionality in practice where not always we know the type of a datatype constant.

According Lemmas 21. and 22., if domain of values is closed, i.e. for each interpretation attributes can only take values from existing datatypes ($\Delta_V = \bigcup_{1=i}^{n} VS_{T_i}$), then $\mathsf{CQ}_{\mathcal{D}}$ query answering (CQ with possible datatypes in it) is not FOL rewritable.

Additionally, we want to avoid anomalies caused by an ongoing set of datatypes in our datatype lattice. For instance , if we extend our logic with disjunction on the rhs then, an assertion $\exists U \sqsubseteq \exists U.T_1 \sqcup \ldots \sqcup \exists U.T_n$ is a tautology if a datatype lattice (or a current standard like XML) have only datatypes $T_1, \ldots, T_n$. Then if the lattice (or a standard) is extended with a new dataype, the inclusion is not tautology any more [17].

Secondly, in practical scenarios (like OBDA scenario) due to technical reasons, the data are not complete and some datatype constants are not typed, or they might be typed with unknown datatype. For this reason, we introduce untyped constants in the lattice.

On a given datatype lattice $\mathcal{D}$ (with n datatypes) we define several conditions that might hold or not.

(infinite) There exists no $T_1, \ldots, T_k$ ($1 \leq k \leq n$) s.t. there have afinite number of data values in common, i.e. exists natural number $m \geq 2$ s.t. $|\prod_{i=1}^{k} T_i| = m$.

(sup-union) There exists no T_{sup} and $T_1, \ldots, T_k$ ($1 \leq k \leq n$) s.t.:
$T_{sup} \not\sqsubseteq T_i$ (for $1 \leq i \leq k$) and every data value from T_{sup} belongs to some T_i ($1 \leq i \leq k$), i.e. $T_{sup} \sqsubseteq \bigsqcup_{i=1}^{k} T_i$.

(infinite-diff.) There exists no T_{sup} and $T_1, \ldots, T_k$ $(1 \leq k \leq n)$ s.t. $T_{sup} \not\sqsubseteq T_i$ (for $1 \leq i \leq k$) and there are finitely many data values that are in T_{sup} but not in T_i $(1 \leq i \leq k)$, i.e. $|T_{sup} \setminus \bigsqcup_{i=1}^{k} T_i| < \infty$.

(open-domain) There are infinitively many untyped values in $\mathcal{D}$, i.e. for every ontology over $\mathcal{D}$ and an interpretation $\mathcal{I}$ over the ontology, $|\Delta_V^{\mathcal{I}} \setminus \bigcup_{i=1}^{n} VS_{T_i}| = \infty$ (see 4.1.2). Lattices with no or finitely many untyped values we denote with $\mathcal{D} = \{T_1, \ldots, T_n\}$.

Finally, we define different type of datatype lattices based on their properties:

($\mathcal{D}_0$) Every lattice $\mathcal{D}$ is of the type 0 (no restrictions).

($\mathcal{D}_1$) A datatype lattice $\mathcal{D}$ is of the type 1 if it obeys (infinite) restriction defined above.

($\mathcal{D}_2$) A datatype lattice $\mathcal{D}$ is of the type 2 if it obeys (infinite), (infinite-diff.), (open-domain) restrictions defined above.

Notice that (infinite-diff.) implies (sub-union), but not other way round.

In order to simplify notations, we will use $\mathcal{D}$ ($\mathcal{D}'$ or similar) to denote datatype lattices of unspecified type, $\mathcal{D}_0$ ($\mathcal{D}'_0$ or similar) to denote datatype lattices of type 0, $\mathcal{D}_1$ ($\mathcal{D}'_1$ or similar) to denote datatype lattices of type 1 and $\mathcal{D}_2$ ($\mathcal{D}'_2$ or similar) to denote datatype lattices of type 2. With a abuse of notation, we will use $\mathcal{D}_0$,$\mathcal{D}_1$ and $\mathcal{D}_2$ to denote classes of type 0,1 and 2 respectfully.

Comment 1. Restrictions over datatype lattices are based on the question what conditions a datatype lattice needs to satisfy in order to guarantee the FOL-rewritability of a particular *DL-Lite* logic and a query language $\mathcal{Q}$.

Comment 2. (open-domain) Strictly considering (open-domain) is not a necessary condition for FOL-rewritability of $\textit{DL-Lite}_{core}^{(\mathcal{HF})} + \mathcal{D}_1 + \mathsf{UCQ}$. However, it is a necessary condition in the case of $\textit{DL-Lite}_{core}^{(\mathcal{HF})} + \mathcal{D}_2 + \mathsf{UCQ}_{\mathcal{D}}$ (see lemma 22).

Chapter 4

OBDA freamwork

In this chapter we identify the theoretical base for the OBDA scenario, called OBDA framework. Firstly, we present *DL-Lite* common interfaces for connecting ontology elements (in particular attributes) with datatypes (denoting it $\mathcal{DL} + \mathcal{D}$). We define semantics for it. Secondly, we extend starting definitions of certain answers and FOL-rewritability to a three component framework, denoted with $\mathcal{DL} + \mathcal{D} + \mathcal{Q}$. Here, $\mathcal{DL}$ stated for a DL language, $\mathcal{D}$ stated for a datatype language and $\mathcal{Q}$ stated for a query language. Lastly, we introduce new query language, called Union of Conjunctive Queries with datatypes ($\mathsf{UCQ}_{\mathcal{D}}$), that extends UCQs. We provide a formal semantics for it.

4.1 Adding datatypes to *DL-Lite*: $\mathcal{DL} + \mathcal{D}$

As it is discussed in the Section 1.2.2, datatypes and concepts as well as object and data constants are essentially different wrt semantics. Also DL languages are not suitable to describe datatypes, because they can't capture the sophisticated relations between predefined datatypes, like a finite number of elements in a difference between two datatypes. The idea is that, reasoning over datatype is done by external reasoning, which ontology reasoners call as a external function.

For this reason, both components, a DL language with datatype interface and datatype language with own characteristics should be considered as orthogonal issues in the light of the OBDA scenario.

4.1.1 Syntax

To introduce datatypes in *DL-Lite* family we extend the vocabulary $\mathcal{V}$ of *DL-Lite*$_{core}^{(\mathcal{HF})}$, with infinite set of attributes $\Gamma_U^{\mathcal{V}} : U_1, U_2, \ldots$ and the vocabulary of datatypes. An attribute connects an object with its quantitative measure, represented throughout some datatype value. [1]

In particular, basic concepts and roles are defined with:

$$B \longrightarrow \quad \bot \quad | \quad A \quad | \quad \exists R \quad | \quad \exists U$$

$$R \longrightarrow \quad P \quad | \quad P^{-}$$

[1] In OWL2 attributes are called *data property*

A *DL-Lite* terminological assertion that we are interested in are of the form:

$B \sqsubseteq B$,	$B \sqsubseteq \neg B$	*(concept inclusions (positive and negative resp.))*
$B \sqsubseteq \exists R.B$		*(quantified concept inclusions (positive)*
$R \sqsubseteq R$,	$R \sqsubseteq \neg R$	*(role inclusions (positive and negative resp.))*
$U \sqsubseteq U$,	$U \sqsubseteq \neg U$	*(attribute inclusions (positive and negative resp.))*
$B \sqsubseteq \exists U.T$		*(local datatype restrictions (inclusion))*
$\mathsf{Rng}(U) \sqsubseteq T$		*(global datatype restrictions (inclusion))*
(funct U),	(funct R)	*(functional constraints)*
(symm R),	(asym R)	*(symmetric and asymmetric constraints resp.)*
(refl R),	(iref R)	*(reflexive and irreflexive constraints resp.)*

where U states for an attribute and T for a datatype. Notice that, $R_1 \sqsubseteq \neg R_2$ ($U_1 \sqsubseteq \neg U_2$ resp.) and $R_2 \sqsubseteq \neg R_1$ ($U_2 \sqsubseteq \neg U_1$ resp.) are semantically equal (see semantics later), so we use $\mathsf{Disj}(R_1, R_2)$ ($\mathsf{Disj}(U_1, U_2)$ resp.) to denote any of them. We define for a basic role $R^- = P_i$ if $R = P_i^-$ and $R^- = P_i^-$ if $R = P_i$.

All inclusion with the "positive" in the name together with local datatype restrictions, and symmetric and reflexive constraints are called *positive inclusions* (PIs). All the rest are *negative inclusions* (NIs).

A membership assertion in *DL-Lite* are of the form:

$$A(o_1), \qquad P(o_1, o_2), \qquad U(o_1, d_1),$$

where o_1 and o_2 are object constant and d_1 is a data constant (typed or untyped).

As usual, a TBox in some *DL-Lite* logic $\mathcal{DL}$ is a finite set of terminological assertion, an ABox is a finite set of membership assertions, and a *DL-Lite* ontology, $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, is a pair of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$.

Definition 11. *Let $\mathcal{DL}$ be a DL-Lite DL and $\mathcal{D}$ datatype lattice type. The logic $\mathcal{DL} + \mathcal{D}$ is the logic obtained, extending $\mathcal{DL}$ with syntax constructors that contains attributes and datatypes. Datatypes lattices that appear in $\mathcal{DL}+\mathcal{D}$ logic are datatype lattice of $\mathcal{D}$ type.*

For example, *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ represents a logic where *DL-Lite*$_{core}^{(\mathcal{HF})}$ logic is extended with datatype lattice of type $\mathcal{D}_1$.

4.1.2 Semantics

An interpretation, $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, consists of a nonempty domain $\Delta^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$. The interpretation domain $\Delta^{\mathcal{I}}$ is the union of two non-empty disjoint sets, $\Delta^{\mathcal{I}} = \Delta_O^{\mathcal{I}} \cup \Delta_V^{\mathcal{I}}$, the domain of objects $\Delta_O^{\mathcal{I}}$ and the domain of values $\Delta_V^{\mathcal{I}}$.

We assume that all interpretations agree on the semantics assigned to each datatype in some datatype lattice $\mathcal{D}$. In particular, for an ontology $\mathcal{DL} + \mathcal{D}$, defined over datatype lattice $\mathcal{D}'$:

$$\Delta_V^{\mathcal{I}} = \Delta_{\mathcal{D}'} = \bigcup_{T_i \text{ from } \mathcal{D}'} T_i^{\mathcal{D}'} \cup \bigcup_{u_i \text{ from } \mathcal{D}'} u_i^{\mathcal{D}'}$$

The rest, $\mathcal{I}$ interprets as usual, each object constant $o_i^{\mathcal{I}} \in \Delta_O^{\mathcal{I}}$, each data constant $d_i^{\mathcal{I}} = d_i^{\mathcal{D}'} \in \Delta_{\mathcal{D}}^{\mathcal{I}}$ (type or untyped), each atomic concept $A_i^{\mathcal{I}} \subseteq \Delta_O^{\mathcal{I}}$, each atomic role $P_i^{\mathcal{I}} \subseteq \Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}}$, and each atomic attribute $U_i^{\mathcal{I}} \subseteq \Delta_O^{\mathcal{I}} \times \Delta_{\mathcal{D}}^{\mathcal{I}}$. Additionally, object constants adopt *unique name assumption* (UNA): $o_i^{\mathcal{I}} \neq o_j^{\mathcal{I}}$, for $i \neq j$. Notice that (UNA) is not imposed over datatype constants.

Also we extend interpretations over datatypes and predefined concept, $\top_d^{\mathcal{I}} = \Delta_{\mathcal{D}}^{\mathcal{I}}$, $\bot_d^{\mathcal{I}} = \emptyset$, $\top^{\mathcal{I}} = \Delta_O^{\mathcal{I}}$, $\bot^{\mathcal{I}} = \emptyset$, and :

$$
\begin{aligned}
(\neg B)^{\mathcal{I}} &= \Delta_O^{\mathcal{I}} \setminus B^{\mathcal{I}} & (\exists R)^{\mathcal{I}} &= \{o | \exists o'.(o', o) \in R^{\mathcal{I}}\} \\
(\neg R)^{\mathcal{I}} &= (\Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}}) \setminus R^{\mathcal{I}} & (\exists R.C)^{\mathcal{I}} &= \{o | \exists o'.(o', o) \in R^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\} \\
(P^-)^{\mathcal{I}} &= \{(o, o') | (o', o) \in R^{\mathcal{I}}\} & (\exists U)^{\mathcal{I}} &= \{o | \exists v.(o, v) \in U^{\mathcal{I}}\} \\
(\neg U)^{\mathcal{I}} &= (\Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}}) \setminus U^{\mathcal{I}} & (\exists U.T_i)^{\mathcal{I}} &= \{o | \exists v.(o, v) \in U^{\mathcal{I}} \wedge v \in T_i^{\mathcal{I}}\} \\
T_i^{\mathcal{I}} &= T_i^{\mathcal{D}} & (\mathsf{Rng}(U))^{\mathcal{I}} &= \{v | \exists o.(o, v) \in U^{\mathcal{I}}\} \\
\neg T_i^{\mathcal{I}} &= \Delta_V^{\mathcal{I}} \setminus T_i^{\mathcal{D}}
\end{aligned}
$$

Finally, let $\alpha_1 \sqsubseteq \alpha_2$ be a terminological assertion, then $\mathcal{I} \models \alpha_1 \sqsubseteq \alpha_2$ if $(\alpha_1)^{\mathcal{I}} \subseteq (\alpha_2)^{\mathcal{I}}$. Functional, reflexive, irreflexive, symmetric and asymmetric constraints, or shortly role and attribute constraints, are interpreted:

$$
\begin{aligned}
&\mathcal{I} \models (\mathsf{funct}\ R) && \text{if} && \forall o, o_1, o_2 \in \Delta_O^{\mathcal{I}}.(o, o_1), (o, o_2) \in P^{\mathcal{I}} \rightarrow o_1 = o_2, \\
&\mathcal{I} \models (\mathsf{funct}\ U) && \text{if} && \forall o \in \Delta_O^{\mathcal{I}} \forall d_1, d_2 \in \Delta_V^{\mathcal{I}}.(o, d_1), (o, d_2) \in U^{\mathcal{I}} \rightarrow d_1 = d_2, \\
&\mathcal{I} \models (\mathsf{refl}\ R) && \text{if} && \forall o \in \Delta_O^{\mathcal{I}}.(o, o) \in R^{\mathcal{I}}, \\
&\mathcal{I} \models (\mathsf{iref}\ R) && \text{if} && \forall o \in \Delta_O^{\mathcal{I}}.(o, o) \notin R^{\mathcal{I}}, \\
&\mathcal{I} \models (\mathsf{symm}\ R) && \text{if} && \forall o_1, o_2 \in \Delta_O^{\mathcal{I}}.(o_1, o_2) \in R^{\mathcal{I}} \rightarrow (o_2, o_1) \in (R^-)^{\mathcal{I}}, \\
&\mathcal{I} \models (\mathsf{asym}\ R) && \text{if} && \forall o_1, o_2 \in \Delta_O^{\mathcal{I}}.(o_1, o_2) \in R^{\mathcal{I}} \rightarrow (o_2, o_1) \notin (R^-)^{\mathcal{I}}.
\end{aligned}
$$

TBox $\mathcal{T}$ is a finite set of terminological assertions and role and attribute constraints. $\mathcal{I} \models \mathcal{T}$ if $\mathcal{I}$ models each assertion and constraint in $\mathcal{T}$.

Membership assertions are interpreted in a usual way: $\mathcal{I} \models A(o)$ if $o^{\mathcal{I}} \in A^{\mathcal{I}}$, $\mathcal{I} \models P(o_1, o_2)$ if $o_1^{\mathcal{I}}, o_2^{\mathcal{I}}) \in P^{\mathcal{I}}$ and $\mathcal{I} \models U(o_1, d_1)$ if $(o_1^{\mathcal{I}}, d_1^{\mathcal{I}}) \in U^{\mathcal{I}}$. $\mathcal{I} \models \mathcal{A}$ if $\mathcal{I} \models \alpha$ for all membership assertions α from an ABox $\mathcal{A}$.

Finally, $\mathcal{I} \models \langle \mathcal{T}, \mathcal{A} \rangle$ if $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \models \mathcal{A}$.

Comment. In *DL-Lite*$_{\mathcal{A}}$ datatypes are considered pairwise disjoint. Obviously, "disjointness" of the datatypes in *DL-Lite*$_{\mathcal{A}}$ was condition that ensures UNA over the datatypes. Once we have UNA over all constants in *DL-Lite*$_{\mathcal{A}}$ we can syntactically simplify *DL-Lite*$_{\mathcal{A}}$ and investigate logic properties without considering attributes. In $\mathcal{DL} + \mathcal{D}$ we are preserving UNA over the object constants, but not over datatype values. On the other hand, such relaxation have to be chosen carefully in order to preserve FOL-rewritability of query answering.

Another motivation for abandoning UNA over the datatypes comes form the practice. Some of those distinguishable datatype properties are presented 1.2.2. Datatypes included in OWL2 are essentially XML datatypes which complies with [1], and therefore they are not strictly disjoint.

Finally, in $\mathcal{DL} + \mathcal{D}$ we are not binding to any specific set of datatypes, but we are trying to put certain restriction on the datatype set, and then we

are investigating consequences. Defined constructors follows generally accepted standards in declaring datatypes and their behavior specified by [1].

Example 2 We present an example of an academic ontology, using *DL-Lite* language constructors. We illustrate the descriptive capabilities of *DL-Lite* logics. On the Fig. 2 is the UML picture of a modeling domain. Since, the domain is described with DL language it adopts formal semantics, which can be used to reason over it.

Using concept is-a inclusion we model a hierarchy of classes.

$$\begin{aligned} \mathit{RegularStudent} &\sqsubseteq \mathit{Student} \\ \mathit{PhdStudent} &\sqsubseteq \mathit{Student} \\ \mathit{PhdStudent} &\sqsubseteq \mathit{Employee} \\ \mathit{Professor} &\sqsubseteq \mathit{Employee} \\ \mathit{Employee} &\sqsubseteq \mathit{Person} \\ \mathit{Student} &\sqsubseteq \mathit{Person} \end{aligned} \qquad \begin{aligned} \mathit{RegularStudent} &\sqsubseteq \neg \mathit{PhdStudent} \\ \mathit{RegularStudent} &\sqsubseteq \neg \mathit{Professor} \\ \mathit{Professor} &\sqsubseteq \neg \mathit{PhdStudent} \end{aligned}$$

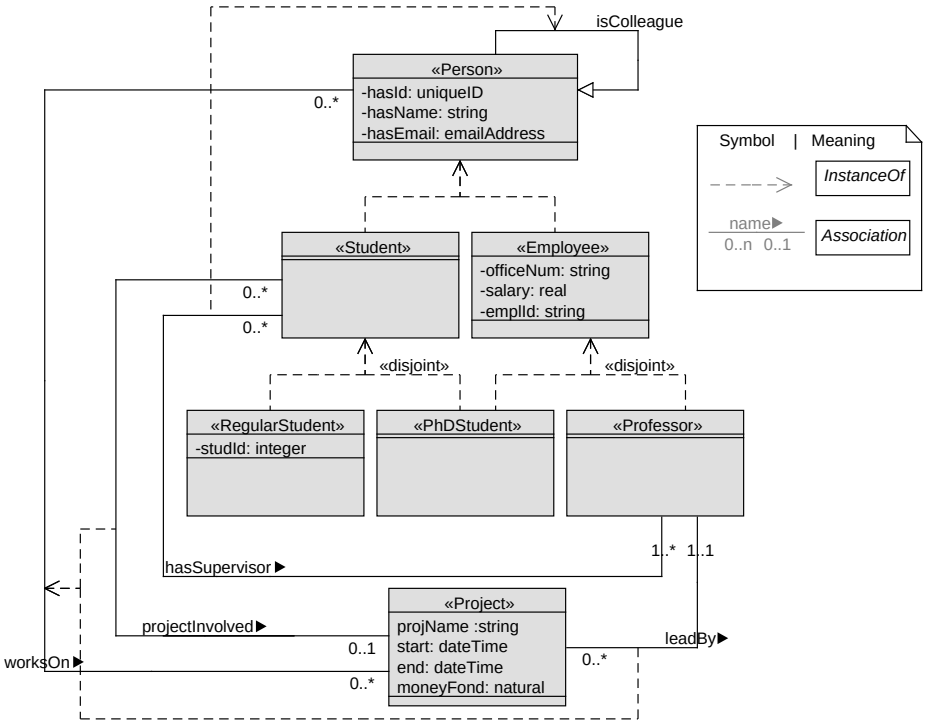


Figure 4.1: UML representation of the academy ontology

Using roles in concept inclusion we define range and domain restriction for

them. Additionally using functionality we specify the multiplicity for the role.

$$\exists hasSupervisor^- \sqsubseteq Professor \quad \text{(restriction on range)}$$
$$\exists hasSupervisor \sqsubseteq Student \quad \text{(restriction on domain)}$$
$$\exists Student \sqsubseteq hasSupervisor \quad \text{(role domain multiplicity 1..*)}$$
$$\exists leadBy^- \sqsubseteq Professor \quad \text{(restriction on range)}$$
$$\exists leadBy \sqsubseteq Project \quad \text{(restriction on domain)}$$
$$\exists Student \sqsubseteq hasSupervisor$$
$$(\mathsf{funct}\ leadBy) \quad \text{(role domain multiplicity 1..1)}$$
$$\exists projectInvolved^- \sqsubseteq Project \quad \text{(restriction on range)}$$
$$\exists projectInvolved \sqsubseteq Student \quad \text{(restriction on domain)}$$
$$\exists isColleague^- \sqsubseteq Person \quad \text{(restriction on range)}$$
$$\exists isColleague \sqsubseteq Person \quad \text{(restriction on domain)}$$
$$(\mathsf{refl}\ isColleague) \quad \text{(collegialism is mutual)}$$

Multiplicity 0..* is obtained without further restrictions.

Role inclusions are used to define more abstract relations.

$$projectInvolved \sqsubseteq worksOn \quad \text{(students are involved)}$$
$$leadBy^- \sqsubseteq worksOn \quad \text{(project leader are involved)}$$
$$hasSupervisor \sqsubseteq isColleague \quad \text{(collegiality in academia :)}$$

Apart from the modeling ontological part of $\mathcal{DL} + \mathcal{D}$, we model datatypes and adopts them based on the needs

$$\mathsf{academy : emailAddress} := \mathsf{xsd : string}[\mathsf{RegExpr}(\gamma)]$$
$$\mathsf{academy : natural} := \mathsf{xsd : integer}[\geq_0]$$
$$\mathsf{academy : uniqueID} := \mathsf{xsd : string} \sqcup \mathsf{xsd : integer}$$

where γ = `\b[A-Z0-9._\%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}\)]` is a regular expression that accepts only regular email addresses.

Finally we define attribute range for each attribute. Notice that, some attributes are contained in others, but modeling data ranges we expend the data range for a super attribute.

$$\mathsf{Rng}(projName) \sqsubseteq \mathsf{xsd : string} \qquad \mathsf{Rng}(studId) \sqsubseteq \mathsf{xsd : integer}$$
$$\mathsf{Rng}(start) \sqsubseteq \mathsf{xsd : dateTime} \qquad \mathsf{Rng}(officeNum) \sqsubseteq \mathsf{xsd : string}$$
$$\mathsf{Rng}(end) \sqsubseteq \mathsf{xsd : dateTime} \qquad \mathsf{Rng}(salary) \sqsubseteq \mathsf{owl : real}$$
$$\mathsf{Rng}(moneyFond) \sqsubseteq \mathsf{academy : natural} \qquad \mathsf{Rng}(emplId) \sqsubseteq \mathsf{xsd : string}$$
$$\mathsf{Rng}(hasEmail) \sqsubseteq \mathsf{academy : emailAddress} \qquad \mathsf{Rng}(hasName) \sqsubseteq \mathsf{xsd : string}$$
$$\mathsf{Rng}(hasId) \sqsubseteq \mathsf{academy : uniqueID}$$
$$(\mathsf{funct}\ projName) \qquad (\mathsf{funct}\ studId)$$
$$(\mathsf{funct}\ start) \qquad (\mathsf{funct}\ officeNum)$$
$$(\mathsf{funct}\ end) \qquad (\mathsf{funct}\ salary)$$
$$(\mathsf{funct}\ moneyFond) \qquad (\mathsf{funct}\ emplId)$$
$$(\mathsf{funct}\ hasId) \qquad (\mathsf{funct}\ hasName)$$
$$emplId \sqsubseteq hasId \qquad studId \sqsubseteq hasId$$

Δ

4.2 OBDA framework: $\mathcal{DL} + \mathcal{D} + \mathcal{Q}$

FOL-rewritability is the major prerequisite that one logic has to satisfy in order to be eligible candidate for the OBDA scenario. Traditionally, OBDA candidates were explored only considering ontology language, while query language was fixed (UCQ)[10, 4].

In order to construct a robust approach to OBDA, we identify three major components of the notion of FOL-rewritability. That are a DL language $\mathcal{DL}$, datatype lattices of type $\mathcal{D}$, and a query language $\mathcal{Q}$. Such framework we denote with $\mathcal{DL} + \mathcal{D} + \mathcal{Q}$.

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a $\mathcal{DL} + \mathcal{D}$ a ontology. An interpretation $DB(\mathcal{A}) = \langle \Delta^{DB(\mathcal{A})}, \cdot^{DB(\mathcal{A})} \rangle$ is obtained from an ABox $\mathcal{A}$, as a minimal $\mathcal{DL} + \mathcal{D}$ interpretation (in inclusion sense) of an ontology $\mathcal{O}' = \langle \emptyset, \mathcal{A} \rangle$. In short, $\Delta_O^{\mathcal{I}} = \{o | o \text{ is object constant that appears in } \mathcal{A}\}$, $A^{DB(\mathcal{A})} = \{o | A(o) \in \mathcal{A}\}$ for each atomic concept A, $P^{DB(\mathcal{A})} = \{(o_1, o_2) | P(o_1, o_2) \in \mathcal{A}\}$ for each atomic role P. Additionally, $\mathcal{I}$ adopts the semantics of datatypes and untyped constants defined by $\mathcal{D}$ ($\Delta_V^{\mathcal{I}} = \Delta_{\mathcal{D}}^{\mathcal{I}}$) and for each atomic attribute U in $\mathcal{A}$, $U^{DB(\mathcal{A})} = \{(o, d^{\mathcal{I}}) | U(o, d) \in \mathcal{A}\}$.

Certain Answers. An ontology encodes the information that is very often incomplete, in the sense that it can admit different models (interpretations), and each model is plausible "description" of the information presented in the ontology.

Given a query q over an ontology $\mathcal{O}$, different ontology models can produce different answerers over the query. One legal view could be to consider a tuple $\vec{o}$ as an answer of evaluating the query over $\mathcal{O}$, if there exists a model $\mathcal{M}$ of the ontology that evaluates $\vec{o} \in q^{\mathcal{M}}$. From logical point of view, this is checking whether the information $q(o)$ contradict the information provided by ontology.

However, we are rather to take a conservative view, that $\vec{o}$ is an answer of evaluating q over $\mathcal{O}$ if $q(\vec{o})$ is entailed by every model of $\mathcal{O}$ (infinite models ones as well). Then we are certain that regardless of the real world model of $\mathcal{O}$ we guarantee that $\vec{o}$ will be a correct answer. The idea comes from the

Information Integration [14].

Definition 12. *Let $\mathcal{DL} + \mathcal{D}$ be a DL, and $\mathcal{O}$ ontology form $\mathcal{DL}$. Let $\mathcal{Q}$ be a query language and q a query from $\mathcal{Q}$ over $\mathcal{O}$. A tuple $\vec{c}$ of constants from $\mathcal{O}$ is a **certain answer** wrt q, written $\vec{c} \in cert(q, O)$, if for every model $\mathcal{I}$ of $\mathcal{O}$ holds $\vec{c}^{\mathcal{I}} \in q^{\mathcal{I}}$.*

The notion of FOL-rewritability of satisfiability is defined in preliminaries and it can be applied to an arbitrary DL. However, we need to extend the notion of FOL-rewritability of query answering, while we consider arbitrary query language. Intuitively, eligible languages for FOL-rewritability of query answering will be fragments of FOL query language.

Definition 13 (FOL-rewritability of $\mathcal{DL}+\mathcal{D}+\mathcal{Q}$)**.** *Let $\mathcal{DL}+\mathcal{D}$ be a DL language extended with datatypes, and $\mathcal{Q}$ a query language. Answering $\mathcal{Q}$ queries in $\mathcal{DL} + \mathcal{D}$ is **FOL-rewritable**, or shortly $\mathcal{DL} + \mathcal{D} + \mathcal{Q}$ is FOL-rewritable, if for every TBox $\mathcal{T}$ from $\mathcal{DL}$ and a query q from $\mathcal{Q}$, there exists a FOL query q_{FOL},*

over the alphabet of $\mathcal{T}$ *, such that for every non-empty ABox* $\mathcal{A}$ *and every tuple of constants* $\mathbf{t}$ *occurring in* $\mathcal{A}$*, we have that* $\mathbf{t} \in cert(q, (\mathcal{T}, \mathcal{A}))$ *if and only if* $\mathbf{t}^{DB(\mathcal{A})} \in q_{FOL}^{DB(\mathcal{A})}$.

4.3 Union of Conjunctive queries with datatypes($\mathsf{UCQ}_\mathcal{D}$)

As we seen in Preliminaries, CQs correspond to SQL `Select-Project-Join` queries. Precisely, `Select` is expressed by an atom appearance in a query, `Project` is obtained by distinguishable variables (others are under existential quantifier), and `Join` is obtained by shared variables in the query.

UCQs can be seen as SQL `Select-Project-Join-Union` queries. Still, `Select-Project-Join-Union` queries are less expressible then full SQL (without aggregation), so the question is can we extend the query language of UCQs and still be a FOL rewritable.

On the other side, datatype restrictions, like numerical restrictions ($\{\neq, <, >, \geq, \leq\}$) or regular expressions restriction over strings, are commonly used restriction in SQL queries.

Then a logical question would be, can we somehow enrich our query language (UCQ) with datatype restrictions but still remain FOL rewritable. The answer is yes, but not without certain restrictions over datatype lattices.

In this section, we are presenting a query language $\mathsf{UCQ}_\mathcal{D}$, that strictly contains UCQ.

4.3.1 Syntax

A datatype atom is an atom of the form $T_i(t)$, where t is a term (a variable or a constant).

Conjunctive queries with datatypes($\mathsf{CQ}_\mathcal{D}s$) are queries constructed as a FOL query ϕ using conjunction ($\wedge$), existential quantifier ($\exists x$), atomic concepts ($A(t_1)$) , atomic roles ($P(t_1, t_2)$), atomic attributes ($U(t_1, t_2)$) and datatype atoms ($T_i(t_1)$), where t_1 and t_2 are terms. Formally:

$$t \longrightarrow x \mid o_i \mid d_i$$
$$\varphi \longrightarrow \exists x.\varphi \mid \varphi_1 \wedge \varphi_2 \mid A(t) \mid P(t_1, t_2) \mid U(t_1, t_2) \mid T_i(t_1)$$

where ϕ is a proper FOL formula, [2] x is a variable, o_i is a object constant, d_i is a datatype constant. Further, a $\mathsf{CQ}_\mathcal{D}$ φ obeys a safety restriction:

(Safety) Every distinguishable variable x that appears in a datatype atom $T_i(x)$ has to appear in at least one atom which is not datatype atom (concept, role or attribute)

To denote, $\mathsf{CQ}_\mathcal{D}$ query we will use datalog notation:

$$q(\vec{x}) :- conj^\mathcal{D}(\vec{x}, \vec{y}),$$

where q names a query, and we called it the head of a query, and $conj^\mathcal{D}$ denotes the conjunction described above, and we called it the body of a query. As usual,

[2] in particular $\exists x\varphi$ can be written only if x exists in φ and it is unbounded. Additionally, we assume that $\exists x$ can appear at most once in a $\mathsf{CQ}_\mathcal{D}$.

$\vec{x}$ represents a vector of distinguishable variables $\langle x_1, \ldots, x_n \rangle$, and $\vec{y}$ a vector of non-distinguishable variables $\langle y_1, \ldots, y_m \rangle$. Arity of $\mathsf{CQ}_\mathcal{D}$ query is the arity of $\vec{x}$. For the sake of readability, we will denote distinguishable variables with $\vec{x}, x_1, \ldots$ and non distinguishable variables with $\vec{y}, y_1, \ldots$.

Comment. A FOL query φ is *safe* if $\varphi^\mathcal{I}$ is finite for all finite instances $\mathcal{I}$. Considering that a query over an ontology can have for the answers only the tuples from the ontology ABox, we expect that all $\mathsf{CQ}_\mathcal{D}s$ are also safe queries. However, without (Safety) condition this is not the case. Consider an ontology $\mathcal{O} = \langle \emptyset, \{A(o)\} \rangle$ and query $q(x_1, x_2) : -A(x_1), T(x_2)$. Then for every interpretation $\mathcal{I}$ of $\mathcal{O}$, the answer to the query, $q^\mathcal{I} = \{(o, v) | v \in T^\mathcal{I}\}$, returns tuples that are not in the ABox of $\mathcal{O}$. In addition, $q^\mathcal{I}$ is infinite when T is infinite.

Union of Conjunctive queries with datatypes($\mathsf{UCQ}_\mathcal{D}$), is a FOL query that is constructed as a disjunction of $\mathsf{CQ}_\mathcal{D}$ queries of the same arity:

$$q(\vec{x}) : - \bigvee_{1 \leq i \leq k} conj_i^\mathcal{D}(\vec{x}, \vec{y}_i)$$

We denote with $q(\vec{o})$ a formula obtained from $q(x)$ by replacing $\vec{x}$ with $\vec{o}$.

4.3.2 Semantics

Let $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ be an interpretation and q an $\mathsf{UCQ}_\mathcal{D}$ of arity n. We denote with $q^\mathcal{I}$ a set of tuples $\vec{o} \in (\Delta^\mathcal{I})^n$ s.t. $\mathcal{I} \models q(\vec{o})$.

Similarly, as in the case for CQ, we define the notion of homomorphism between a $\mathsf{UCQ}_\mathcal{D}$ and an interpretation and we establish Chandra-Merlin theorem [13].

Definition 14 (Homomorphism between a model and a query)**.** *Let $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ be an interpretation, $q(x)$ a $\mathsf{CQ}_\mathcal{D}$ query, and a tuple $o = \langle o_1, \ldots, o_n \rangle$ of objects and data constants from $\Delta^\mathcal{I}$ s.t. x has arity n. Mapping μ is a homomorphism from $q(o)$ to $\mathcal{I}$ if it maps variables and constants from $q(x)$ to $\Delta^\mathcal{I}$ on the following way:*

- *$\mu(c) = c^\mathcal{I}$, for each constant c in $q(x)$,*
- *$\mu(x_i) = o_i$, for $i \in \{1, \ldots, n\}$, and*
- *$(\mu(t_1), \ldots, \mu(t_n)) \in P^\mathcal{I}$, for each atom $P(t_1, \ldots, t_n)$ that appears in $q(x)$.*

Theorem 3. *Given a CQ $q(x) = \exists y.conj(x, y)$ over an interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, and a tuple $o = (o_1, \ldots, o_n)$ of objects of $\Delta^\mathcal{I}$ of the same arity as $x = (x_1, \ldots, x_n)$, we have that $o \in q^\mathcal{I}$ if and only if there is a homomorphism from $q(o)$ to $\mathcal{I}$.*

Chapter 5

OBDA framework: $\boldsymbol{DL\text{-}Lite}_{\boldsymbol{core}}^{(\mathcal{HF})} + \mathcal{D}_1 + \mathsf{UCQ}$

In the first part of this chapter, we are proving that $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1 + \mathsf{UCQ}$ is FOL-rewritable. This is done in two steps. Firstly, by showing that satisfiability is FOL-rewritable in $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1$. And secondly, by constructing the perfect reformulation from the given UCQ.

In the second part of the chapter, we are proving that the conditions over datatype lattices in $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1 + \mathsf{UCQ}$ are necessary. In other words, FOL-rewritability of $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1 + \mathsf{UCQ}$ will be lost if we violated them. This is done by encoding coNP-hard problems into the problem of query answering.

$\boldsymbol{DL\text{-}Lite}_{\boldsymbol{core}}^{(\mathcal{HF})} + \mathcal{D}_1$

With a slight abuse of notation we will denote with $\mathcal{D}_1$ datatype lattices of type $\mathcal{D}_1$.

A $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ TBox, $\mathcal{T}$ defined over a datatype lattice $\mathcal{D}_1$, is a finite set of: concept inclusions (positive and negative resp.), quantified concept inclusions (positive), role inclusions (positive and negative resp.), attribute inclusions (positive and negative resp.), local datatype restrictions (inclusion), global datatype restrictions (inclusion) and functional constraints, where datatype symbols are from $\mathcal{D}_1$.

Additionally, $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ TBox $\mathcal{T}$ has to obey a condition:

(FH) If an atomic role P appears with (funct P) or (funct P^-) in $\mathcal{T}$, then it does not appear on the rhs of a positive role inclusion (e.g. $R \sqsubseteq P$ or $R \sqsubseteq P^-$) or quantified concept inclusions (e.g. $B \sqsubseteq \exists P.B$ or $B \sqsubseteq \exists P^-.B$) in $\mathcal{T}$. The same holds for attributes in $\mathcal{T}$.

A $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ ABox, $\mathcal{A}$ defined over a datatype lattice $\mathcal{D}_1$, is a finite set of *DL-Lite* membership assertions, where datatype constants are from $\mathcal{D}_1$.

And a $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ ontology, $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, is a pair of a $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ TBox $\mathcal{T}$ and a $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ ABox $\mathcal{A}$.

Comment 1. Justification of (FH) condition one can find in [10, p. 320], where unrestricted interplay between (funct R) and R on the rhs of a concept inclusion, creates a logic where instance checking is NLogSpace-hard for data complexity,

i.e. (U)CQ answering is not FOL rewritable. Moreover, Kontchakov et al. [4, p. 47] showed that the data complexity of UCQ answering over $DL\text{-}Lite_{core}^{\mathcal{HF}}$ is PTIME-complete.

Comment 2. Recall, that $\mathcal{D}_1$ class of lattices impose the condition (infinite) $(|\prod_{i=1}^{k} T_i| = m > 0, 1 \leq k \leq n)$. Justification of (infinite) condition in $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ logic is confirmed with Lemma 11. We present it in the second part of this chapter.

5.1 Reasoning

Final goal of this section is to prove FOL-rewritability of $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1 +$ UCQ. This is done in four step. Firstly, an ontology is simplified in the normalization step. Secondly, a representative model is defined (canonical model), from which we establish FOL-rewritability of ontology satisfiability.

Lastly, an algorithm that constructs perfect reformulation is presented.

5.1.1 Normalization

In order to simplify analyses of $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ properties, we will rewrite some terminological rules. Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ ontology, then we proceed:

- Each assertion of the form $B \sqsubseteq \bot$ we will rewrite into $\mathsf{Disj}(B, B,)$. Similarly, $\mathsf{Rng}(U) \sqsubseteq \bot_{\mathcal{D}}$ is rewriten to $\mathsf{Disj}(B, B)$. On the other hand we delete all inclusion of the form $B \sqsubseteq \top$, $\bot \sqsubseteq B$ and $\mathsf{Rng}(U) \sqsubseteq \top_{\mathcal{D}}$, as they have no impact. After this, the ontology wont contain symbols $\top$, $\bot$, $\top_{\mathcal{D}}$, $\bot_{\mathcal{D}}$.
- Each $B \sqsubseteq \exists R.B'$ is replaced with $B \sqsubseteq \exists R_{new}$, $R_{new} \sqsubseteq R$, $\exists R_{new}^{-} \sqsubseteq B'$, where R_{new} is a new role name in the ontology.
- Each $B \sqsubseteq \exists U.T_i$ is replaced with $B \sqsubseteq \exists U_{new}$, $U_{new} \sqsubseteq U$, $\mathsf{Rng}(U_{new}) \sqsubseteq T_i$, where U_{new} is a new attribute name in the ontology.
- In order to impose symmetry or asymmetry of a role we introduce new unique named role, Id. When ontology contains $(\mathsf{refl}\ R)$ we introduce Id in a way:
  - For every object constants o from $\mathcal{A}$ add $\top(o)$ and $Id(o, o)$ to $\mathcal{A}$.
  - Extend $\mathcal{T}$ with $\top \sqsubseteq \exists Id$ [1] and $Id^{-} \sqsubseteq Id$.
  - Replace each $(\mathsf{refl}\ R)$ with $Id \sqsubseteq R$ and each $(\mathsf{iref}\ R)$ with $R \sqsubseteq \neg Id$.
- Each $(\mathsf{symm}\ R)$ is replaced with $R \sqsubseteq R^{-}$ and each $(\mathsf{asym}\ R)$ is replaced with $R \sqsubseteq \neg R^{-}$.
- Lastly, we normalize datatypes in $\mathcal{O}$. For each attribute U in $\mathcal{O}$ we define a new datatype $\mathsf{Rng}_{max}^{T}(U)$ if it doesn't exists already (see 5.1.2). A new inclusion $\mathsf{Rng}(U) \sqsubseteq \mathsf{Rng}_{max}^{T}(U)$ is added to $\mathcal{T}$.

[1] $\top \sqsubseteq Id$ is a new constructor, and it is interpreted in a usual way. However, it is only allowed for Id.

Comment 1. The new unique role *Id* in only introduced for technical reasons and doesn't change the syntax perception of *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$. Although, we could kept ($\mathsf{refl}\ R$) (resp. ($\mathsf{iref}\ R$)) in a ontology, and proceed with the proves, the reformulation is more convenient as it goes along with the proves without considering ($\mathsf{refl}\ R$) (resp. ($\mathsf{iref}\ R$)) separately.

Comment 2. Notice, that all rewriting, except quantified concept inclusion ($B \sqsubseteq \exists R.B$) and local datatype restriction ($B \sqsubseteq \exists U.T_i$), are replaced with equivalent DL expressions. Nevertheless, $B \sqsubseteq \exists R.B$ (resp. $B \sqsubseteq \exists U.T_i$) is rewritten in expressions, s.t. every model on them is also a model of $B \sqsubseteq \exists R.B$ (resp. $B \sqsubseteq \exists U.T_i$). Opposite way doesn't hold, but only because of the rewritings has more predicates. To formalize it,

Definition 15. *Two TBoxes $\mathcal{T}_1$ and $\mathcal{T}_2$ are* ***modulo equivalent*** *if: for every interpretation $\mathcal{I}$ that interprets only symbols from $\mathcal{T}$ s.t. $\mathcal{I} \models \mathcal{T}_1$ exists positive extension of $\mathcal{I}$, $\mathcal{I}'$, s.t. $\mathcal{I}' \models T'$. And vice versa. Positive extension of $\mathcal{I}$ is an interpretation $\mathcal{I}$ that contains more positive facts (grounded atoms).*

Example 3 $B \sqsubseteq \exists R.B'$ is modulo equivalent with $B \sqsubseteq \exists R_{new}$, $R_{new} \sqsubseteq R$, $\exists R_{new}^{-} \sqsubseteq B'$. While there is no modulo equivalent in *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ for inclusion $B \sqsubseteq \forall U.T_i$. $\bigtriangleup$

If it is not explicitly mentioned, we will consider that every *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ ontology is normalized.

5.1.2 Canonical model

In order to check whether a *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ ontology $\mathcal{O}$ is satisfiable, instead of searching for a model, we would like to build a "representative" model for $\mathcal{O}$, in the sense that such model satisfies $\mathcal{O}$ if and only if $\mathcal{O}$ is satisfiable. This model is called *canonical model* and its build by the notion of chase [2], that builds a model starting from an ABox as a base and then expanding it according positive inclusions in the TBox. We adopt the chase from [10]. For a moment we fix a *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ ontology $\mathcal{O}$.

Maximal range

In building chase new constants are introduced. Adding new object constants, is not a concern as their semantics depends on an interpretation. On the other hand, new value constants for a canonical model has to be selected carefully, as their semantics is already predefined by corresponding datatypes.

An idea is to define a datatype $\mathsf{Rng}_{max}^{\mathcal{T}}(U)$, called a maximal range of an attribute U wrt TBox $\mathcal{T}$. Intuitively, if $\mathsf{Rng}_{max}^{\mathcal{T}}(U)$ is defined for U, the chase should always select datatype constants from $\mathsf{Rng}_{max}^{\mathcal{T}}(U)$.

Formally we define,

Definition 16. *Let $\mathcal{T}$ be a DL-Lite TBox with assigned datatype lattice $\mathcal{D}$. For each attribute U from $\mathcal{T}$ we define a set of datatypes from $\mathcal{D}$, $\mathsf{RngSet}^{\mathcal{T}}(U)$, s.t.:*

$T_i \in \mathsf{RngSet}^{\mathcal{T}}(U)$ *for each datatype T_i s.t. $\rho(U) \sqsubseteq T_i \in \mathcal{T}$*

$\mathsf{RngSet}^{\mathcal{T}}(U') \subseteq \mathsf{RngSet}^{\mathcal{T}}(U)$, *for each attribute U' s.t. $U \sqsubseteq U' \in \mathcal{T}$*

If there exists a datatype T_i form $\mathcal{D}$ s.t. $T_i \equiv \bigsqcap_{T_j \in \mathsf{RngSet}^{\mathcal{T}}(U)} T_j$, we set $\mathsf{Rng}^{\mathcal{T}}_{max}(U) := T_i$. Otherwise we define a new datatype:

$$\mathsf{Rng}^{\mathcal{T}}_{max}(U) := \bigsqcap_{T_j \in \mathsf{RngSet}^{\mathcal{T}}(U)} T_j$$

Lemma 1. *For each attribute U and DL-Lite ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, $\mathsf{Rng}^{\mathcal{T}}_{max}(U)$ can be calculated in a linear time wrt the size of TBox $\mathcal{T}$.*

Proof. We denote with $\sqsubseteq^*$ a transitive closure of $\sqsubseteq$ between attributes.

Based on attributes inclusion in $\mathcal{T}$ we create, a graph $G = \langle V, E \rangle$, where attribute names are vertices and attribute inclusion are edges, i.e.

$$V = \{U | U \text{ is an attribute from } \mathcal{O}\}, \qquad E = \{(U_2, U_1) | U_1 \sqsubseteq U_2 \in \mathcal{T}\}$$

Using Tarjan's algorithm [2] we detect strongly connected components (SCCs) in G. The strongly connected components of a directed graph G are its maximal strongly connected subgraphs, where strongly means that any two vertices of a subgraph are mutually reachable. A SCC in G represent set of equivalent attributes, because $\mathcal{T}$ entails $U_i \sqsubseteq U_j$ and $U_j \sqsubseteq U_i$ for each two attributes U_i and U_j from the SCC. For each SCC in G we denote an equivalent class $[U_i]_{\sqsubseteq}$ wrt $\sqsubseteq$ as a relation.

Based on SCC in G we define SCC graph $G' = (V', E')$, where vertices are SCC from G, and edges are edges between elements of SCCs (without self loops), i.e.

$$V' = \{[U_i]_{\sqsubseteq} | [U_i]_{\sqsubseteq} \text{ is a SCC in } G\}$$

$$E' = \{([U_i]_{\sqsubseteq}, [U_j]_{\sqsubseteq}) | \exists U'_i \in [U_i]_{\sqsubseteq}\ \exists U'_j [U_i]_{\sqsubseteq}.(U'_j, U'_i)\mathcal{T}\}$$

We claim that obtained graph G' is acyclic, i.e. a tree. Assume contrary that exists a cycle and let $[U_i]_{\sqsubseteq}$ and $[U_j]_{\sqsubseteq}$ be a different vertices on a cycle path. Then according, G, $U_i \sqsubseteq^* U_j$ and $U_j \sqsubseteq^* U_i$ are in $\mathcal{T}$. But then, by definition of a SCC must be $[U_i]_{\sqsubseteq} = [U_j]_{\sqsubseteq}$. Contradiction.

Finally, starting from the leaves of G'(vertices without outgoing edges), we can construct $\mathsf{Rng}^{\mathcal{T}}_{max}([U_i]_{\sqsubseteq})$ for each SCC, in a bottom-up fashion. Two properties emerge:

- All attributes from a SCC ($[U]_{\sqsubseteq}$) will have the same $\mathsf{Rng}^{\mathcal{T}}_{max}([U]_{\sqsubseteq})$.
- if $U_1 \sqsubseteq^* U_2$ then $\mathsf{Rng}^{\mathcal{T}}_{max}(U_1) \sqsubseteq \mathsf{Rng}^{\mathcal{T}}_{max}(U_2)$.
-

$$\mathsf{Rng}^{\mathcal{T}}_{max}(U) \equiv \bigsqcap_{\rho(U) \sqsubseteq T_i} T_i \sqcap \bigsqcap_{U \sqsubseteq U'} \mathsf{Rng}^{\mathcal{T}}_{max}(U')$$

To conclude, Tajran's algorithm calculates SCCs in a linear time in the size of a graph ($|G| = |V| + |E|$). G is linear in the size of the TBox $\mathcal{T}$, and we traverse graph G only two times, once to detect SCCs and once in calculating maximal ranges. □

[2] http://en.wikipedia.org/wiki/Tarjan's_strongly_connected_components_algorithm

Example 4 We extend the UML model of academy (Example 2). Each student assigned to a project obtains a project email of the form: `name@project-name.[the-rest]`. This information is contained in attribute *hasProjectEmail*:

$$\exists hasProjectEmail \sqsubseteq Student \qquad \mathsf{Rng}(hasProjectEmail) \sqsubseteq \mathsf{xsd:string}[\mathsf{RegExpr}(\gamma_1)]$$

, where γ_1 = `[A-Z0-9\.-]+@project\.[A-Z0-9.-]+\.[A-Z]{2,4}\)]` is a regular expression.

Further, each student at Uni Bolzano has an email of the form: `name@[something].unibz.it`. This information is contained in attribute *hasUniBZEmail*:

$$\exists hasUniBZEmail \sqsubseteq Student \qquad \mathsf{Rng}(hasUniBZEmail) \sqsubseteq \mathsf{xsd:string}[\mathsf{RegExpr}(\gamma_2)]$$

, where γ_2 = `[A-Z0-9.-]+@[A-Z0-9.-]+unibz\.it` is a regular expression.

Finally, there exists an attribute *hasProjectBZEmail* that contains emails of the Uni Bolzano students that are assigned to some project. Then a natural constraints would be:

$$hasProjectBZEmail \sqsubseteq hasProjectEmail \qquad hasProjectBZEmail \sqsubseteq hasUniBZEmail$$

If there are no further constrains on *hasProjectBZEmail*, we can calculate a maximal range for *hasProjectBZEmail*.Firstly we define a new datatype:

$$\mathsf{Rng}^{\mathcal{T}}_{max}(hasProjectBZEmail) := \mathsf{xsd:string}[\mathsf{RegExpr}(\gamma_1)] \sqcap \mathsf{xsd:string}[\mathsf{RegExpr}(\gamma_2)]$$

And then set the range:

$$\mathsf{Rng}(hasProjectBZEmail) \sqsubseteq \mathsf{Rng}^{\mathcal{T}}_{max}(hasProjectBZEmail)$$

Notice that in the example we slightly abuse the notation, and use expression like $\mathsf{xsd:string}[\mathsf{RegExpr}(\gamma_1)]$ and $\mathsf{Rng}^{\mathcal{T}}_{max}(U)$ do denote datatypes. Formally, this is not correct. However, this formality is mainly introduced to facilitate naming in formal proves. We can imagine that in practical application one can skip such renaming.

Δ

Lemma 2. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$^{(\mathcal{HF})}_{core} + \mathcal{D}_1$ ontology and U an attribute that appears in $\mathcal{O}$. Then:*

(Soundness) $\mathcal{O} \models \mathsf{Rng}(U) \sqsubseteq \mathsf{Rng}^{\mathcal{T}}_{max}(U)$

(Completeness) if $\mathcal{T}$ is satisfiable and $\mathcal{T} \not\models U \sqsubseteq \neg U$ then $\mathcal{T} \not\models \mathsf{Rng}(U) \sqsubseteq T_i$, for each T_i that is a proper sub-datatype of $\mathsf{Rng}^{\mathcal{T}}_{max}(U)$ ($T_i \sqsubseteq \mathsf{Rng}^{\mathcal{T}}_{max}(U)$ and $\mathsf{Rng}^{\mathcal{T}}_{max}(U) \not\sqsubseteq T_i$)

Proof. (Soundness) If $\mathcal{O}$ is unsatisfiable, then everything follows. Assume that $\mathcal{O}$ is satisfiable and for some model $\mathcal{I}$ exists a pair $(o, v) \in U^{\mathcal{I}}$ s.t. $v \notin \mathsf{Rng}^{\mathcal{T}}_{max}(U)$. From the Lemma 1 proof we know: $\mathsf{Rng}^{\mathcal{T}}_{max}(U) = \bigsqcap_{\rho(U) \sqsubseteq T_i} T_i \sqcap \bigsqcap_{U \sqsubseteq U'} \mathsf{Rng}^{\mathcal{T}}_{max}(U')$. So either exists some datatype T_i s.t. $v \notin T_i$ or exists some attribute U' s.t. $v \notin \mathsf{Rng}^{\mathcal{T}}_{max}(U')$. If the former holds, then $\mathcal{I} \not\models \rho(U) \sqsubseteq T_i$. Contradiction. If the later, we know there exists a path (see the proof of previous lemma) in $\mathcal{T}$: $U \sqsubseteq U', U' \sqsubseteq U_1, \ldots, U_{m-1} \sqsubseteq U_m$ and a datatype T_i s.t.

$\mathsf{Rng}(U_m) \sqsubseteq T_i \in \mathcal{T}$, where $v \notin T_i^{\mathcal{I}}$. On the other hand, from $U \sqsubseteq^* U_m$ it follows $(o, v) \in U_m^{\mathcal{I}}$ and then $\mathcal{I} \not\models \mathsf{Rng}(U_m) \sqsubseteq T_i$. Again contradiction.

(Completeness) Let v be a value from $VS_{\mathsf{Rng}^{\mathcal{T}}_{max}(U)}$ s.t. for any datatype T_j if $v \in VS_{T_j}$ then $\mathsf{Rng}^{\mathcal{T}}_{max}(U) \sqsubseteq T_j$, i.e. v is the value that does not belongs to any proper subdatype or datatype with common intersection. According datatypes definition v exists. Additionally, we denote with d_v a datatype constant for which $(d_v)^{\mathcal{D}} = v$ and with T_i an arbitrary proper sub-datatype of $\mathsf{Rng}^{\mathcal{T}}_{max}(U)$.

Let $\mathcal{M}$ be a model of $\mathcal{T}$, where $(o, v') \in U^{\mathcal{M}}$. It means that ontology $\mathcal{O} = \langle \mathcal{T}, \{U(o, d_{v'}), U_{new}(o, d_v)\}\rangle$ is satisfiable, where $(d_{v'})^{\mathcal{D}} = v'$ and U_{new} is a new attribute. According Lemma 7.(see later), $\mathcal{O}$ is satisfied by it's canonical model. Now, let denote with $\mathcal{M}'$ a model which is obtained following the same construction steps of $can(\mathcal{O})$ except the starting base is $chase_0(\mathcal{O}) = \{U(o, d_v, U_{new}(o, d_v)\}$. We claim that $\mathcal{M} \models \mathcal{T}$. If we assume contrary, then the only assertion which can be falsify by $\mathcal{M}'$ are one of the form $\mathsf{Disj}(U_1, U_2)$ and $\mathsf{Rng}(U_1) \sqsubseteq T_j$, where $(o, d_v) \in U_1^{\mathcal{M}}$. In the first case $v \notin \mathsf{Rng}(U_2)^{\mathcal{M}}$, because d_v can not be introduced as a new datatype constant in step of constructing $can(\mathcal{O})$ and $v \notin \mathsf{Rng}(U_2)^{chase_0(\mathcal{O})}$. In the second case, if $(o, d_v) \in U_1^{\mathcal{M}}$ then $U \sqsubseteq^* U_1$ and $\mathsf{Rng}^{\mathcal{T}}_{max}(U) \sqsubseteq \mathsf{Rng}^{\mathcal{T}}_{max}(U_1) \sqsubseteq T_j$, so $\mathsf{Rng}(U_1) \sqsubseteq T_j$ can not be falsified in $\mathcal{M}$. To conclude, $\mathcal{M} \models \mathcal{T}$ and $\mathcal{M} \not\models \mathsf{Rng}(U) \sqsubseteq T_i$. □

Comment. Soundness of Lemma 2, advocates that $\mathsf{Rng}^{\mathcal{T}}_{max}(U)$ is soundly defined wrt $\mathcal{O}$, and adding it to $\mathcal{O}$ will not change the semantics (set of models) of $\mathcal{O}$. Completeness of Lemma 2, advocates that $\mathsf{Rng}^{\mathcal{T}}_{max}(U)$ is a maximal range for U, and introducing $\mathsf{Rng}(U) \sqsubseteq T_i$ into $\mathcal{T}$ for T_i a sub-datype of $\mathsf{Rng}^{\mathcal{T}}_{max}(U)$, can spoil semantics (satisfiability) of $\mathcal{T}$.

Chase

$R(o_1, o_2)$ denotes $P(o_1, o_2)$ when $Q = P$ and $P(o_2, o_1)$ when $Q = P^-$

Definition 17. *Let $\mathcal{A}$ be a set of DL-Lite$^{(\mathcal{HF})}_{core}$+$\mathcal{D}_1$ membership assertions. We denote with $B(o)$ any of the assertions of the form $A(o)$, $R(o, o_2)$, $U(o, d_2)$. We say that a PI α is applicable in $\mathcal{A}$ to a membership assertion $\beta \in \mathcal{A}$ if:*

- $\alpha = B_1 \sqsubseteq B_2, \beta = B_1(o)$ *and* $B_2(o) \notin \mathcal{A}$,
- $\alpha = R_1 \sqsubseteq R_2, \beta = R_1(o_1, o_2)$ *and* $R_2(o_1, o_2) \notin \mathcal{A}$,
- $\alpha = U_1 \sqsubseteq U_2, \beta = U_1(o_1, d_1)$ *and* $U_2(o_1, d_1) \notin \mathcal{A}$,

We are constructing chase extension, a potentially infinite set of assertions $chase(\mathcal{O})$ that extends $\mathcal{A}$ as it is presented in [10]. In order to produce unique chase up to the renaming of fresh constants, it is important to determine the order of applied rules. Firstly we assume that all symbols in our vocabulary $\mathcal{V}$ over $\mathcal{D}$ are (lexicography) fully ordered, so based on this we can order assertions.

In addition we add ordered infinite set of fresh object constant $\Gamma^{\mathcal{V}}_N$, so our $\Gamma^{\mathcal{V}}_O = \Gamma^{\mathcal{V}}_N \cup \Gamma^{\mathcal{V}}_A$ where $\Gamma^{\mathcal{V}}_A$ is a set of object constants that appear in given ontology $\mathcal{O}$.

For each datatype T_i we define the infinite set of datatype constants $Rest_{T_i} = \{d_1^{T_i}, d_2^{T_i}, \ldots\}$, s.t. for each two datatype T_i and T_j, if $i \neq j$ then $Rest_{T_i} \neq Rest_{T_j}$. This is possible, due to the fact that each datatype has infinite number

of datatype constants. Additionally, there is no a datatype constant in $\mathcal{A}$, say d_1 and datatype constant d_2 from some $Rest_{T_i}$, such that $(d_1)^{\mathcal{D}} = (d_2)^{\mathcal{D}}$.

For the attributes which range is not bounded we introduce an infinite set of untyped constants $Rest_{\top_{\mathcal{D}}} = \{u_1, u_2, \ldots\}$ that do not appear in $\mathcal{A}$, and set $\mathsf{Rng}^{\mathcal{T}}_{max}(U) := \top_{\mathcal{D}}$ for each such attribute.

If n is the number of assertions in $\mathcal{A}$ we reserve first n places for them in our order. And let $\mathcal{T}_p$ be the set of PI in $\mathcal{T}$.

Definition 18 (Chase)**.** *We define S_j set of assertions in an iterative way:*

- $S_0 = \mathcal{A}$
- $S_{j+1} = S_j \cup \beta_{new}$ *where β_{new} is a membership assertion numbered with $n + j + 1$ in S_{j+1} and obtained as follows:*

  let β be the first membership assertion in S_j such that there exists a PI $\alpha \in \mathcal{T}_p$ applicable in S_j to β.

  let α be the lexicographically first PI applicable in S_j to β.

  let a_{new} be the constant of Γ_N that follows lexicographically all object constants in S_j and let $d^{T_i}_{new}$ be lexicographically the first constant from $Rest_{T_i}$ that proceeds all such constants in S_j.

  case α, β of

$$
\begin{array}{lllll}
(CR1) & \alpha = A_1 \sqsubseteq A_2 & \wedge & \beta = A_1(o) & \Longrightarrow \beta_{new} = A_2(o) \\
(CR2) & \alpha = A \sqsubseteq \exists R & \wedge & \beta = A(o) & \Longrightarrow \beta_{new} = R(o, o_{new}) \\
(CR3) & \alpha = A \sqsubseteq \exists U & \wedge & \beta = A(o) & \Longrightarrow \beta_{new} = R(o, d^{\mathsf{Rng}^{\mathcal{T}}_{max}(U)}_{new}) \\
(CR4) & \alpha = \exists R \sqsubseteq A_2 & \wedge & \beta = R(o, o') & \Longrightarrow \beta_{new} = A_2(o) \\
(CR5) & \alpha = \exists R_1 \sqsubseteq \exists R_2 & \wedge & \beta = R(o, o') & \Longrightarrow \beta_{new} = R(o, o_{new}) \\
(CR6) & \alpha = \exists R \sqsubseteq \exists U & \wedge & \beta = R(o, o') & \Longrightarrow \beta_{new} = U(o, d^{\mathsf{Rng}^{\mathcal{T}}_{max}(U)}_{new}) \\
(CR7) & \alpha = \exists U \sqsubseteq A_2 & \wedge & \beta = U(o, d) & \Longrightarrow \beta_{new} = A_2(o) \\
(CR8) & \alpha = \exists U \sqsubseteq \exists Q & \wedge & \beta = R(o, d) & \Longrightarrow \beta_{new} = R(o, o_{new}) \\
(CR9) & \alpha = \exists U_1 \sqsubseteq \exists U_2 & \wedge & \beta = R(o, d) & \Longrightarrow \beta_{new} = U_2(o, d^{\mathsf{Rng}^{\mathcal{T}}_{max}(U)}_{new}) \\
(CR10) & \alpha = R_1 \sqsubseteq R_2 & \wedge & \beta = R_1(o, o') & \Longrightarrow \beta_{new} = R_2(o, o') \\
(CR11) & \alpha = U_1 \sqsubseteq U_2 & \wedge & \beta = U(o, d) & \Longrightarrow \beta_{new} = U_2(o, d)
\end{array}
$$

  **Id. For each β_{new} that contains object constant a_{new} that was not present in S_j, in addition to β_{new} we add $Id(a_{new}, a_{new})$ and $\top(a_{new})$ to S_{j+1}*

- *Finally we denote:*

$$chase(\mathcal{O}) := \bigcup_{j \in \mathbb{N}} S_j$$

Intuitively, $can(\mathcal{O})$ model should satisfy all PIs in $\mathcal{T}$. However, one might ask whether all rules are applied uniformly, in the sense that some rules might never get their turn to be applied in $can(\mathcal{O})$. The following criteria confirms uniformity:

Proposition 1. *Let $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ be a DL-Lite$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ ontology, and let α be a PI in $\mathcal{T}$. Then, if there is an $i \in \mathbb{N}$ such that α is applicable in $chase_i(\mathcal{O})$ to a membership assertion $\beta \in chase_i(\mathcal{O})$, then there is a $j \geq i$ such that $chase_{j+1}(\mathcal{O}) = chase_j(O) \cup \beta'$, where β' is the result of applying α to β in $chase_j(\mathcal{O})$.*

Proof. Assume by contradiction that there is no $j \geq i$ s.t. $chase_{j+1}(\mathcal{O}) = chase_j(\mathcal{O}) \cup \beta$. This would mean that either there are infinitely many membership assertions that precede β in the ordering that we choose for membership assertions in $chase(\mathcal{O})$, or that there are infinitely many chase rules applied to some membership assertion that precedes β. However, none of these cases is possible. Indeed, β is assigned with an ordering number m such that exactly $m-1$ membership assertions precede β. Furthermore, a PI can be applied at most once to a membership assertion (afterwards, the precondition is not satisfied and the PI is not applicable anymore), and also there exists only a finite number of PIs. Therefore, it is possible to apply a chase rule to some membership assertion at most times. We can thus conclude that the claim holds. □

Definition 19 (Canonical model)**.** *The canonical interpretation $can(\mathcal{O}) = (\Delta^{can(\mathcal{O})}, \cdot^{can(\mathcal{O})})$ is the interpretation where:*

$\Delta^{can(O)} = \Gamma_O^{\mathcal{V}} \cup \Delta_{\mathcal{D}}$, *where $\mathcal{D}$ is underlying datatype lattice of $\mathcal{O}$.*

$o^{can(\mathcal{O})} = o$, *for each object constant $o \in \Gamma_O^{\mathcal{V}}$, and $d^{can(\mathcal{O})} = (d)^{\mathcal{D}}$ if $d \in LS_{T_i}$ for some datatype T_i.*

$A^{can(\mathcal{O})} = \{o | A(o) \in chase(\mathcal{O})\}$, *for each atomic concept A.*

$P^{can(\mathcal{O})} = \{(o_1, o_2) | P(o_1, o_2) \in chase(\mathcal{O})\}$, *for each atomic role P.*

$U^{can(\mathcal{O})} = \{(o, d^{can(\mathcal{O})}) | U(o, d) \in chase(\mathcal{O})\}$, *for each attribute role U.*

We also define $can_i(\mathcal{O}) = (\Delta^{can_i(\mathcal{O})}, \cdot^{can_i(\mathcal{O})})$, where $\cdot^{can_i(\mathcal{O})}$ is analogous to $\cdot^{can_i(\mathcal{O})}$, except that it refers to $chase_i(\mathcal{O})$ instead of $chase(\mathcal{O})$.

Lemma 3. *(i)Let $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ be a DL-Lite$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ ontology and let $\mathcal{T}_p$ be the set of positive inclusion assertions in $\mathcal{T}$. Then, $can(\mathcal{O}) \models (\mathcal{T}_p, \mathcal{A})$.*

(ii)If two conditions hold, for every $U(o, d) \in \mathcal{A}$ we have $d^{\mathcal{D}} \in \mathsf{Rng}_{max}^{\mathcal{T}}(U)$ and when $\mathcal{O} \models \exists U$ then $\mathsf{Rng}_{max}^{\mathcal{T}}(U) \neq \bot_{\mathcal{D}}$, then $can(\mathcal{O}) \models (\mathcal{T}_p \cup \mathcal{T}_d, \mathcal{A})$, where $\mathcal{T}_d$ is the set of all range constraints ($\mathsf{Rng}(U) \sqsubseteq T_i$) in $\mathcal{T}$.

Proof. (i) $can(\mathcal{O}) \models \mathcal{A}$ by definition as $\mathcal{A} \subseteq chase(\mathcal{O})$. So it remains to show $can(\mathcal{O}) \models \mathcal{T}_p$. Assume contrary that exists assertion $\alpha \in \mathcal{T}_p$ s.t. $\mathcal{O} \not\models \alpha$. We can distinguish several cases for α

- $\alpha = A_1 \sqsubseteq A_2$. This means there exists an object constant $o \in \Gamma_O^{\mathcal{V}}$ s.t. $A_1(o) \in chase(\mathcal{O})$ but $A_2(o) \notin chase(\mathcal{O})$. Obviously in this case $\alpha = A_1 \sqsubseteq A_2$ is applicable to $A_1(o)$ using rule (CR1). According Proposition1 there exists j s.t. $A_2(o) \in S_j$ and consequently $A_2(o) \notin S_j$. Contradiction.
- Other cases for α can be reasoned analogously.

Finally, as existence of such α is not possible so we conclude that $can(\mathcal{O})$ satisfies all assertions in $\mathcal{T}_p$, and therefore $can(\mathcal{O}) \models \langle \mathcal{T}_p, \mathcal{A} \rangle$

(ii) By induction over the construction of $chase(\mathcal{O})$. Notice that if any of two conditions are violated $\mathcal{O}$ is not satisfiable.

Base. From the construction of $\mathsf{Rng}^{\mathcal{T}}_{max}(U)$, we know that $\mathsf{Rng}^{\mathcal{T}}_{max}(U) \sqsubseteq T_i$ for every $\mathsf{Rng}(U) \sqsubseteq T_i \in \mathcal{T}(*)$. Then assuming that for every $U(o,d) \in \mathcal{A}$ we have $d^{\mathcal{D}} \in \mathsf{Rng}^{\mathcal{T}}_{max}(U)$, we conclude that $chase_0(\mathcal{O}) \models \langle \mathcal{T}_d, \mathcal{A} \rangle$.

Induction step. Assume that $chase_i(\mathcal{O}) \models \mathcal{T}_d$. The cases we need to consider are when new assertion of the form $U(o, d_{new})$ is added to $chase_{i+1}(\mathcal{O})$. This possible by two rules, $B \sqsubseteq \exists U$ or $U_1 \sqsubseteq U$. If the first is the case, then d_{new} is from $\mathsf{Rng}^{\mathcal{T}}_{max}(U)$ and taking (*) into account, we conclude that $chase_{i+1}(\mathcal{O}) \models \mathcal{T}_d$. In the second is the case, then according induction base d_{new} must be in $\mathsf{Rng}^{\mathcal{T}}_{max}(U_1)$. On the other hand, $\mathsf{Rng}^{\mathcal{T}}_{max}(U_1) \sqsubseteq \mathsf{Rng}^{\mathcal{T}}_{max}(U)$, so we conclude again $chase_{i+1}(\mathcal{O}) \models \mathcal{T}_d$. In both cases, the condition that $\mathsf{Rng}^{\mathcal{T}}_{max}(U) \neq \bot_d$ guarantees that introducing $U(o, d_{new})$ in $chase_{i+1}(\mathcal{O})$ will not violate $\mathsf{Rng}(U) \sqsubseteq \mathsf{Rng}^{\mathcal{T}}_{max}(U)$. □

Lemma 4. *For every DL-Lite$^{(\mathcal{HF})}_{core}$+$\mathcal{D}_1$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ it holds: $can(\mathcal{O}) \models T_f$ iff $DB(\mathcal{A}) \models T_f$.*

Proof. ($\Rightarrow$) As $can(\mathcal{O}) \models T_f$ and $DB(\mathcal{A}) \subseteq can(\mathcal{O})$ then $DB(\mathcal{A}) \models T_f$.

($\Leftarrow$) By induction over the construction of $chase(\mathcal{O})$.

Induction base follows from assumption $chase_0(\mathcal{O}) = DB(\mathcal{A}) \models \mathcal{T}_f$.

Inductive step. Assume that in the step $(i+1)$ a functional role ($\mathsf{funct}\ R$) or ($\mathsf{funct}\ R^-$) has been violated in $chase_{i+1}(\mathcal{O})$, by a new assertion $R(o_1, o_2)$. Considering that $R_1 \sqsubseteq R$ and $R_1 \sqsubseteq R^-$ are not in $\mathcal{T}$ due to the (FH) restriction, the only possible generating rules for $R(o_1, o_2)$ are $B \sqsubseteq \exists R$ or $B \sqsubseteq \exists R^-$.

If $B \sqsubseteq \exists R$ is the case, then o_2 is a new constant. However, in order to apply the rule, we know that there exists no $R(o_1, o')$ in $chase_i(\mathcal{O})$, for some object constant o'. Then ($\mathsf{funct}\ R$) is not violated. If $B \sqsubseteq \exists R^-$ is the case, then o_1 is a new constant. o_1 doesn't exists in $chase_i(\mathcal{O})$, then adding $R(o_1, o_2)$ can not ($\mathsf{funct}\ R$).

Similarly we reason for ($\mathsf{funct}\ R^-$) and for an attribute functionality ($\mathsf{funct}\ U$). □

In other words lemma shows that functionality constraints will not be violated by extending $DB(\mathcal{A})$ to $can(\mathcal{O})$.

Negative assertions closure

Similarly, as we reason on the positive inclusions we can try to reason over the negative inclusion in a TBox. The only difference is that two negative assertion, e.g. $A_1 \sqsubseteq \neg A_2$ and $A_2 \sqsubseteq \neg A_3$, can't deduce new negative inclusion, i.e. we can't say anything about relation between A_1 and A_3. But from a positive and a negative inclusion can deduce a new negative inclusion, for example if $A_1 \sqsubseteq \neg A_3$ and $A_2 \sqsubseteq A_3$ then $A_1 \sqsubseteq \neg A_2$.

The idea, adopted from [10], is to deduce all negative assertions that follows from a TBox $\mathcal{T}$. Then, in order to check satisfiability it will be sufficient to check those negative inclusion over $DB(\mathcal{A})$.

Let $\mathcal{T}$ be a *DL-Lite$^{(\mathcal{HF})}_{core}$+$\mathcal{D}_1$* TBox. A $cln(\mathcal{T})$ is the smallest set that contains:

- $cln(\mathcal{T})$ contains all functional constraints ($\mathcal{T}_f$), negative inclusions ($\mathcal{T}_n$) and $\mathsf{Rng}(U) \sqsubseteq \mathsf{Rng}^{\mathcal{T}}_{max}(U)$ for each attribute U that appears in $\mathcal{T}$.
- if $B_1 \sqsubseteq B_2 \in \mathcal{T}$ and $\mathsf{Disj}(B_2, B_3) \in cln(\mathcal{T})$ then also $\mathsf{Disj}(B_1, B_3) \in cln(\mathcal{T})$.
- if $R_1 \sqsubseteq R_2 \in \mathcal{T}$ and $\mathsf{Disj}(\exists R_2, B) \in cln(\mathcal{T})$ (or $\mathsf{Disj}(\exists R_2^-, B) \in cln(\mathcal{T})$ resp.) then also $\mathsf{Disj}(R_1, B) \in cln(\mathcal{T})$ (or $\mathsf{Disj}(\exists R_1^-, B) \in cln(\mathcal{T})$ resp.) .
- if $R_1 \sqsubseteq R_2 \in \mathcal{T}$ and $\mathsf{Disj}(R_2, R_3) \in cln(\mathcal{T})$ then also $\mathsf{Disj}(R_1, R_3) \in cln(\mathcal{T})$.
- if any of $\mathsf{Disj}(\exists R, \exists R)$, $\mathsf{Disj}(\exists R^-, \exists R^-)$,$\mathsf{Disj}(R, R)$ is in $\mathcal{T}$, then all three are.
- if $U_1 \sqsubseteq U_2 \in \mathcal{T}$ and $\mathsf{Disj}(\exists U_2, B) \in cln(\mathcal{T})$ then also $\mathsf{Disj}(U_1, B) \in cln(\mathcal{T})$
- if $U_1 \sqsubseteq U_2 \in \mathcal{T}$ and $\mathsf{Disj}(U_2, U_3) \in cln(\mathcal{T})$ then also $\mathsf{Disj}(U_1, U_3) \in cln(\mathcal{T})$.
- if any of $\mathsf{Disj}(\exists U, \exists U)$, $\mathsf{Disj}(U, U)$, $\mathsf{Rng}(U) \sqsubseteq \bot_{\mathcal{D}}$ is in $cln(\mathcal{T})$, then all three are in $cln(\mathcal{T})$.

Obviously, all negative inclusions in $cln(\mathcal{T})$ are logically entailed by $\mathcal{T}$. Next lemma confirms that:

Lemma 5. *For a DL-Lite$^{(\mathcal{HF})}_{core} + \mathcal{D}_1$ TBox $\mathcal{T}$ and an arbitrary NI α it holds: if $cln(\mathcal{T}) \models \alpha$ then $\mathcal{T} \models \alpha$.*

The next lemma, essentially confirms that satisfiability of a *DL-Lite*$^{(\mathcal{HF})}_{core}+\mathcal{D}_1$ ontology if FOL rewritable. The only remaining step is to translate $cln(\mathcal{T})$ into a FOL query, which is rather straightforward.

Lemma 6. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$^{(\mathcal{HF})}_{core}+\mathcal{D}_1$ ontology. Then, $can(\mathcal{O}) \models \mathcal{O}$ iff $DB(\mathcal{A}) \models cln(\mathcal{T})$.*

Proof. ($\Rightarrow$) Each assertion α in $cln(\mathcal{T})$ is logically implied by $\mathcal{T}$, so $can(\mathcal{O}) \models cln(\mathcal{T})$. Now, by restricting the model $can(\mathcal{O})$ we can not falsify any NIs in $cln(\mathcal{T})$. As $DB(\mathcal{A}) \subseteq can(\mathcal{O})$ then $DB(\mathcal{A}) \models cln(\mathcal{T})$.

($\Leftarrow$) By assumption $DB(\mathcal{A}) = chase_0(\mathcal{O}) \models \langle cln(\mathcal{T}), \mathcal{A} \rangle$.

According Lemma 4. and Lemma 3. (i) that $can(\mathcal{O}) \models \langle T_p \cup T_f, \mathcal{A} \rangle$.

So it remain to show that $can(\mathcal{O}) \models \langle T \setminus (T_p \cup T_f), \mathcal{A} \rangle$. We will show even more, $can(\mathcal{O}) \models \langle cln(\mathcal{T}), \mathcal{A} \rangle$. The proof is based on the induction over the construction of $can(\mathcal{O})$.

Base. By assumption $DB(\mathcal{A}) = chase_0(\mathcal{O}) \models \langle cln(\mathcal{T}), \mathcal{A} \rangle$.

Induction step. Assume contrary that for some $i \geq 0$ $chase_i(\mathcal{O}) \models \langle cln(\mathcal{T}), \mathcal{A} \rangle$, but $chase_{i+1}(\mathcal{O}) \not\models \langle cln(\mathcal{T}), \mathcal{A} \rangle$. We distinguish cases, based on the rule that was applied against a membership assertion and a NI that becomes unsatisfiable with a new assertion.

With $B(o)$ we denote any atom of the from $A(o)$, $R(o, o_2)$, $U(o, d)$.

Let $B_1 \sqsubseteq B_2$ be a generating rule for assertion $\beta = B_2(o)$ that is added to $chase_{i+1}(\mathcal{O})$, and $B_1(o) \in chase_i(\mathcal{O})$. We distinguish several cases for the violated NI:

1. $\mathsf{Disj}(B_2, B_3)$,
2. $\mathsf{Disj}(R, R_1)$ for $B_2 = \exists R$,

3. $\mathsf{Disj}(U, U_1)$ for $B_2 = \exists U$, or

4. $\mathsf{Rng}(U) \sqsubseteq T_i$ for $B_2 = \exists U$.

If the first is the case, then we have that $\mathsf{Disj}(B_1, B_3) \in cln(\mathcal{T})$, so $chase_i(\mathcal{O}) \not\models \mathsf{Disj}(B_1, B_3)$. Contradiction.

If the second is the case, $R(o, o_{new})$ is added to $chase_{i+1}(\mathcal{O})$, where o_{new} doesn't appear in $chase_i(\mathcal{O})$, so $\mathsf{Disj}(R, R_1)$ is not violated by $R(o, o_{new})$ for $R_1 \neq Id$. In the case $R_1 = Id$, $(o_{new}, o_{new}) \in Id^{can(\mathcal{O})_{i+1}}$. However, o_{new} is newly introduced so $o \neq o_{new}$. Hence, $\mathsf{Disj}(R, Id)$ cannot be violated. Contradiction.

In the third case we reason as in the previous one.

In the fourth case, new data value is selected from $\mathsf{Rng}^{\mathcal{T}}_{max}(U)$, for which holds $\mathsf{Rng}^{\mathcal{T}}_{max}(U) \sqsubseteq T_i$. The problem can occur only if $\mathsf{Rng}^{\mathcal{T}}_{max}(U) = \bot_{\mathcal{D}}$. But then $\mathsf{Disj}(U, U) \in cln(\mathcal{T})$, and so $\mathsf{Disj}(B_1, \exists U) \in cln(\mathcal{T})$ as well. Furthermore, $\mathsf{Disj}(B_1, \exists U)$ and $B_1 \sqsubseteq \exists U$ makes $\mathsf{Disj}(B_1, B_1) \in cln(\mathcal{T})$, so finally $chase_i(\mathcal{O}) \not\models cln(\mathcal{T})$. Contradiction.

Similarly we reason when the generating rule is of the form $R_1 \sqsubseteq R_2$ and $U_1 \sqsubseteq U_2$. □

5.1.3 Ontology satisfiability

Lemma 7. *$can(\mathcal{O}) \models \mathcal{O}$ iff $\mathcal{O}$ is satisfiable.*

Proof. ($\Leftarrow$). Obviously follows.
($\Rightarrow$). We assume contrary, that $can(\mathcal{O})$ is not a model of $\mathcal{O}$ and that $\mathcal{O}$ is satisfiable by a model $\mathcal{M}$.

Considering Lemma 6., it follows $DB(\mathcal{A}) \not\models cln(\mathcal{T})$. Then exists some NI, functional constraint, or datatype restriction $\alpha \in cln(\mathcal{T})$, s.t. $DB(\mathcal{A}) \not\models \alpha$. Finally, from $cln(\mathcal{T}) \models \alpha$ and Lemma 5 we know that $\mathcal{T} \models \alpha$.

Now, $\mathcal{M} \not\models \alpha$ as $DB(\mathcal{A}) \subseteq \mathcal{M}$ and $\mathcal{M} \models \alpha$ as $\mathcal{M} \models \mathcal{T}$. Contradiction. □

Lemma 8. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$^{(\mathcal{HF})}_{core} + \mathcal{D}_1$ ontology. Then $DB(\mathcal{A}) \models cln(\mathcal{T})$ iff $\mathcal{O}$ is satisfiable.*

Proof. Follows directly from Lemmas 7. and 6. □

The Lemma 8. provides a guide for constructing a FOL query that will check satisfiability of DL-Lite$^{(\mathcal{HF})}_{core} + \mathcal{D}_1$ ontology . We construct a boolean FOL query $\varphi(\mathcal{T})$, that evaluated over $DB(\mathcal{A})$ provide us direct answer whether the ontology is satisfiability or not:

$$\varphi(\mathcal{T}) = \bigvee_{\alpha \in cln(\mathcal{T})} \varphi(\alpha) \tag{5.1}$$

where rewriting $\varphi(\alpha)$ is:

$$\begin{aligned}
&\varphi((\mathsf{funct}\ R)) = \exists x \exists y \exists z . R(x, y) \wedge R(x, z) \wedge y \neq z,\\
&\varphi((\mathsf{funct}\ U)) = \exists x \exists v \exists w . U(x, v) \wedge U(x, w) \wedge v \neq w,\\
&\varphi(\mathsf{Disj}(B_1, B_2)) = \exists x . \varphi(B_1(x)) \wedge \varphi(B_2(x)),\\
&\varphi(\mathsf{Disj}(R_1, R_2)) = \exists x \exists y . \varphi(R_1(x, y)) \wedge \varphi(R_2(x, y)),\\
&\varphi(\mathsf{Disj}(U_1, U_2)) = \exists x \exists v . U_1(x, v) \wedge U_2(x, v),\\
&\varphi(\mathsf{Rng}(U) \sqsubseteq T_i) = \exists x \exists v . U(x, v) \wedge \neg T_i(v),
\end{aligned}$$

where $\varphi(A(x)) = A(x)$, $\varphi(\exists P(x)) = \exists y.P(x,y)$, $\varphi(\exists P^-(x)) = \exists y.P(y,x)$, $\varphi(\exists U(x)) = \exists v.U(x,v)$, $\varphi(P(x,y)) = P(x,y)$, $\varphi(P^-(x,y)) = P(y,x)$.

To summarize, from the Lemma 8 we know that $(\varphi(\mathcal{T}))^{DB(\mathcal{A})} = \bot$ iff $\mathcal{O}$ is satisfiable.

We observe that $\mathcal{A}$ remains intact in constructing $\varphi(\mathcal{T})$, and the size of $\varphi(\mathcal{T})$ depends only on $\mathcal{T}$. As a consequence, data complexity of a *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ ontology satisfiability will be data complexity of FOL query answering over $DB(\mathcal{A})$, which in AC^0 wrt the size of the ABox. Considering previously stated we conclude:

Theorem 4. *Satisfiability problem is FOL rewritable in DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$.

5.1.4 Certain answers

In the following part we are proving that *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1 + \mathsf{UCQ}$ is FOL-rewritable. Similarly, to the satisfiability part, we are "strongly" exploiting the idea of canonical model.

We will show that query answers over a canonical model matches the certain answers in the case of satisfiable ontology. Lastly, we will use $can(\mathcal{O})$ as a guide for the construction of a perfect reformulation of an given UCQ. A FOL query is a perfect reformulation, if evaluating it against the ABox we obtain all certain answers. If we can construct perfect reformulation for a given ontology and a UCQ query over it, then we have proven FOL- rewritability of *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1 + \mathsf{UCQ}$.

The following lemma adopts the lemma from [10], that holds for *DL-Lite*$_{core}^{(\mathcal{HF})}$ ontologies. We will denote with $\mathcal{V}(A)$ the set of all atomic predicates (concepts, roles, attributes) in our vocabulary $\mathcal{V}$, and with $\mathcal{V}(\mathcal{D})$ the set of datatype names in our vocabulary $\mathcal{V}$.

Lemma 9. *Let $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ be a satisfiable DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ *ontology, and let $\mathcal{M} = (\Delta^{\mathcal{M}}, \cdot^{\mathcal{M}})$ be a model of $\mathcal{O}$. Then, there is a homomorphism from $can(\mathcal{O})$ to $\mathcal{M}$ over predicate set $\mathcal{V}(A)$.*

Proof. We are constructing a homomorphism μ form $can(\mathcal{O})$ to $\mathcal{M}$, by induction over the construction of $can(\mathcal{O})$. In addition, for each step i in the induction, we show that:

(hom.) For each objects $o, o' \in \Delta_O^{can_i(\mathcal{O})}$ and value $v \in \Delta_V^{can_i(\mathcal{O})}$ holds:
for any atomic concept A from $\mathcal{O}$, if $o \in A^{can_i(\mathcal{O})}$ then $\mu(o) \in A^{\mathcal{M}}$,
for any atomic role P from $\mathcal{O}$, if $(o, o') \in P^{can_i(\mathcal{O})}$ then $(\mu(o), \mu(o')) \in P^{\mathcal{M}}$,
for any atomic attribute U from $\mathcal{O}$, if $(o, v) \in U^{can_i(\mathcal{O})}$ then $(\mu(o), \mu(v)) \in U^{\mathcal{M}}$.

If the property *(hom.)* holds for each i, then it immediately establishes μ as a homomorphism form $can(\mathcal{O})$ to $\mathcal{M}$.

Base. For each object constant o that appears in $\mathcal{A}$ we set $\mu(o) = o^{\mathcal{M}}$. For each datatype constant d that appears in $\mathcal{A}$ we set $\mu(d) = d^{\mathcal{M}} (= d^{\mathcal{D}})$. Since $\mathcal{M} \models \mathcal{A}$, *(hom.)* property follows directly for $chase_0(\mathcal{O})$.

Induction step. Here we distinguish several cases based on the assertion β and the applied rule α in obtaining $can_{i+1}(\mathcal{O})$ from $can_i(\mathcal{O})$. For example, assume $\alpha = B \sqsubseteq \exists U$ and $\beta = B(o)$, where $B(o)$ denotes a membership assertion of

the form $A(o)$, $R(o, o_1)$ or $U(o, d_1)$. Then $can_{i+1}(\mathcal{O}) = can_i(\mathcal{O}) \cup U(o, (d_{new})^{\mathcal{D}})$, where d_{new} is a fresh datatype constant from $\mathsf{Rng}^{\mathcal{T}}_{max}(U)$. According induction hypothesis, exists $o_m \in \Delta^{\mathcal{M}}$ s.t. $\mu(o) = o_m$ and $o_m \in A^{\mathcal{M}}$. Since $\mathcal{M} \models \alpha$ then must exists some datatype constant d' s.t. $(o_m, (d')^{\mathcal{M}}) \in U^{\mathcal{M}}$. We set $\mu(d_{new}) = (d')^{\mathcal{M}}$ and conclude $(\mu(o), \mu(d_{new})) \in U^{\mathcal{M}}$. This together with ind. hypothesis that *(hom.)* property holds for $can_i(\mathcal{O})$ implies that *(hom.)* property holds for $can_{i+1}(\mathcal{O})$ as well.

Similarly, we can reason for $can_{i+1}(\mathcal{O})$ in other cases of α and β. □

Comment. Notice that in the theorem above, considered homomorphisms are over the set of predicates from $\mathcal{V}(A)$, and not including $\mathcal{V}(\mathcal{D})$. Then a question could be, does the theorem holds if the homomorphisms are over the set of predicates $\mathcal{P} = \mathcal{V}(A) \cup \mathcal{V}(\mathcal{D})$. The answer is no, and it is confirmed with the following example.

Example 5 We extend the example 2, with two datatypes:

$$\mathsf{myowl : studentAge} := \mathsf{xsd : int}[\geq_{16}], \quad \mathsf{myowl : oldStudentAge} := \mathsf{xsd : int}[\geq_{25}]$$

, and a new attribute *hasAge*, that states the age of a student:

$$\mathit{Student} \sqsubseteq \exists \mathit{hasAge}, \quad \mathsf{Rng}(\mathit{hasAge}) \sqsubseteq \mathsf{myowl : studentAge}, \quad (\mathsf{funct}\ \mathit{hasAge})$$

Additionally, assume that datatype constants $\text{"16"}^{\wedge\wedge}\mathsf{xsd : int}, \ldots, \text{"24"}^{\wedge\wedge}\mathsf{xsd : int}$ are already present in the ABox (*). Also assume that in the ABox we have fact $\mathit{Student}(\mathsf{John})$ but no $\mathit{hasAge}(\mathsf{John}, d)$ for a datatype constant d.

Then for each model of the ontology, must exists a datatype constant d from $\mathsf{myowl : studentAge}$. For the canonical model $can(\mathcal{O})$, considering (*) there exists exactly one constant $d^{\mathcal{D}} \in \mathsf{myowl : oldStudentAge}^{\mathcal{D}}$.

However, this doesn't necessarily hold for all models, and let $\mathcal{M}$ be a model of the ontology, where $(\mathsf{John}, \text{"20"}^{\wedge\wedge}\mathsf{xsd : int}) \in \mathit{hasAge}^{\mathcal{M}}$.

Finally, there is no homomorphism between $can(\mathcal{O})$ and $\mathcal{M}$ over $\mathcal{V}(A) \cup \mathcal{V}(\mathcal{D})$, since d must be mapped to $\text{"20"}^{\wedge\wedge}\mathsf{xsd : int}$, but then $(\text{"20"}^{\wedge\wedge}\mathsf{xsd : int})^{\mathcal{D}} \notin (\mathsf{myowl : oldStudentAge})^{\mathcal{D}}$ while $d^{\mathcal{D}} \in (\mathsf{myowl : oldStudentAge})^{\mathcal{D}}$.

△

Exploiting the previous lemma we will show that a canonical model $can(\mathcal{O})$ properly represents all models of $\mathcal{O}$ wrt certain answers, i.e. certain answers are matching the answers of the canonical model for UCQ queries.

Proposition 2. *Let q be a* UCQ *over a satisfiable DL-Lite$^{(\mathcal{HF})}_{core} + \mathcal{D}_1$ ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ and let t be a tuple of constants that from $\mathcal{A}$. Then,*

$$t \in cert(q, \mathcal{O}) \quad \text{iff} \quad t^{can(\mathcal{O})} \in q^{can(\mathcal{O})}$$

Proof. ($\Rightarrow$). Follows from the definition of a certain answer.
($\Leftarrow$). Then if q is UCQ composed with CQ $\{q_1, \ldots, q_n\}$, there must be at least one $q_i(x) \leftarrow conj_i(x, y)$ s.t. $t^{can(\mathcal{O})} \in q_i^{can(\mathcal{O})}$. According Theorem 1. there is a homomorphism μ from $conj_i(t, y)$ to $can(\mathcal{O})$.

Now, let $\mathcal{M}$ be an arbitrary model of $\mathcal{O}$. According Lemma 9. there exists a homomorphism η from $can(\mathcal{O})$ to $\mathcal{M}$. Since, homomorphisms are closed under the composition we get that $\eta \circ \mu$ is a homomorphism from $conj_i(t, y)$ to $\mathcal{M}$. Again using Theorem 1. we conclude that $t^{\mathcal{M}} \in q^{\mathcal{M}}$. Since, $\mathcal{M}$ is chosen arbitrary it follows that $t \in cert(q, \mathcal{O})$. □

The next lemma states that certain answers of a UCQ query can be obtained from the certain answers of its CQs.

Lemma 10. *For an DL-Lite$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ ontology O and a UCQ q over it, holds:*

$$cert(q, \mathcal{O}) = \bigcup_{q_i \in q} cert(q_i, \mathcal{O})$$

Proof. ($\subseteq$:) By definition.

($\supseteq$:) If the ontology is unsatisfiable then it follows trivially. Otherwise, let t be a tuple from $cert(q, \mathcal{O})$. Then from Proposition 2. $t^{can(\mathcal{O})} \in q^{can(\mathcal{O})}$. It follows that exists some CQ q_i in q, $t^{can(\mathcal{O})} \in q_i^{can(\mathcal{O})}$. Then again according Proposition 2. $t \in cert(q_i, \mathcal{O})$. □

Perfect rewriting

In this part we are constructing a FOL query, called *perfect rewriting*, based on a given UCQ q and DL-Lite$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ TBox $\mathcal{T}$. Perfect rewriting is an evidence that confirms FOL-rewritability. To remind, answering UCQ is FOL-rewritable in DL-Lite$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$, if for every DL-Lite$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ TBox $\mathcal{T}$ and UCQ q, there exists is a FOL query q_{pr} (perfect rewriting), such that for every *DL-Lite* ABox $\mathcal{A}$, ${q_{pr}}^{DB(\mathcal{A})} = cert(q, \mathcal{T})$.

So far, we have see that satisfiability of a DL-Lite$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ ontology is FOL rewritable. Further, we proved that one can obtain certain answers over UCQs by evaluating it against the canonical model 2.

However, a canonical model can be infinite, so it will not be feasible to implement it. Despite this, a canonical model can be used as a "guide" in constructing prefect reformulation (see later the proof of Theorem 6.1.2). But be before that we define some necessary definitions.

We denote with $B(x)$ any atom of the form $A(x)$, $R(x, \_)$, $U(\_, x)$ or $U(x, \_)$, where,

$$R(x, \_) = \begin{cases} P(x, \_) & \text{for} \quad R = P \\ P(\_, x) & \text{for} \quad R = P^- \end{cases}$$

We say that an argument of an atom in a query is bounded if it corresponds to either a distinguished variable or a shared variable, i.e., a variable occurring at least twice in the query body, or a constant. Instead, an argument of an atom in a query is unbound if it corresponds to a non-distinguished non-shared variable. As usual, we use the symbol "_" to represent non-distinguished non-shared variables. Using this, we define whether a PI is *applicable* to an atom:

- A PI α is applicable to an atom $A(x)$, if α has A on its right-hand side.
- A PI α is applicable to an atom $P(x_1, x_2)$, if one of the following conditions hold:
  - $x_2 = \_$ and the right-hand side of α is $\exists P$ or
  - $x_1 = \_$ and the right-hand side of α is $\exists P^-$ or
  - α is a role inclusion and its right-hand side is either P or P^-.
- A PI α is applicable to an atom $U(x_1, x_2)$, if one of the following conditions hold:

– $x_2 = \_$ and the right-hand side of α is $\exists U$ or

– α is a attribute inclusion assertion and its right-hand side is U.

In the Figure1, we provide the algorithm $\mathsf{PerfectRef}(q, \mathcal{T})$, which reformulates UCQ q (considered as a set of CQs) by taking into account the PIs of a TBox $\mathcal{T}$. Two functions are presented there:

- $\mathsf{rewrite}(q, g, \alpha)$. We denote with $B(x)$ any of the atoms of the form $A(x)$, $R(x, \_)$, $U(x, \_)$, where $R(x, \_) = P(x, \_)$ when $R = P$ and $R(x, \_) = P(\_, x)$ when $R = P^-$. Also, we denote with B any of the *DL-Lite* concept expressions of the form A, $\exists R$ or $\exists U$. Additionally, for a given B we define $B(x)$ as $A(x)$ when $A = B$, $R(x, \_)$ when $B = \exists R$, $U(x, \_)$ when $B = \exists U$. Finally, we define rewriting $(\mathsf{rw})(q, \alpha)$, for a given atom g and a given PI α according the table:

g	α	$(\mathsf{rw})(q, \alpha)$
$B_1(x)$	$B_2 \sqsubseteq B_1$	$B_2(x)$
$R_1(x_1, x_2)$	$R_2 \sqsubseteq R_1$	$R_2(x_1, x_2)$
$U_1(x_1, x_2)$	$U_2 \sqsubseteq U_1$	$U_2(x_1, x_2)$

  An exceptional case is the rule of the form $\alpha = \top \sqsubseteq \exists Id$. If $q' = \mathsf{rewrite}(q, Id(x_1, \_), \alpha)$ applies rule α, then q' is obtained from q by replacing $Id(x_1, \_)$ with the top concept $\top(x_1)$. As usual, the semantics of the top concept is defined with $(\top)^{\mathcal{I}} = \Delta_O^{\mathcal{I}}$, for an arbitrary model $\mathcal{I}$.

  In other cases, $\mathsf{rewrite}(q, g, \alpha)$ is a CQ obtained from q by replacing the atom q with a new atom $(\mathsf{rw})(q, \alpha)$.

- $\mathsf{reduce}(q, g_1, g_2)$. Method $\mathsf{reduce}(q, g_1, g_2)$ creates a new CQ form q by applying two operations on q. Firstly, it calculates the most general unifier (mgu) ν of g_1 and q_2, and it applies ν over q to obtain $q' = \nu(q)$. This is done with remark, that in unifying g_1 and g_2, each occurrence of the symbol _ has to be considered as a different unbound variable.

  After unification, new variables can become unbound variables. Therefore, the second operation is replacement of the unbound variables with _ symbol (say from q' we obtain q'' by this operation). Notice that, possible new rules will be applicable in q'', that were not before in q.

  Finally, a CQ q" = $\mathsf{reduce}(q, g_1, g_2)$ is returned as a result.

An import remark concerns the size of the UCQ query q_{pr}. Namely, the size of each CQ in q_{pr} is less or equal then the longest CQ from q. Simply because, both functions $\mathsf{rewrite}(\cdot, \cdot, \cdot)$ and $\mathsf{reduce}(\cdot, \cdot, \cdot)$ always produce a CQ that is less or equal then the size of the original CQ. So each CQ in q_{pr} is less or equal to $|q|$, where $|\cdot|$ denotes the size measure.

Further, new CQs in q_{pr} can only be built using variables, constants and atoms from q and $\mathcal{T}$. Which means, there are at most $|\mathcal{T}| \times |q|^2$ possible CQs in q_{pr}.

To summarize, the size of each CQ in q_{pr} less or equal to $|q|$ and there are at most $|\mathcal{T}| \times |q|^2$ of them in q_{pr}. In other words:

$$|q_{pr}| \leq (|\mathcal{T}| \times |q|^2)^{|q|}. \tag{5.2}$$

Algorithm 1 The algorithm that computes PerfectRef of a UCQ q given a $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ TBox $\mathcal{T}$

Input: UCQ q, $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ TBox $\mathcal{T}$
Output: UCQ q_{pr}

```
1: procedure PerfectRef(q, T)
2:     q_pr ← q
3:     repeat
4:         q'_pr ← q_pr
5:         for all CQ q in q_pr do
6:             for all atoms g in q do                          ▷ step (rw)
7:                 for all PI α in T do
8:                     if α is applicable to g then
9:                         q'_pr ← q'_pr ∪ rewrite(q, g, α);
10:                    end if
11:                end for
12:            end for
13:            for all pair of atoms g1, g2 in q do             ▷ step (red)
14:                if g1 and g2 unify then
15:                    q'_pr ← q'_pr ∪ reduce(q, g1, g2);
16:                end if
17:            end for
18:        end for
19:    until q_pr = q'_pr
20: return q_pr;
21: end procedure
```

Although it might seems obvious, the important consequence of inequality 5.2 is the termination of the PerfectRef algorithm. Complexity consequences are presented later.

The last step in proving that query answering of UCQs is FOL rewritable in $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1$, is the next theorem.

Theorem 5. *Given a satisfiable* $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ *ontology* $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ *and a* UCQ *query* q *over it, we claim that holds:*

$$(q_{pr})^{DB(\mathcal{A})} = cert(q, \mathcal{O})$$

,where $q_{pr} = \mathsf{PerfectRef}(q, \mathcal{T})$.

Proof. Firstly we define the notion of a witness. Given an ontology $\mathcal{O}$, a tuple t of constants from $\mathcal{O}$, and a CQ $q(x)$ over $\mathcal{O}$, we say that a set of membership assertion Θ, is a witness for t wrt q, if there exists substation σ ($\{x \mapsto t\} \in \sigma$) over atoms in q s.t. $\sigma(q(x)) = \Theta$ (q is considered set-wise).

Considering Lemma 10 we have that $(q_{pr})^{DB(\mathcal{A})} = \bigcup_{\bar{q} \in q_{pr}} \bar{q}^{DB(\mathcal{A})}$. Moreover, considering Proposition 2, it will be sufficient to show:

$$\bigcup_{\bar{q} \in q_{pr}} \bar{q}^{DB(\mathcal{A})} = q^{can(\mathcal{O})}$$

($\subseteq$:) We have to show that for an arbitrary $\bar{q} \in q_{pr}$, $\bar{q}^{DB(\mathcal{A})} \subseteq q^{can(\mathcal{O})}$.

We will show it by induction over the number of steps (rw) and (red) used for generating $\bar{q}$.

Induction base. No reformulation applied, so $\bar{q} \in q$. Then $\bar{q}^{DB(\mathcal{A})} \subseteq \bar{q}^{can(\mathcal{O})} \subseteq q^{can(\mathcal{O})}$.

Induction step. Induction hypothesis states that for all $\bar{q} \in q_{pr}$ s.t $\bar{q}$ is generated applying steps (rw) and (red) less then $n+1$ times , then $\bar{q}^{DB(\mathcal{A})} \subseteq \bar{q}^{can(\mathcal{O})} \subseteq q^{can(\mathcal{O})}$. It will be sufficient to show that for each tuple t in $can(\mathcal{O})$ and for each $\bar{q_2}$ generated in $(n+1)$ steps, holds: if $t^{can(\mathcal{O})} \in \bar{q_2}^{can(\mathcal{O})}$ then $t^{can(\mathcal{O})} \in q^{can(\mathcal{O})}$.

Firstly, we assume that $\bar{q_2}$ is generated in $(n+1)$ steps, where the last step was (rw) on the rule $\beta : A \sqsubseteq \exists U$, i.e. $\bar{q_2} = \mathsf{rewrite}(\bar{q_1}, \beta, U)$. By assumption, $\bar{q_1}^{can(\mathcal{O})} \subseteq q^{can(\mathcal{O})}$. If for a tuple t from $\mathcal{O}$, we have a witness Θ wrt $\bar{q_2}$ in $can(\mathcal{O})$, then there are two possibilities. Firstly, that Θ is already a witness for t wrt $\bar{q_1}$. If not, then must be a ground atom $A(o_i) \in \Theta$ that applied on β in $can(\mathcal{O})$ to produce a new fact $U(o_i, d_{new})$. Then either $\Theta \cup \{U(o_i, d_{new})\}$ or $(\Theta \setminus \{A(o_i)\}) \cup \{U(o_i, d_{new})\}$ is a witness for t wrt q_1 from $can(\mathcal{O})$.

Finally, in both cases, $t^{can(\mathcal{O})} \in \bar{q_1}^{can(\mathcal{O})}$. Considering the i.h. it holds $t^{can(\mathcal{O})} \in \bar{q}^{can(\mathcal{O})}$.

Secondly, let's assume $\bar{q_2} = \mathsf{rewrite}(\bar{q_1}, \beta, Id)$, where $\beta = \top \sqsubseteq Id$. Also, let's assume Θ is a witness for t wrt q_2, i.e. $t^{can(\mathcal{O})} \in q_2^{can(\mathcal{O})}$.

Top concept $\top(x)$ is satisfied with each object constant from $can(\mathcal{O})$. Hence, Θ is a witness for t wrt q_1 as well. Finally, according i.h. $t^{can(\mathcal{O})} \in q_2^{can(\mathcal{O})} \subseteq q_1^{can(\mathcal{O})} \subseteq q^{can(\mathcal{O})}$.

Similarly, we can prove other cases for (rw) step.

If $\bar{q_2} = \mathsf{reduce}(\bar{q_1}, g_1, g_2)$, then every witness for t wrt $\bar{q_2}$ is also a witness wrt $\bar{q_1}$.

($\supseteq$:) Assume $t^{can(\mathcal{O})} \in q^{can(\mathcal{O})}$ and that Θ is a witness. Then exists a CQ $q' \in q$ for which Θ is the witness. We are constructing $\bar{q}$ from q_{pr} s.t. $t \in \bar{q}^{DB(\mathcal{A})}$.

The canonical model $can(\mathcal{O})$ can be seen as a forest, where nodes are membership assertions in $can(\mathcal{O})$ and edges correspond to the rules that generate new membership assertions from $\mathcal{A}$. In particular, all facts from $\mathcal{A}$ we consider as roots. $can(\mathcal{O})$ is truly a forest because no node name are duplicated and there is at most one incoming edge for each node.

Out of this forest, potentially infinite, we will consider a minimal as possible (final) sub-forest $\mathcal{F}$, where leaves are the assertions in Θ, roots are the roots of the assertions in Θ, and edges are one that establish paths between the leaves and the roots, s.t. Θ is contained in the set of nodes from $\mathcal{F}$. Notice that each membership assertion can appear only once in $\mathcal{F}$. Notice also, that not necessarily all elements from Θ are leaves in $\mathcal{F}$.

The idea is to construct $\bar{q}$ starting from q' and then following the rules (inverted) that generate Θ, directing from the leaves towards roots. Assume that $\mathcal{F}$ has n edges. We define a chain of sub-forests $\mathcal{F}_0 = \mathcal{F}, \mathcal{F}_1, \ldots, \mathcal{F}_n$ of $\mathcal{F}$, where $\mathcal{F}_{i+1}$ is obtained from $\mathcal{F}_i$ by deleting one leaf. Additionally we define the list of queries $\bar{q}_0, \bar{q}_1, \ldots, \bar{q}_n$, where $\bar{q}_{i+1}$ is obtained from $\bar{q}_i$, according the rule β_i that corresponds to the deleted edge in $\mathcal{F}_i$ in obtaining $\mathcal{F}_{i+1}$. Additionally, following the witness substitution for t from $\mathcal{F}_{i+1}$ wrt $\bar{q}_{i+1}$, we apply Reduction step (see bellow) on each $\bar{q}_{i+1}$. This is done, to ensure correct (rw) application for $\bar{q}_{i+1}$ in the next step.

Two criteria we have to show: (i) the rule β_i is applicable to corresponding atom in q_i, (ii)(induction hypothesis) and for each i there exists a witness in $\mathcal{F}_i$ for t wrt $\bar{q}_i$ s.t. no two atoms in $\bar{q}_i$ are assigned by the witness substitution to the same membership assertion from the witness (injection between atoms in $\bar{q}_i$ and the elements of a witness). As a consequence (i) we have that $\bar{q}_n$ is generated from q applying (rw) and (red) rules and that (ii) there is a witness in $\mathcal{F}_n \subseteq DB(\mathcal{A})$ for t wrt $\bar{q}_n$, i.e. $t \in \bar{q}_n^{DB(\mathcal{A})}$.

Reduction step on $\bar{q}_i$. Each witness Θ_i determines a substitution σ, where $\sigma(\bar{q}_i(x)) = \Theta_i$ and $\{x \mapsto t\} \in \sigma$. Following this, we will apply (red) step to all pairs of atoms in $\bar{q}_i$ that become identical after applying substitution σ. After this atoms in $\bar{q}_i$ and assertions in Θ relates 1 to 1.

Induction base. Let's denote with $\bar{q}_0$ a query obtained from q after applying Reduction step. So, no two atoms in $\bar{q}_0$ relates to the same assertion in $\mathcal{F}_0$(injection). Nevertheless, $\sigma(q') = \sigma(\bar{q}_0)$(set-wise) and $\Theta \subseteq \mathcal{F}_0$ is a witness of t wrt $\bar{q}_0$.

Induction step. Assume that that exists a witness in $\mathcal{F}_i$ for t wrt $\bar{q}_i$. Now let $\mathcal{F}_{i+1} = \mathcal{F}_i \setminus \gamma$ where the edge label is rule β_i and γ_i is deleted membership assertion. If β_i has a form $A_1 \sqsubseteq A$, $R_1 \sqsubseteq R_2$ or $U_1 \sqsubseteq U_2$ than it is very straightforward to show that there exists a witness in $\mathcal{F}_{i+1}$ for t wrt $\bar{q}_{i+1} = \mathsf{rewrite}(\bar{q}_i, \beta_i, A_1)$ (or U_1 or R_1), and that (rw) step is correctly applied.

Slightly more complicated cases are $B \sqsubseteq \exists R$ and $B \sqsubseteq \exists U$, because the correct application of a step (rw) isn't straightforward.

So, lets assume $\beta = A \sqsubseteq \exists P$ and $\gamma_i = P(o, o_{new})$ (the case $B \sqsubseteq \exists U$ is analogous). As $\gamma_i = P(o, o_{new})$ is a leaf of $\mathcal{F}_i$ then o_{new} doesn't appear in any other membership assertion from $\mathcal{F}_i$. This means that a variable y in an atom $P(z, y) \in \bar{q}_i$ that corresponds to $P(o, o_{new})$ is non distinguishable one. Additionally y is not shared one, for the two reasons. Firstly there is no another

atom $P(z_2, y) \in \bar{q}_i$ that corresponds to $P(o, o_{new})$ as Reduction step is applied on $\bar{q}_i$. Secondly, y can not appear in some other atom, for example $A(y) \in \bar{q}_i$, because the witness defines 1-1 (injection) relation between atoms in q_i and nodes in $\mathcal{F}_i$. So after application of Reduction step, $P(z, y)$ must be rewritten in $P(z, \_) \in \bar{q}_i$ and the rewriting using β to obtain $\bar{q}_{i+1}$ is correct. □

We summarize the steps that need to be taken in order to evaluate a UCQ q against a *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$:

SAT Check ontology satisfiability by evaluating boolean query $\varphi(\mathcal{T})$(5.1) over $DB(\mathcal{A})$. If $(\varphi(\mathcal{T}))^{DB(\mathcal{A})} = \mathit{true}$ then ontology $\mathcal{O}$ is unsatisfiable. From FOL point of view, $\mathcal{O}$ entails everything and then for any query q the answer should be all possible tuples of the arity of q that can be constructed from the constants in $\mathcal{O}$. However, this is not feasible in the practice and a message should be printed that address inconsistency of $\mathcal{O}$.

If $(\varphi(\mathcal{T}))^{DB(\mathcal{A})} = \mathit{false}$, then $\mathcal{O}$ is satisfiable and we can move to the next step.

$q_{pr}^{DB(\mathcal{A})}$ First step is to calculate perfect reformulation of q according PerfectRef algorithm 1, $q_{pr} = \mathsf{PerfectRef}(q, \mathcal{T})$. Then q_{pr} should be evaluated against an ABox $\mathcal{A}$. If we consider that $\mathcal{A}$ is stored in a database $DB(\mathcal{A})$ than q_{pr} should be reformulated as a SQL query and posed against $DB(\mathcal{A})$. q_{pr} reformulation in SQL format is not a hard task, taking into account that every CQ corresponds to a `select-project-join`, where `select` corresponds to atom names in a CQ, `project` corresponds to distinguishable variables and `join` corresponds to shared variables. Moreover, disjunction of CQs directly corresponds to SQL statement `union`.

Unfortunately, the data in the real world application are not stored as tables with one or two columns, but as tuples of strictly typed constants. In this case, one have to establish mapping language between database and ontology, that can express correlation between typed data in the database and ontology objects. This problem is called *impedance mismatch problem.*

The problem and a solution for this are addressed in [10, p. 329]. The authors proposed a mapping language between databases and *DL-Lite* ontologies. The mappings are not materialized, i.e. the ABox is not instantiated, but they are used in a step where perfect reformulation is rewritten in a SQL query. Finally, they proved the correctness of the methods, that reformulated SQL provides the same answer as if they materialize the data throughout the mappings.

Example 5.12. Let us consider again the query of Example 5.9

Computational complexity

From everything that has been shown in this chapter we conclude

Theorem 6. *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1 +$ UCQ *is FOL-rewritable.*

Here we summarize complexity measure of *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1 + UCQ$.

- $\mathsf{PerfectRef}(q, \mathcal{T}) \in \text{PTIME}(\mathcal{T})$.
  The meaning is that $\mathsf{PerfectRef}(q, \mathcal{T})$ is a polynomial algorithm in the size of $\mathcal{T}$. This is a consequence of the formula 5.2.

- (Data complexity) Answering UCQs in $\mathit{DL\text{-}Lite}_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ is in PTIME in the size of the ontology.

  It follows from the facts. Firstly, $\mathit{cln}(\mathcal{T})$ is polynomial in the size of $\mathcal{T}$ and that each CQ in $\varphi(\mathcal{T})$5.1 is bounded in size and has at maximum three variables. Each CQ in $\varphi(\mathcal{T})$ can be evaluated separately over the ABox. Second fact is stated in the point above: $\mathsf{PerfectRef}(q, \mathcal{T}) \in \text{PTIME}(\mathcal{T})$.

- (Schema complexity) Answering UCQs in $\mathit{DL\text{-}Lite}_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ is in AC^0 in the size of the ABox.

  Answering UCQ queries over a database is in AC^0 for the data complexity. Firstly, both $\varphi(\mathcal{T})$ and q_{pr} do not depend on the ABox. Secondly, the ABox is only addressed in satisfiability check and query evaluating this is done by evaluate $\varphi(\mathcal{T})$ and q_{pr} against it (resp.). To conclude, both steps of the UCQ query answering process over $\mathit{DL\text{-}Lite}_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ are in AC^0, so the query answering is then in AC^0 as well.

- (Combined complexity) Answering UCQs in $\mathit{DL\text{-}Lite}_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ is NP-complete in combined complexity.

  Lower bounds comes from the combined complexity of UCQ evaluation over the databases.

  To show that answering UCQs in $\mathit{DL\text{-}Lite}_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ is NP, we need to describe non-deterministic evaluation of perfect reformulation. According the proof of theorem 6.1.2, for each tuple t that is a certain answer, there is a CQ in q_{pr} that can be computed from q applying at most polynomial number of (rw) and (red) step such that exists a witness for t in $\mathit{DB}(\mathcal{A})$. The same holds if the input query is a boolean UCQ. To conclude, given a boolean UCQ q we non-deterministically compute a CQ from q_{pr} which evaluates to true over $\mathit{DB}(\mathcal{A})$ iff q evaluates to true under $\mathit{can}(\mathcal{O})$ iff q is entailed by the ontology.

To summarize, the above results show that query answering in $\mathit{DL\text{-}Lite}_{core}^{(\mathcal{HF})} + \mathcal{D}_1 + \mathsf{UCQ}$ is computationally no worse than answering UCQs over the ABox only, i.e. UCQ answering in standard databases.

5.2 Relaxing datatype conditions

In this section we will prove that condition (infinite) over datatype lattices is necessary condition for FOL-rewritability of UCQ answering over $\mathit{DL\text{-}Lite}_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ ontologies. The prove is based on the reduction of the coNP-hard problems to the query answering in $\mathit{DL\text{-}Lite}_{core}^{(\mathcal{HF})} + \mathcal{D} + \mathsf{UCQ}$, where $\mathcal{D}$ violates (infinite) condition.

We remind the reader, that data complexity of FOL query evaluation is in AC^0 complexity class, which is strictly contained in coNP. Which means, if a problem is coNP-hard, it can't be solved with the method that is in AC^0. We will encode coNP-hard problem into the problem of UCQ answering an

ontology, where TBox and query are fixed (bounded) and only the ABox is adopting wrt the problem instance. Consequently, that data complexity of the query answering is at least coNP-hard. In other words, we will not be able to rewrite a given UCQ into a FOL query that is perfect reformulation.

The encodings are expressed using only concept inclusions ($B_1 \sqsubseteq B_2$), disjointness between attributes ($\mathsf{Disj}(U_1, U_2,)$) and attribute ranges ($\mathsf{Rng}(U) \sqsubseteq T_i$). Role hierarchies and functional restrictions are not used. It means that already for *DL-Lite*$_{core}$ with attributes disjointness, (infinite) condition is necessary for the efficient UCQ answering.

Assume $k \geq 2$ is the number of datatype values in common of a violating set of datatypes. The prove is based on the two lemmas. In the first (lemma 11), we reduce the problem complement to 3-colorability problem into *UCQ* answering, for $k \geq 3$. In the second (lemma 12), we adopt the proof of lemma 21, that encodes 2+2-CNF unsatisfiability problem, for $k = 2$.

Lemma 11. *Let $\mathcal{O}$ be a DL-Lite$_{core}^{(\mathcal{HF})} + \mathcal{D}$ ontology defined over a datatype lattice $\mathcal{D}$ where exists $T_1, \ldots, T_k$ $(1 \leq k \leq n)$ s.t. they have exactly k values in common for $k \geq 3$, i.e. $|\prod_{i=1}^{k} T_i| = k \geq 3$. Then UCQ answering over $\mathcal{O}$ is not FOL rewritable.*

Proof. 3-COLOR is a decision problem where given an undirected graph, one has to compute whether it is possible to assign to each node one of the color from the set $\{red, blue, green\}$, such that no two adjustment vertices has the same color. The problem is NP-complete.

So assume we are given an undirected graph $G = (V, E)$ where $E \subseteq V \times V$ is a set of edges and V set of vertices. We define a *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}$ ontology $\mathcal{O}_G = \langle \mathcal{T}_G, \mathcal{A}_G \rangle$ where $\mathcal{T}_G$ is fixed and $|\mathcal{A}| \in \mathcal{O}(k \times |V| + |E|)$. Also we assume that datatype constants $\{d_1, \ldots, d_n\}$ are constants that corresponds to the k common datatype values. Then we set:

$$\mathcal{A}_G = \{A(v_i) | v_i \in V\} \cup \{E(v_i, v_j) | (v_i, v_j) \in E\} \cup \{V(v_i, d_k) | 1 \leq i \leq n \wedge k \geq 4\},$$

$$\mathcal{T}_G = \{A \sqsubseteq \exists U, V \sqsubseteq \neg U, \mathsf{Rng}(U) \sqsubseteq \prod_{i=1}^{k} T_i\}$$

and

$$q() \leftarrow U(x_1, c), U(x_2, c), E(x_1, x_2).$$

We claim

$$G \text{ is 3-COLOR iff } () \notin \mathit{cert}(\mathcal{O}_G, q())$$

($\Rightarrow$:). G is 3-COLOR then exists $\beta : V \rightarrow \{d_1, d_2, d_3\}$, s.t. if $(v_i, v_j) \in E$ then $\alpha(v_i) \neq \alpha(v_j)$. We use $\{d_1, d_2, d_3\}$ to denote $\{red, blue, green\}$ respectfully.

Based on this we define a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ s.t. $\mathcal{I} \models \mathcal{O}_G$ and $\mathcal{I} \not\models q()$.

Let $\{v_1, \ldots, v_n\} \cup T \subseteq \Delta^{\mathcal{I}}$, $v_i^{\mathcal{I}} = v_i$ and $d_j^{\mathcal{I}} = d_j^{\mathcal{D}} = d_j$. Atoms are interpreted as: $A^{\mathcal{I}} = \{v_i | v_i \in V\}$, $U^{\mathcal{I}} = \{(v_i, v_j) | (v_i, v_j) \in E\}$, $V^{\mathcal{I}} = \{(v_i, d_k) | 1 \leq i \leq n \wedge k \geq 4\}$, $U^{\mathcal{I}} = \{(v_i, \beta(v_i) | v_i \in V\}$.

Inclusions $A \sqsubseteq \exists U$, $V \sqsubseteq \neg U$ and $\mathsf{Rng}(U) \sqsubseteq \prod_{i=1}^{k} T_i$ are satisfied by $\mathcal{I}$ by definition. Finally, $I \not\models \exists x_1, x_2, c.U(x_1, c), U(x_2, c), E(x_1, x_2)$ because β assigns different colours to adjustment nodes.

($\Leftarrow$:) Now assume there exists a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ s.t. $\mathcal{I} \models \mathcal{O}_G$ and $\mathcal{I} \not\models q()$. Then take $U^{\mathcal{I}}$ relation to define an assignment $\beta : V \to \{d_1, d_2, d_3\}$. β is correctly defined as each $v \in V$ has some $d \in \prod_{i=1}^{k} T_i$ considering $A \sqsubseteq \exists U$ and $\mathsf{Rng}(U) \sqsubseteq \prod_{i=1}^{k} T_i$. On the other hand, $\mathsf{Rng}(U) \subseteq \{d_1, d_2, d_3\}$ because $V \sqsubseteq \neg U$. Finally as $I \not\models \exists x_1, x_2, c.U(x_1, c), U(x_2, c), E(x_1, x_2)$ we know that β assigns different colours to adjustment nodes and G is 3-colorable.

3-COLOR is an NP-hard problem, which means that the problem of answering UCQ over the ontology that violates condition (finate), is at least coNP-hard. On the other hand data complexity of FOL query answering is in AC^0, which means we wont be able to rewrite q in a FOL query that is perfect reformulation. □

Lemma 12. *Let $\mathcal{O}$ be a DL-Lite$_{core}^{(\mathcal{HF})}$+$\mathcal{D}$ ontology defined over a datatype lattice $\mathcal{D}$ where exists $T_1, \ldots, T_k$ ($1 \leq k \leq n$) s.t. they have exactly two data values in common, i.e. $|\prod_{i=1}^{k} T_i| = 2$. Then UCQ answering over $\mathcal{O}$ is not FOL-rewritable.*

Proof. We adopt the proof of lemma 21. The proof idea is the same, and we adopt only the encoding ontology $\mathcal{O}_F = \langle \mathcal{T}, \mathcal{A}_F \rangle$. Assume that $(\prod_{i=1}^{k} T_i)^{\mathcal{D}} = \{\mathbf{t}, \mathbf{f}\}$, where $(\mathit{true})^{\mathcal{D}} = \mathbf{t}$ and $(\mathit{false})^{\mathcal{D}} = \mathbf{f}$. We set,

$$
\begin{aligned}
\mathcal{T} &= \{O \sqsubseteq \exists U, \mathsf{Rng}(U) \sqsubseteq \prod_{i=1}^{k} T_i\} \\
&\quad \cup \{\mathsf{Rng}(U_{pos}) \sqsubseteq \prod_{i=1}^{k} T_i\} \cup \{U_{pos} \sqsubseteq \neg U^*_{pos}\} \\
&\quad \cup \{\mathsf{Rng}(U_{neg}) \sqsubseteq \prod_{i=1}^{k} T_i\} \cup \{U_{neg} \sqsubseteq \neg U^*_{neg}\}, \\
\mathcal{A}_F &= \{O(l_1), \ldots, O(l_m)\} \\
&\quad \cup \ \{P_1(c_k, l^k_{1+}), P_2(c_k, l^k_{2+}), N_1(c_k, l^k_{1-}), N_2(c_k, l^k_{2-}) | 1 \leq k \leq n\} \\
&\quad \cup \ U(l_{true}, \mathit{true}) \\
&\quad \cup \ U(l_{false}, \mathit{false}) \\
&\quad \cup \ \{U_{pos}(l^k_{1+}, \mathit{false}), U_{pos}(l^k_{2+}, \mathit{false}) | 1 \leq k \leq n\} \\
&\quad \cup \ \{U^*_{pos}(l^k_{1+}, \mathit{true}), U^*_{pos}(l^k_{2+}, \mathit{true}) | 1 \leq k \leq n\} \\
&\quad \cup \ \{U_{neg}(l^k_{1-}, \mathit{true}), U_{neg}(l^k_{2-}, \mathit{true}) | 1 \leq k \leq n\} \\
&\quad \cup \ \{U^*_{neg}(l^k_{1-}, \mathit{false}), U^*_{neg}(l^k_{2-}, \mathit{false}) | 1 \leq k \leq n\}
\end{aligned}
$$

and for a $\mathsf{CQ}_{\mathcal{D}}$:

$$\begin{aligned} q() \leftarrow & P_1(c, f_1), U(f_1, vf_1), U_{pos}(f_1, vf_1), \\ & P_2(c, f_2), U(f_2, vf_2), U_{pos}(f_2, vf_2), \\ & N_1(c, t_1), U(t_1, vt_1), U_{neg}(t_1, vt_1), \\ & N_2(c, t_2), U(t_2, vt_2), U_{neg}(t_2, vt_2). \end{aligned}$$

The rest of the proof proceeds in the same fashion as in the lemma 21. □

Chapter 6

OBDA framework: $\boldsymbol{DL\text{-}Lite}_{\boldsymbol{core}}^{(\mathcal{HF})} + \mathcal{D}_2 + \mathsf{UCQ}_{\mathcal{D}}$

In this chapter, we provide another significant example of OBDA framework, $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_2 + \mathsf{UCQ}_{\mathcal{D}}$. We provide reacher query language, but impose more conditions over admissible datatype lattices.

FOL-rewritability of satisfiability of $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ ontologies directly follows more general case of $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ satisfiability. However, we adopt the canonical model in order to prove the next step. In the next step, we further extend the algorithm, that for a given $\mathsf{UCQ}_{\mathcal{D}}$ constructs perfective reformulation over a satisfiable $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ ontology.

In the second part of the chapter, we prove necessity for the conditions in $\mathcal{D}_2$ in order to remain FOL-rewritable, by encoding coNP-hard problems into the problems of query answering.

6.1 Reasoning

In this section, we are aiming to prove FOL-rewritability of $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_2 + \mathsf{UCQ}_{\mathcal{D}}$. We are approaching in the same fashion as in the case of $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1 + \mathsf{UCQ}$ (Section 5.1). Although, FOL-rewritability of ontology satisfiability follows from the more general case, $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1$, we will need to adopt the notions of canonical model and of homomorphism between models in order to prove FOL-rewritability of query answering. We start with satisfiability.

Before that, we remind the reader on the conditions that a datatype lattice $\mathcal{D}_2$ has to satisfy in order to be of type 2:

(infinite) There exists no $T_1, \ldots, T_n$ in $\mathcal{D}_2$ for $n \geq 1$ s.t. $|\bigsqcap_{i=1}^{k} T_i| < \infty$.

(infinite-diff.) There exists no T_s and $T_1, \ldots, T_n$ in $\mathcal{D}_2$ for $n \geq 1$ s.t. $T_s \not\sqsubseteq T_i$ $(1 \leq i \leq n)$ and $|T_{sup} \setminus \bigsqcup_{i=1}^{k} T_i| < \infty$.

(open-domain) $\mathcal{D}_2$ doesn't impose a closed domain, i.e. exists interpretation $\mathcal{I}$ s.t.:

$$|\Delta_V^{\mathcal{I}} \setminus \bigcup_{i=1}^{n} VS_{T_i}| = \infty \text{ where } T_1, \ldots, T_n \text{ denote all datatypes defined in } \mathcal{D}_2.$$

Example 6 Notice that, T_is from (infinite-diff.) condition are not necessarily sub-types of T_i. For example:

$$T_k := \mathsf{xsd : integer}[<_0 \cup \geq_{10}], \quad T_i := \mathsf{xsd : integer}[\geq_0]$$

then $T_i \not\sqsubseteq T_k$, but $|T_i \setminus T_k| = 11$. $\bigtriangleup$

Before starting with formal definitions we would like to make a short comment over the datatype constructs in $\mathcal{D}_2$. New datatypes in $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ can be build using various constructors. The constructors include facet expression ($T_i := T_j[\varphi]$), union of two datatypes ($T_i := T_1 \sqcup T_2$), intersection of two datatypes ($T_i := T_1 \sqcap T_2$), difference of two datatypes ($T_i := T_1 \sqcup T_2$), and any finite combination of them.

However, only data ranges that includes only intersection (f.e. $T_{k+1} = T_1 \sqcap \ldots \sqcap T_k$), one can define without considering consequences. In other words, if $\{T_1, \ldots T_k\}$ is set that obey $\mathcal{D}_2$ conditions, then adding T_{k+1} in the set will not violate any of $\mathcal{D}_2$ conditions.

On the other side this doesn't hold for other constructors. For an example, if we define

$$\mathsf{myowl : negative} := \mathsf{xsd : integer} \setminus \mathsf{xsd : nonNegativeInteger}.$$

then such definition directly violate (sup-union) condition as

$$\mathsf{xsd : integer} \equiv \mathsf{myowl : negativeInteger} \sqcup \mathsf{xsd : nonNegativeInteger}.$$

Similar situation we can have with union:

$$\mathsf{myowl : myinteger} := \mathsf{myowl : negativeInteger} \sqcup \mathsf{xsd : nonNegativeInteger}$$

, and with facet expressions:

$$\mathsf{myowl : negativeInteger} := \mathsf{integer}[<_0], \quad \mathsf{myowl : nonNegativeInteger} := \mathsf{integer}[\geq_0].$$

6.1.1 Satisfiability

Every $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ ontology is also a $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ ontology. Hence, all properties relating satisfiability that holds for $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ ontologies, will hold for $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ ontologies as well. The most important consequence is that satisfiability of $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ ontologies is FOL-rewritable.

On the hand, we will show that canonical model of $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ ontologies will not return certain answers of over the $\mathsf{UCQ}_{\mathcal{D}}s$. Thus, we need to adopt the canonical model for $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ ontologies. But then, we have to show again that properties relating satisfiability holds for the $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ canonical model.

Luckily, the change on the $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ canonical model is small, and we can easily adopt all satisfiability properties.

Normalization

Given a *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ ontology $\mathcal{O}$, first step will be normalization. Normalization is done identically as the *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ normalization (see 5.1.1).

Canonical model

The main consequence of the (infinite-diff.) condition is infinite number of values of a datatype that do not belong to any proper sub-type or a type . This allows us to that in $can(\mathcal{O})$ model while considering a rule $B \sqsubseteq \exists U$, $can(\mathcal{O})$ model can always take a values of from $\mathsf{Rng}^{\mathcal{T}}_{max}(U)$ that do not belongs to any proper sub-type of $\mathsf{Rng}^{\mathcal{T}}_{max}(U)$.

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ ontology and let $\mathcal{D}_2$ be a datatype lattice assigned to $\mathcal{O}$. For each datatype T_i defined in $\mathcal{D}$ we define an infinite set of datatype constants $Rest_{T_i} = \{{d_1}^{T_i}, \ldots\}$, where for each ${d_j}^{T_i} \in Rest_{T_i}$ holds:

$$\text{if } ({d_j}^{T_i})^{\mathcal{D}_2} \in (T_k)^{\mathcal{D}_2} \text{ for some datatype } T_k \neq T_i \text{ in } \mathcal{D}_2 \text{ then } T_i \sqsubseteq T_k$$

We point out that existence of $Rest_{T_i}$ is a consequence of conditions (infinite) and (infinite-diff.).

Once we have defined $Rest_{T_i}$, the construction of a canonical model for $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ precede in the same fashion as in the case of *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ ontologies. Firstly a chase is constructed starting from the ABox and expanding iteratively according the positive rules in $\mathcal{T}$(see Definition 18). Finally, a model $can(\mathcal{O}) = \langle \Delta^{can(\mathcal{O})}, \cdot^{can(\mathcal{O})} \rangle$ is defined, that match ground facts from the chase and adopts semantics of the assigned datatype lattice.

In the following, we adopt the properties that hold for *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ ontologies, for the case of *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_2$.

Lemma 13. *(i)Let $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ be a DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ *ontology and let $\mathcal{T}_p$ be the set of positive inclusion assertions in $\mathcal{T}$. Then, $can(\mathcal{O}) \models (\mathcal{T}_p, \mathcal{A})$.*

(ii)If two conditions hold, for every $U(o, d) \in \mathcal{A}$ we have $d^{\mathcal{D}} \in \mathsf{Rng}^{\mathcal{T}}_{max}(U)$ and when $\mathcal{O} \models \exists U$ then $\mathsf{Rng}^{\mathcal{T}}_{max}(U) \neq \bot_{\mathcal{D}}$, then $can(\mathcal{O}) \models (\mathcal{T}_p \cup \mathcal{T}_d, \mathcal{A})$, where $\mathcal{T}_d$ is the set of all range constraints ($\mathsf{Rng}(U) \sqsubseteq T_i$) in $\mathcal{T}$.

Lemma 14. *For every DL-Lite*$_{core}^{(\mathcal{HF})}+\mathcal{D}_2$ *ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ it holds: $can(\mathcal{O}) \models T_f$ iff $DB(\mathcal{A}) \models T_f$.*

Negative assertions closure

So far we have seen that $can(\mathcal{O})$ model satisfies all PIs in $\mathcal{T}$, i.e. $can(\mathcal{O}) \models \mathcal{T}_p$ (Lemma 13(i)). Also it satisfies all data ranges in $\mathcal{T}$, i.e. $can(\mathcal{O}) \models \mathcal{T}_d$, whenever $\mathcal{O}$ is not inconsistent with having an attribute U for which $\mathcal{O} \models \exists U$ and $\mathsf{Rng}^{\mathcal{T}}_{max}(U) = \bot_{\mathcal{D}}$(Lemma 13(ii)). Moreover, $can(\mathcal{O})$ doesn't violate any functional constraint in $\mathcal{T}$ if some functional constraint has not been violated already by the ABox (Lemma 14)

Considering stated, the remaining step is to create a method that will check when $can(\mathcal{O})$ violate NIs in $\mathcal{T}$. Based on the interaction between negative and positive inclusions in a *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ TBox $\mathcal{T}$, we construct the NIs closure $cln(\mathcal{T})$, in the same way as for *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$. The canonical model of *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ is a special case of the canonical model in *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$, so the following properties holds directly:

Lemma 15. *For a DL-Lite$_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ TBox $\mathcal{T}$ and an arbitrary NI α it holds: if $cln(\mathcal{T}) \models \alpha$ then $\mathcal{T} \models \alpha$.*

Lemma 16. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ ontology. Then, $can(\mathcal{O}) \models \mathcal{O}$ iff $DB(\mathcal{A}) \models cln(\mathcal{T})$.*

Lemma 17. *For every DL-Lite$_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ ontology $\mathcal{O}$ it holds: $can(\mathcal{O}) \models \mathcal{O}$ iff $\mathcal{O}$ is satisfiable.*

Finally from Lemmas 16. and 17. we conclude:

Lemma 18. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ ontology. Then $DB(\mathcal{A}) \models cln(\mathcal{T})$ iff $\mathcal{O}$ is satisfiable.*

Lemma 18 gives us direct guide how to construct a boolean FOL query for which $\varphi(\mathcal{T})^{db} = \bot$ iff $\mathcal{O}$ is satisfiable.

$$\varphi(\mathcal{T}) = \bigvee_{\alpha \in cln(\mathcal{T})} \varphi(\alpha) \tag{6.1}$$

The transformation φ maps every NI α to its negation expressed as a FOL sentence (see formula 5.1).

And we conclude,

Theorem 7. *Satisfiability problem is FOL-rewritable in DL-Lite$_{core}^{(\mathcal{HF})} + \mathcal{D}_2$.*

6.1.2 Certain answers

We remind the reader, that $\mathcal{V}(A)$ denotes a set of all predicates from our vocabulary $\mathcal{V}$, i.e. all concepts, roles and attributes. With $\mathcal{V}(\mathcal{D})$ we denote the set of all datatype names in $\mathcal{V}$.

The next lemma is very similar to its counter-part in *DL-Lite$_{core}^{(\mathcal{HF})}$*$+\mathcal{D}_1$ logic. Nevertheless, in this case we define a homomorphism over a bigger set of atoms, $\mathcal{V}(A) \cup \mathcal{V}(\mathcal{D})$. The subtle difference is of a crucial importance for the ensuring that a composition of two homomorphisms is again a homomorphism, need in the lemma after this.

Lemma 19. *Let $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ be a satisfiable DL-Lite$_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ ontology $\mathcal{D}_2$ as assigned lattice, and let $\mathcal{M} = (\Delta^{\mathcal{M}}, \cdot^{\mathcal{M}})$ be a model of $\mathcal{O}$. Then, there is a homomorphism from $can(\mathcal{O})$ to $\mathcal{M}$ over the set of predicates from $\mathcal{V}(A) \cup \mathcal{V}(\mathcal{D})$.*

Proof. We are constructing a homomorphism μ form $can(\mathcal{O})$ to $\mathcal{M}$, by induction over the construction of $can(\mathcal{O})$. In addition, for each step i in the induction, we show that:

(hom.) For each objects $o, o' \in \Delta_O^{can_i(\mathcal{O})}$ and value $v \in \Delta_V^{can_i(\mathcal{O})}$ holds:

for any atomic concept A from $\mathcal{O}$, if $o \in A^{can_i(\mathcal{O})}$ then $\mu(o) \in A^{\mathcal{M}}$,

for any atomic role P from $\mathcal{O}$, if $(o, o') \in P^{can_i(\mathcal{O})}$ then $(\mu(o), \mu(o')) \in P^{\mathcal{M}}$,

for any atomic attribute U from $\mathcal{O}$, if $(o, v) \in U^{can_i(\mathcal{O})}$ then $(\mu(o), \mu(v)) \in U^{\mathcal{M}}$,

for any datatype name T_i from $\mathcal{D}_2$, if $d^{can(\mathcal{O})} \in T_i$ then $(\mu(d))^{\mathcal{M}} \in T_i$. Notice that $d^{can(\mathcal{O})} = d^{\mathcal{D}}$ and $(\mu(d))^{\mathcal{M}} = (\mu(d))^{\mathcal{D}}$.

If the property *(hom.)* holds for each i, then it immediately establishes μ as a homomorphism form $can(\mathcal{O})$ to $\mathcal{M}$ over the set of predicates from$\mathcal{V}(A)\cup\mathcal{V}(\mathcal{T})$.

Base. For each object constant o that appears in $\mathcal{A}$ we set $\mu(o) = o^{\mathcal{M}}$. For each datatype constant d that appears in $\mathcal{A}$ we set $\mu(d) = d^{\mathcal{M}}(= d^{\mathcal{D}})$. Since $\mathcal{M} \models \mathcal{A}$, *(hom.)* property follows directly for $chase_0(\mathcal{O})$.

Induction step. Here we distinguish several cases based on the assertion β and the applied rule α in obtaining $can_{i+1}(\mathcal{O})$ from $can_i(\mathcal{O})$. For example, assume $\alpha = B \sqsubseteq \exists U$ and $\beta = B(o)$, where $B(o)$ denotes a membership assertion of the form $A(o)$, $R(o, o_1)$ or $U(o, d_1)$. Then $can_{i+1}(\mathcal{O}) = can_i(\mathcal{O})\cup U(o, (d_{new})^{\mathcal{D}})$, where d_{new} is a fresh datatype constant from $\mathsf{Rng}^{\mathcal{T}}_{max}(U)$. From the definition of $Rest_{T_i}$, d_{new} neither belongs to any proper subtype of $\mathsf{Rng}^{\mathcal{T}}_{max}(U)$ nor to a datatype with common intersection with $Rest_{T_i}$ (*).

According induction hypothesis, exists $o_m \in \Delta^{\mathcal{M}}$ s.t. $\mu(o) = o_m$ and $o_m \in A^{\mathcal{M}}$. Since $\mathcal{M} \models \alpha$ then must exists some datatype constant d' s.t. $(o_m, (d')^{\mathcal{M}}) \in U^{\mathcal{M}}$. We set $\mu(d_{new}) = (d')^{\mathcal{M}}$ and conclude $(\mu(o), \mu(d_{new})) \in U^{\mathcal{M}}$. Between attributes U is the only attribute that has changes semantics and we conclude that the property (hom.) holds for attributes in $can_{i+1}(\mathcal{O})$. Concepts and roles are the same as in $can_i(\mathcal{O})$ so the (hom.) property holds for them in $can_{i+1}(\mathcal{O})$ as well.

Now only remains to check (hom.) property on datatypes in $\mathcal{D}_2$. Let T_i be a datatype from $\mathcal{D}$. Taking into account (*) T_i can be only $\mathsf{Rng}^{\mathcal{T}}_{max}(U)$ or $\mathsf{Rng}^{\mathcal{T}}_{max}(U) \sqsubseteq T_i$. From the (soundess) of the Lemma 2, we have $\mathcal{O} \models \mathsf{Rng}(U) \sqsubseteq \mathsf{Rng}^{\mathcal{T}}_{max}(U)$. Since $\mathcal{M} \models \mathcal{O}$ then $(d')^{\mathcal{M}} \in \mathsf{Rng}^{\mathcal{T}}_{max}(U)$. To conclude, for any datatype T_i, and for every datatype constant in $can_{i+1}(\mathcal{O})$ if $d^{can_{i+1}(\mathcal{O})} \in T_i$ then $(\mu(d))^{\mathcal{M}} \in T_i$.

To complete μ formally over the $\Delta_{\mathcal{D}_2}$, for every datatype constant d that $d^{\mathcal{D}}$ doesn't appear in $can(\mathcal{O})$ construction we set $\mu(d^{\mathcal{D}}) = d^{\mathcal{D}}$.

Similarly, we can reason for $can_{i+1}(\mathcal{O})$ in other cases of α and β. □

A consequence of previous lemma we have an important property:

Proposition 3. *Let q be a* UCQ *over a satisfiable DL-Lite$^{(\mathcal{HF})}_{core} + \mathcal{D}_2$ ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ and let t be a tuple of constants that from $\mathcal{A}$. Then,*

$$t \in cert(q, \mathcal{O}) \quad \text{iff} \quad t^{can(\mathcal{O})} \in q^{can(\mathcal{O})}$$

Proof. ($\Rightarrow$). Follows from the definition of a certain answer.
($\Leftarrow$). Then if q is UCQ composed with CQ $\{q_1, \ldots, q_n\}$, there must be at least one $q_i(x) \leftarrow conj_i(x, y)$ s.t. $t^{can(\mathcal{O})} \in q_i^{can(\mathcal{O})}$. According Theorem 1. there is a homomorphism μ from $conj_i(t, y)$ to $can(\mathcal{O})$.

Now let $\mathcal{M}$ be an arbitrary model of $\mathcal{O}$. According Lemma 19. there exists a homomorphism η from $can(\mathcal{O})$ to $\mathcal{M}$. Since, homomorphisms are closed under the composition we get that $\eta \circ \mu$ is a homomorphism from $conj_i(t, y)$ to $\mathcal{M}$. Again using Theorem 1. we conclude that $t^{\mathcal{M}} \in q^{\mathcal{M}}$. Since, $\mathcal{M}$ is chosen arbitrary it follows that $t \in cert(q, \mathcal{O})$. □

Comment. The lemma 19 has significant importance for the previous proposition. Notice that we have already showed in lemma 9 there exists a homomorphism between a $can(\mathcal{O})$ and an arbitrary model $\mathcal{M}$. However, the homomorphism in 9 was over the set of atoms from $\mathcal{P}(A)$ only. Because DL-Lite$^{(\mathcal{HF})}_{core} + \mathcal{D}_1$

has weaker datatype conditions, the homomorphism doesn't hold for $\mathcal{P}(A) + \mathcal{P}(\mathcal{D})$.

Further more, $\mathsf{CQ}_\mathcal{D}$ contains datatypes atoms while lemma 9 address CQs only. Consequently composition of a homomorphism between models over $\mathcal{P}(A)$ and a homomorphism between a model and a UCQ is not always a homomorphism.

Example 7 continues on the example 5 Assume the ontology in example 5 is satisfiable and we have two models $can(\mathcal{O})$ and $\mathcal{M}$ which are partially described in example 5. Then consider a $\mathsf{CQ}_\mathcal{D}$ query:

$$q(x) : -\exists y.Student(x), \mathsf{myowl} : \mathsf{hasAge}(x, y), \mathsf{myowl} : \mathsf{oldStudentAge}(y)$$

Then $(\mathsf{John})^{can(\mathcal{O})} \in q^{can(\mathcal{O})}$. On the contrary, $\mathsf{John} \notin cert(\mathcal{O}, q)$ because $(\mathsf{John})^{\mathcal{M}} \notin q^{\mathcal{M}}$.

The example shows that although $can(\mathcal{O})$ and $\mathcal{M}$ establish homomorphism from Lemma 9. this is not sufficient for certain answers on $\mathsf{CQ}_\mathcal{D}$ queries. Essentially, this a reason why we need (infinite-diff.) condition on datatypes in $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_2 + \mathsf{UCQ}_\mathcal{D}$. $\bigtriangleup$

The next lemma states that certain answers of a UCQ query can be obtained from the certain answers of its CQs.

Lemma 20. *Let $\mathcal{O}$ be an $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ ontology and q a $\mathsf{UCQ}_\mathcal{D}$ over it. We claim:*

$$cert(q, \mathcal{O}) = \bigcup_{q_i \in q} cert(q_i, \mathcal{O})$$

Proof. ($\subseteq$:) By definition.

($\supseteq$:) If the ontology is unsatisfiable then it follows trivially. Otherwise, let t be a tuple from $cert(q, \mathcal{O})$. Then from Proposition 2. $t^{can(\mathcal{O})} \in q^{can(\mathcal{O})}$. It follows that exists some $\mathsf{CQ}_\mathcal{D}$ q_i in q, $t^{can(\mathcal{O})} \in q_i^{can(\mathcal{O})}$. Then again according Proposition 2. $t \in cert(q_i, \mathcal{O})$. □

Perfect reformulation

Perfect reformulation in $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_2 + \mathsf{UCQ}_\mathcal{D}$ we will calculate using the same PerfectRef algorithm as for the $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_1 + \mathsf{UCQ}$. Nevertheless, we will need to slightly adopt rewriting rules. But firstly, we describe how to rewrite a UCQ query in a equivalent UCQ query that contains less inconsistencies (f.e. shared variable between concept and datatype).

So far, a bound variable was defined as a variable that is either distinguish or shared between atoms in a query. An unbounded variable was the opposite one, non-shared and non-distinguishable variable. We used the symbol "_" do denote them.

(Safety) condition of $\mathsf{CQ}_\mathcal{D}$ impose that each distinguishable x that appears in some datatype atom $T_j(x)$ has also to appear in some atom that is not datatype. Without lose of generality, we can assume that for every $T_j(x_2)$, where x distinguishable or shared, appears only on the second position in some attribute $U(x_1, x_2)$. Cases in which x_2 appears on other places, such $\mathsf{CQ}_\mathcal{D}$ are semantically false and we can discard them. Query without atom is logically equal to $\bot$ (*false*).

Additionally we assume that we have no datatype atoms that have no shared variable with some attribute. There are tow possible counter-cases. The atom has the form $T_i(d_j)$ and then we simply check if $d^{\mathcal{D}} \in T_i$ then discard $T_i(d_j)$ from a $\mathsf{CQ}_{\mathcal{D}}$, or if not then discard the $\mathsf{CQ}_{\mathcal{D}}$. Secondly, if we have $T_1(y_1), \ldots, T_k(y_1)$ in a $\mathsf{CQ}_{\mathcal{D}}$, then either $\bigsqcap_{1 \leq i \leq k} T_i \equiv \bot_{\mathcal{D}}$ so we discard the $\mathsf{CQ}_{\mathcal{D}}$, or if $\bigsqcap_{1 \leq i \leq k} T_i \not\equiv \bot_{\mathcal{D}}$ then we discard $T_1(y_1), \ldots, T_k(y_1)$ for the $\mathsf{CQ}_{\mathcal{D}}$.

We summarize the previous analysis in following:

- $(\mathsf{Safety})^*$ Each datatype atom in a $\mathsf{CQ}_{\mathcal{D}}$ appears in the form $T_i(x)$ where x is a variable that appears at only at the second place of an attribute atom ($U(t_1, x)$) and exists at least one such attribute.

We say that a variable is bounded if it is distinguishable or it is shared between concept, role and attribute atoms (not datatype atoms). For example,

$$q_1() : -\exists x, y.A(x), U(x, y), T_i(y) \qquad q_2() : -\exists x, y.A(x), U(x, y), T_i(y), U_2(x, y)$$

y is not shared variable in q_1, while it is a shared variable in q_2.

Unbound variables are not bounded. We use the symbols "$\_{}_i$" to represent them, where i is a natural number. Each unbounded variable will get uniquely numbered symbol, except variables from datatype atoms, that take the symbol of the attributes that associate with them. For example, in a query :

$$q_3() : -\exists x, y, z.A(x), U(x, y), T_i(y), B(z)$$

a renaming with "$\_{}_i$" symbols will be

$$q_3() : -\exists x.A(x), U(x, \_{}_1), T_i(\_{}_1), B(\_{}_2)$$

In the rest, we will use $\_{}_i$ to denote an arbitrary non-bounded variable.

The following list defines whether an atom in $\mathsf{CQ}_{\mathcal{D}}$ is applicable to a PI α from $\mathcal{T}$. The definition is slightly changed comparing to $DL\text{-}Lite_{core}^{(\mathcal{HF})}+\mathcal{D}_1+\mathsf{UCQ}$ counterpart. Formally:

- A PI α is applicable to an atom $A(x)$, if α has A in its right-hand side.
- A PI α is applicable to an atom $P(x_1, x_2)$, if one of the following conditions holds:
  - $x_2 = \_{}_i$ and the right-hand side of α is $\exists P$ or
  - $x_1 = \_{}_i$ and the right-hand side of α is $\exists P^-$ or
  - α is a role inclusion assertion and its right-hand side is either P or P^-.
- A PI α is applicable to an atom $U(x_1, x_2)$, if one of the following conditions holds:
  - $x_2 = \_{}_i$ and the right-hand side of α is $\exists U$ and there exists no datatype atom in the $\mathsf{CQ}_{\mathcal{D}}$ s.t. $T_j(\_{}_i)$, or
  - α is a attribute inclusion assertion and its right-hand side is U.

In the algorithm figure2 we provide the algorithm $\mathsf{PerfectRef}_{\mathcal{D}}$, which reformulates a $\mathsf{UCQ}_{\mathcal{D}}$ (considered as a set of $\mathsf{CQ}_{\mathcal{D}}s$) by taking into account the PIs of a TBox $\mathcal{T}$. $\mathsf{PerfectRef}_{\mathcal{D}}$ algorithm is based on three functions:

- $\mathsf{rewrite}(q, g, \alpha)$. We denote with $B(x)$ any of an atom of the form $A(x)$, $R(x, \__i)$, $U(x, \__i)$, where $R(x, \__i) = P(x, \__i)$ when $R = P$ and $R(x, \__i) = P(\__i, x)$ when $R = P^-$. Also, we denote with B any of the *DL-Lite* concept expression of the form A, $\exists R$ or $\exists U$. Additionally, for a given B we define $B(x)$ as $A(x)$ when $A = B$, $R(x, \__i)$ $B = \exists R$, $U(x, \__i)$ when $B = \exists U$. Finally, we define rewriting $(\mathsf{rw})(q, \alpha)$, for a given atom g and a given PI α according the table:

g	α	$(\mathsf{rw})(q, \alpha)$
$B_1(x)$	$B_2 \sqsubseteq B_1$	$B_2(x)$
$R_1(x_1, x_2)$	$R_2 \sqsubseteq R_1$	$R_2(x_1, x_2)$
$U_1(x_1, x_2)$	$U_2 \sqsubseteq U_1$	$U_2(x_1, x_2)$

  Then, $\mathsf{rewrite}(q, g, \alpha)$ is a CQ obtained from q by replacing the atom q with a new atom $(\mathsf{rw})(q, \alpha)$.

- $\mathsf{reduce}(q, g_1, g_2)$. Method $\mathsf{reduce}(q, g_1, g_2)$ creates a new $\mathsf{CQ}_\mathcal{D}$ form q by applying two operations on q. Firstly, calculates the most general unifier (mgu) ν of atoms g_1 and q_2, and applies it over q, $q' = \nu(q)$. This is done with remark, that unifying g_1 and g_2, each occurrence of the symbol $\__i$ has to be considered as a different unbound variable.

  After unification, new variables can become non-distinguishable non-shared (unbound) variables. Therefore, the second operation is replacement of the unbound variables with $\__i$ symbol (say from q' we obtain q'' by this operation). Notice that, possible new rules will be applicable in q'' in the step (rw), that were not before in q.

  Finally, a CQ q" = $\mathsf{reduce}(q, g_1, g_2)$ is returned as a result.

- $\mathsf{deleteRng}(U(x, \__i), q)$. Method Rng ( ) creates a new $\mathsf{CQ}_\mathcal{D}$ from q by deleting all datatype atoms with variable $\__i$, i.e. datatype atom of the form $T_j(\__i)$. We elaborate the method correctness:

  The condition for the (range) step is $\mathsf{Rng}^\mathcal{T}_{max}(U) \sqsubseteq \mathsf{Rng}^q(U(x, \__i))$. Considering that $\mathsf{Rng}(U) \sqsubseteq \mathsf{Rng}^\mathcal{T}_{max}(U)$, the datatype restriction from $\mathsf{Rng}^q(U(x, \__i))$ have no effect on U, and from the logic point of view we can do no harm by deleting them. Moreover, if we delete them potentially new rules of the form $B \sqsubseteq \exists U$ will become applicable.

  On the other hand, if $\mathsf{Rng}(U) \not\sqsubseteq \mathsf{Rng}^\mathcal{T}_{max}(U)$ then datatype restrictions from $\mathsf{Rng}^q(U(x, \__i))$ remains. This is also logically correct, considering the construction of $can(\mathcal{O})$. Namely, $can(\mathcal{O})$ model on the rules like $B \sqsubseteq \exists U$, will add new constants from $\mathsf{Rng}^\mathcal{T}_{max}(U)$ that are not in the intersection $\mathsf{Rng}^q(U(x, \__i))$. Then if $DB(\mathcal{A})$ doesn't contains facts that satisfies $U(x, \__i) \sqcap \mathsf{Rng}^q(U(x, \__i))$ then neither $can(\mathcal{O})$ will satisfy it.

The algorithm that computes $\mathsf{UCQ}_\mathcal{D}$ $\mathsf{PerfectRef}_\mathcal{D}(q, \mathcal{T})$ for a given $\mathsf{UCQ}_\mathcal{D}$ q and TBox $\mathcal{T}$ is presented on Algorithm 2.

All three functions in the algorithm $\mathsf{PerfectRef}_\mathcal{D}$ produces $\mathsf{CQ}_\mathcal{D}$ that is equal or shorter than original one. Further, all $\mathsf{CQ}_\mathcal{D}$ created in $\mathsf{PerfectRef}_\mathcal{D}$ are build using the variables, constants and predicate names from $\mathcal{T}$ and q. To conclude,

Algorithm 2 The algorithm that computes PerfectRef of a $\mathsf{UCQ}_{\mathcal{D}}$ q given a $\mathit{DL\text{-}Lite}_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ TBox $\mathcal{T}$

Input: $\mathsf{UCQ}_{\mathcal{D}}$ q, $\mathit{DL\text{-}Lite}_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ TBox $\mathcal{T}$
Output: $\mathsf{UCQ}_{\mathcal{D}}$ q_{pr}

1: **procedure** PerfectRef$(q, \mathcal{T})$
2: $q_{pr} \leftarrow q$
3: **repeat**
4: $q'_{pr} \leftarrow q_{pr}$
5: **for all** $\mathsf{CQ}_{\mathcal{D}}$ q in q_{pr} **do**
6: **for all** atoms g in q **do** ▷ step (rw)
7: **for all** PI α in $\mathcal{T}$ **do**
8: **if** α is applicable to g **then**
9: $q'_{pr} \leftarrow q'_{pr} \cup \mathsf{rewrite}(q, g, \alpha)$;
10: **end if**
11: **end for**
12: **end for**
13: **for all** pair of atoms g_1, g_2 in q **do** ▷ step (red)
14: **if** g_1 and g_2 unify **then**
15: $q'_{pr} \leftarrow q'_{pr} \cup \mathsf{reduce}(q, g_1, g_2)$;
16: **end if**
17: **end for**
18: **for all** $U(x, \__i) \in q$ **do** ▷ step (range)
19: **if** $\mathsf{Rng}^{\mathcal{T}}_{max}(U) \sqsubseteq \mathsf{Rng}^{q}(U(x, \__i))$ **then**
20: $q'_{pr} \leftarrow q'_{pr} \cup \mathsf{deleteRng}(q, \mathsf{Rng}^{q}(U(x, \__i)))$;
21: **end if**
22: **end for**
23: **end for**
24: **until** $q_{pr} = q'_{pr}$
25: **return** q_{pr};
26: **end procedure**

the size of each CQ in q_{pr} less or equal $|q|$ and there are at most $|\mathcal{T}| \times |q|^2$ of them in q_{pr}. In other words:

$$|q_{pr}| \leq (|\mathcal{T}| \times |q|^2)^{|q|}. \tag{6.2}$$

Formula 6.2 quarantines the termination of $\mathsf{PerfectRef}_{\mathcal{D}}$ algorithm.

The last step in proving that $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_2 + \mathsf{UCQ}_{\mathcal{D}}$ is FOL-rewritable is the next theorem.

Theorem 8. *Given a satisfiable $DL\text{-}Lite_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ and a $\mathsf{UCQ}_{\mathcal{D}}$ query q over it, we claim that holds:*

$$(q_{pr})^{DB(\mathcal{A})} = cert(q, \mathcal{O})$$

, where $q_{pr} = \mathsf{PerfectRef}_{\mathcal{D}}(q, \mathcal{T})$.

Proof. Firstly we define a notion of witness. Given an ontology $\mathcal{O}$, t tuple of constants from $\mathcal{O}$, and a $\mathsf{CQ}_{\mathcal{D}}$ $q(x)$ over $\mathcal{O}$, we say that a set of membership assertion Θ, is a witness for t wrt q, if there exists substation σ ($\{x \mapsto t\} \in \sigma$) over atoms in q s.t. $\sigma(q(x)) = \Theta$ (q is considered set-wise).

Considering Lemma 20 we have that $(q_{pr})^{DB(\mathcal{A})} = \bigcup_{\bar{q} \in q_{pr}} \bar{q}^{DB(\mathcal{A})}$. Moreover, considering Proposition 3, it is sufficient to show:

$$\bigcup_{\bar{q} \in q_{pr}} \bar{q}^{DB(\mathcal{A})} = q^{can(\mathcal{O})}$$

($\subseteq$:) We have to show that for an arbitrary $\bar{q} \in q_{pr}$, $\bar{q}^{DB(\mathcal{A})} \subseteq q^{can(\mathcal{O})}$.

We will show it by induction over the number of steps (rw), (red) and (range) used for generating $\bar{q}$.

Induction base. No reformulation applied, so $\bar{q} \in q$. Then $\bar{q}^{DB(\mathcal{A})} \subseteq \bar{q}^{can(\mathcal{O})} \subseteq q^{can(\mathcal{O})}$.

Induction step. Induction hypothesis is that for all $\bar{q} \in q_{pr}$ s.t $\bar{q}$ is generated using steps (rw), (red) and (range) in less then n times , $\bar{q}^{DB(\mathcal{A})} \subseteq \bar{q}^{can(\mathcal{O})} \subseteq q^{can(\mathcal{O})}$. It is sufficient to show that for each tuple t from $\mathcal{O}$ and for each $\bar{q_2}$ generated in $(n+1)$ steps holds: if $t^{can(\mathcal{O})} \in \bar{q_2}^{can(\mathcal{O})}$ then $t^{can(\mathcal{O})} \in q^{can(\mathcal{O})}$.

Firstly, we assume that $\bar{q_2}$ is generated in $(n+1)$ steps, where the last step was (rw) on the rule $\beta : A \sqsubseteq \exists U$, i.e. $\bar{q_2} = \mathsf{rewrite}(\bar{q_1}, \beta, U)$. By assumption, $\bar{q_1}^{can(\mathcal{O})} \subseteq q^{can(\mathcal{O})}$. If for a tuple t from $\mathcal{O}$, we have a witness Θ wrt $\bar{q_2}$ in $can(\mathcal{O})$, then there are two possibilities. Firstly, that Θ is already a witness for t wrt $\bar{q_1}$. If not, then must be a ground atom $A(o_i)$ that was applied on β in $can(\mathcal{O})$ to obtain witness Θ. Then we enrich Θ, with the $U(o_i, d)$, obtained by application of $A(o_i)$ over β, to create a witness $\Theta_1 \cup can(\mathcal{O})$ for t wrt $\bar{q_1}$. Notice that the fact $U(o_i, d)$ doesn't violate any datatype atom in $\bar{q_1}$. Finally, in both cases, $t^{can(\mathcal{O})} \in \bar{q_1}^{can(\mathcal{O})}$. Considering the hypothesis it holds $t^{can(\mathcal{O})} \in \bar{q}^{can(\mathcal{O})}$.

Similarly, we can prove other cases for (rw) step.

If $\bar{q_2} = \mathsf{reduce}(\bar{q_1}, g_1, g_2)$, then every witness for t wrt $\bar{q_2}$ is also a witness wrt $\bar{q_1}$.

If $\bar{q_2} = \mathsf{deleteRng}(U(x, \__i), \bar{q_1})$, then every witness for t wrt $\bar{q_2}$ is also a witness wrt $\bar{q_1}$. Since, the condition of $\mathsf{deleteRng}(,)$ holds, then for a witness atom $U(o, d)$ that corresponds to $U(x, \__i)$, it must hold $d \in \mathsf{Rng}^q(U(x, \__i))$.

($\supseteq$:) Assume $t^{can(\mathcal{O})} \in q^{can(\mathcal{O})}$ and that Θ is a witness. Then exists a $\mathsf{CQ}_{\mathcal{D}}$ $q' \in q$ for which Θ is the witness. Analogically to the theorem 6.1.2, we are constructing $\bar{q}$ from q_{pr} s.t. $t \in \bar{q}^{DB(\mathcal{A})}$.

The canonical model $can(\mathcal{O})$ can be seen as a forest, where nodes are membership assertions in $can(\mathcal{O})$ and edges correspond to the rules that generate new membership assertions from $\mathcal{A}$. In particular, all facts from $\mathcal{A}$ we consider as roots. Out of this forest, potentially infinite, we will consider a minimal as possible (final) sub-forest $\mathcal{F}$, where leaves are the assertions in Θ, roots are the roots of the assertions in Θ, and edges are one that establish paths between the leaves and the roots, s.t. Θ is contained in the set of nodes from $\mathcal{F}$. Notice that each membership assertion can appear only once in $\mathcal{F}$.

The idea is to construct $\bar{q}$ starting from q' and following the rules (inverted) that generate Θ, from the leaves towards roots. Assume that $\mathcal{F}$ has n edges. We define a chain of sub-forests $\mathcal{F}_0 = \mathcal{F}, \mathcal{F}_1, \ldots, \mathcal{F}_n$ of $\mathcal{F}$, where $\mathcal{F}_{i+1}$ is obtained from $\mathcal{F}_i$ by deleting one leaf. Additionally we define the list of queries $\bar{q}_0, \bar{q}_1, \ldots, \bar{q}_n$, where $\bar{q}_{i+1}$ is obtained from $\bar{q}_i$, according the rule β_i that corresponds to the deleted edge in $\mathcal{F}_i$ in obtaining $\mathcal{F}_{i+1}$. Additionally, following the witness substitution for t from $\mathcal{F}_{i+1}$ wrt $\bar{q}_{i+1}$, we apply Reduction step (see bellow) on each $\bar{q}_{i+1}$. This is done, to ensure correct (rw) application for $\bar{q}_{i+1}$ in the next step.

Two criteria we have to show: (i) the rule β_i is applicable to corresponding atom in q_i, (ii)(induction hypothesis) and for each i there exists a witness in $\mathcal{F}_i$ for t wrt $\bar{q}_i$ s.t. no two atoms in $\bar{q}_i$ are assigned by the witness substitution to the same membership assertion from the witness (injection between atoms in $\bar{q}_i$ and the elements of a witness). As a consequence (i) we have that $\bar{q}_n$ is generated from q applying (rw), (red) and (range) rules and that (ii) there is a witness in $\mathcal{F}_n \subseteq DB(\mathcal{A})$ for t wrt $\bar{q}_n$, i.e. $t \in \bar{q}_n^{DB(\mathcal{A})}$.

Reduction step on $\bar{q}_i$. Each witness Θ_i determines a substitution σ, where $\sigma(\bar{q}_i(x)) = \Theta_i$ and $\{x \mapsto t\} \in \sigma$. Following this, we will apply (red) step to all pairs of atoms in $\bar{q}_i$ that become identical after applying substitution σ. After this atoms in $\bar{q}_i$ and assertions in Θ relates 1 to 1. Additionally, in $\bar{q}_i$ we applying (range) rule as much as possible. Then for each attribute of the form $U(x, \__i)$ in $\bar{q}_i$, if it has assigned datatype atoms then $\mathsf{Rng}^{\mathcal{T}}_{max}(U) \not\sqsubseteq \mathsf{Rng}^{q}(U(x, \__i))$.

Induction base. Let denote with $\bar{q}_0$ a query obtained from q after applying Reduction step. So, no two atoms in $\bar{q}_0$ relates to the same assertion in $\mathcal{F}_0$(injection). Nevertheless, $\sigma(q') = \sigma(\bar{q}_0)$(set-wise) and $\Theta \subseteq \mathcal{F}_0$ is a witness of t wrt $\bar{q}_0$.

Induction step. Assume that that exists a witness in $\mathcal{F}_i$ for t wrt $\bar{q}_i$. Now let $\mathcal{F}_{i+1} = \mathcal{F}_i \setminus \gamma$ where the edge label is rule β_i and γ_i is deleted membership assertion. Cases in β_i deal only with concepts and role are similar to the cases in . So we are focus on the other remaining case, $\beta_i = U_1 \sqsubseteq U_2$ or $\beta_i = B_1 \sqsubseteq \exists U_2$.

Firstly assume, $\beta_i = U_1 \sqsubseteq U_2$. Then $\bar{q}_{i+1} = \mathsf{rewrite}(\bar{q}_i, \beta_i, U_1)$ and the rule $\bar{q}_i$ is applicable. By assumption, $\mathcal{F}_i$ contains a witness for t wrt $\bar{q}_{i+1}$. Wlg assume, $U_2(x_1, x_2)$ was replaced with $U_1(x_1, x_2)$, and $U_1(o, d)$ is a fact from the witness that is assigned to $U_1(x_1, x_2)$. Consequently $U_1(o, d)$ is the new leaf in $\mathcal{F}_{i+1}$, $\mathcal{F}_i$ contains a wintess for $\bar{q}_{i+1}$. Considering that $\mathsf{Rng}^{\mathcal{T}}_{max}(U_1) \sqsubseteq \mathsf{Rng}^{\mathcal{T}}_{max}(U_2)$, $U_1(o, d)$ will not violate any of datatype restriction from $\mathsf{Rng}^{q}(U_1(x_1, x_2))$, because $\mathsf{Rng}^{q}(U_1(x_1, x_2)) = \mathsf{Rng}^{q}(U_2(x_1, x_2))$ and they are not violated by U_2 according assumption.

Secondly assume, $\beta_i = B \sqsubseteq \exists U$ and $\gamma_i = U(o, d_{new})$

As $\gamma_i = U(o, d_{new})$ is a leaf of $\mathcal{F}_i$ then d_{new} doesn't appear in any other membership assertion from $\mathcal{F}_i$. This means that a variable y in an atom $U(z, y) \in \bar{q}_i$

that corresponds to $U(o, o_{new})$ is not distinguishable one. Additionally y is not shared one, for the two reasons. Firstly there is no another atom $U(z_2, y) \in \bar{q}_i$ that corresponds to $U(o, d_{new})$ as Reduction step is applied on $\bar{q}_i$. Secondly, y can not appear in some other atom, for example $U(x_2, y) \in \bar{q}_i$, because the witness defines 1-1 (injection) relation between atoms in q_i and nodes in $\mathcal{F}_i$. So after application of Reduction step, $U(z, y)$ must be rewritten in $U(z, \__i) \in \bar{q}_i$ that has no assigned datatype atoms of the form $T_j(\__i)$. Then the rewriting using β to obtain $\bar{q}_{i+1}$ is correct.

□

From everything that has been shown in this chapter we conclude

Theorem 9. *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_2 + \mathsf{UCQ}_{\mathcal{D}}$ *is FOL-rewritable.*

Finally we summarize the complexity measure of *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_2 + \mathsf{UCQ}_{\mathcal{D}}$. The reasons given are the same as for the *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_2 + \mathsf{UCQ}_{\mathcal{D}}$, so please refer for details.

- $\mathsf{PerfectRef}(q, \mathcal{T}) \in \text{PTIME}(\mathcal{T})$.
  The meaning is that $\mathsf{PerfectRef}(q, \mathcal{T})$ is a polynomial algorithm in the size of $\mathcal{T}$. This is a consequence of the formula 5.2.
- (Data complexity) Answering $\mathsf{UCQ}_{\mathcal{D}}s$ in *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ is in PTIME in the size of the ontology.
- (Schema complexity) Answering $\mathsf{UCQ}_{\mathcal{D}}s$ in *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1$ is in AC^0 in the size of the ABox.
- (Combined complexity) Answering $\mathsf{UCQ}_{\mathcal{D}}s$ in *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_2$ is NP − *complete* in combined complexity.

6.2 Relaxing datatype conditions

In this section, we are proving that datatype conditions over $\mathcal{D}_2$ are necessary for *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_2 + \mathsf{UCQ}_{\mathcal{D}}$ to be FOL-rewritable. The proves are based on the reduction of the coNP-hard problems to the query answering in *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D} + \mathsf{UCQ}_{\mathcal{D}}$, where $\mathcal{D}$ violates at least one condition from $\mathcal{D}_2$.

So far we have seen (section 5.2), that condition (infinite), is already necessary condition for the framework *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D}_1 + \mathsf{UCQ}$ in order to be FOL-rewritable. Consequently, it is a necessary condition for *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D} + \mathsf{UCQ}_{\mathcal{D}}$ as well. Then only remains to show that violating either (infinite-diff.) or (open-domain) in *DL-Lite*$_{core}^{(\mathcal{HF})} + \mathcal{D} + \mathsf{UCQ}_{\mathcal{D}}$, query answering problem becomes coNP-hard wrt data complexity.

Reductions are expressed using concept inclusions ($B_1 \sqsubseteq B_2$), disjointness between attributes ($\mathsf{Disj}(U_1, U_2)$), and datatype range restriction ($\mathsf{Rng}(U) \sqsubseteq T_i$). Role hierarchies and functional restrictions are not used. It means, that already for *DL-Lite*$_{core}$ with attributes disjointness, conditions of type $\mathcal{D}_2$ are necessary for the efficient answering of $\mathsf{UCQ}_{\mathcal{D}}s$

Interesting observation can be made about the datatypes that violate the the conditions. For example, in the case of $|T_1 \setminus T_2| = k$, T_1 has to appear in the ontology but T_2 doesn't, in order prove coNP-hardness. The same situation

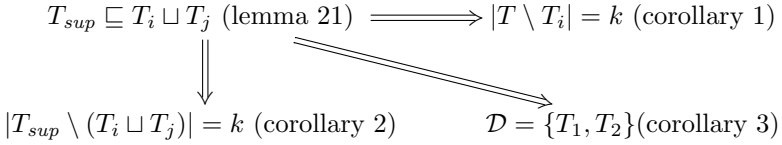
is for $|T_1 \sqsubseteq T_2 \sqcup T_3|$, where only T_1 appears in terminological part while T_2 and T_3 appear only in the $\mathsf{UCQ}_{\mathcal{D}}$. This advocates, that T_2 and T_3 need not to be considered as "regular" datatypes, but as a restrictions(facets) over T_1 to get coNP-hardness. This proves that our approach of introducing datatype atoms in UCQs instead of restrictions (facets) only, is very optimal in the sense, that we can't make better characterization of necessary conditions by bounding datatypes to terminological part and restrictions (with potentially more freedom) to queries.

Finally, we point out, that the condition (sup-union) is a special case of the (infinite-diff.) condition. The condition (sup-union) is also necessary, however it is not sufficient. In other words, there are frameworks that do not violate (sup-union), but that are still not FOL-rewritable.

We distinguish two cases. First, when (sup-union) and (infinite-diff.) are violated with two and three datatypes (n=2), and secondly when they are violated with at least four datatypes ($n \geq 3$). In the first case we reduce coNP-hard problem 2+2-CNF unsatisfiability ([22]), while in the second, we reduce coNP-hard problem which is k-COLOR complement. Formal definitions are provided within the proves.

Before presenting the formal proves, we elaborate the logical structure of reduction by cases ($\Longrightarrow$ means follows with adoption on the proof):

- 2+2-CNF unsatisfiability. We encode the 2+2-CNF unsatisfiability problem in the case of $T_{sup} \sqsubseteq T_i \sqcup T_j$ in lemma 21). Then adopting the proof we proved the corollaries 1, 2 and 3.



- k-color complement. We encode the k-color complement into the scenario of closed domain $\mathcal{D} = \{T_1, \ldots, T_k\}$ ($\Delta_V^{\mathcal{I}} = \bigcup_{1 \leq i \leq k} VS_{T_i}$). Adopting the lemma 22 proof we prove corollary 4.

$$\mathcal{D} = \{T_1, \ldots, T_k\}, k \geq 3 \text{ (lemma 22)}$$

$$\Downarrow$$

$$|T_{sup} \setminus \bigsqcup_{i=1}^{k} T_i| = k, k \geq 3 \text{(corollary 4)}$$

Lemma 21. *Let $\mathcal{O}$ be a DL-Lite$_{core}^{(\mathcal{HF})}$+$\mathcal{D}$ ontology defined over a datatype lattice $\mathcal{D}$ where exists three different datatypes T_{sup}, T_i, T_j s.t. $T_{sup} \not\sqsubseteq T_i$,$T_{sup} \not\sqsubseteq T_j$ and $T_{sup} \sqsubseteq T_i \sqcup T_j$. Then $\mathsf{UCQ}_{\mathcal{D}}$ answering over $\mathcal{O}$ is not FOL-rewritable.*

Proof. In order to show that a query answering in not FOL-rewritable in $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ we will reduce coNP-hard problem to query answering problem over $\mathcal{O}$, varying the size of the ABox only, i.e. making data complexity coNP-hard.

TBox remains fixed. As data complexity FOL-rewritable problems are in AC^0 this implies that query answering is not FOL-rewritable.

A coNP-complete problem that we are going to reduce is 2+2-CNF unsatisfiability. We adopted the proof idea from [10, p. 304].

A $2+2$-CNF formula on an alphabet $P = \{l_1, \ldots, l_m\}$ is a CNF formula $F = C_1 \wedge \ldots \wedge C_n$ in which each clause $C_i = L^i_{1+} \vee L^i_{2+} \vee \neg L^i_{1-} \vee \neg L^i_{2-}$ has exactly four literals, two positive ones, L^i_{1+} and L^i_{2+} , and two negative ones, $\neg L^i_{1-}$ and $\neg L^i_{2-}$, where the propositional letters $L^i_{1+}, L^i_{2+}, \neg L^i_{1-}$ and $\neg L^i_{2-}$ are elements of $\mathcal{P} \cup \{true, false\}$.

Given a $2+2$-CNF formula F as above, we associate with it an ontology $\mathcal{O}_F = \langle \mathcal{T}, \mathcal{A}_F \rangle$ and a boolean CQ $q()$ as follows. $\mathcal{O}_F$ has one constant for each letter $l \in \mathcal{P}$, one constant c_i for each clause C_i, plus two constants *true* and *false* for the corresponding propositional constants. The atomic concepts of $\mathcal{O}_F$ are O, the atomic roles are P_1, P_2, N_1, N_2, and the atomic attribute U. Also we fix two constants $true_i \in LS_{T_i}$ s.t. $(true_i)^{\mathcal{D}} \in T_{sup}$, $(true_i)^{\mathcal{D}} \notin T_j$, and $false_j \in LS_{T_j}$ s.t. $(true_j)^{\mathcal{D}} \in T_{sup}$, $(false_j)^{\mathcal{D}} \notin T_i$. By assumption, $true_i$ and $false_j$ exist. Then, we define:

$$
\begin{aligned}
\mathcal{T} =& \{O \sqsubseteq \exists U, \mathsf{Rng}(U) \sqsubseteq T_{sup}\}, \\
\mathcal{A}_F =& \{O(l_1), \ldots, O(l_m)\} \\
& \cup \ \{P_1(c_k, l^k_{1+}), P_2(c_k, l^k_{2+}), N_1(c_k, l^k_1), N_2(c_k, l^k_2) | 1 \leq k \leq n\} \\
& \cup \ U(l_{true}, true_i) \\
& \cup \ U(l_{false}, false_j)
\end{aligned}
$$

and a $\mathsf{CQ}_{\mathcal{D}}$:

$$
\begin{aligned}
q() \leftarrow & P_1(c, f_1), U(f_1, vf_1), T_j(vf_1), \\
& P_2(c, f_2), U(f_2, vf_2), T_j(vf_2), \\
& N_1(c, t_1), U(t_1, vt_1), T_i(vt_1), \\
& N_2(c, t_2), U(t_2, vt_2), T_i(vt_2).
\end{aligned}
$$

Intuitively, $O \sqsubseteq \exists U$ should represent assignment of a variable $l \in \mathcal{P}$ to a truth value where $value(l) = true$ iff exists $d \in T_i$ s.t. $(l, \langle d \rangle) \in U(*)$. P_1, P_2 are binded to positive literals, while N_1, N_2 are binded to negative literals. Intuitively, if $() \in cert(q, \mathcal{O}_F)$ holds then for each model $\mathcal{I}$ of O_F there must be one clause which assigns "negative values" from T_j to literals in P_1 and P_2 and "positive values" from T_i to literals in N_1 and N_2 by "assignment" U. Notice that T_i and T_j need not to be disjoint as the truth values of literals are greedily defined on the side of "positive values"(see above (*)). It is only important that T_i and T_j "cover" all values in T_{sup}.

Finally, constants *true* and *false* are encoded throughout literals l_{true} and l_{false} respectfully.

Formally, F is unsatisfiable iff $() \in cert(q, \mathcal{O}_F)$.

($\Rightarrow$:) Assume F is unsatisfiable and $\mathcal{I} \models \mathcal{O}_F$. Let β be a truth assignment over variables in $\mathcal{P}$ defined by: $\beta(l) = true$ iff there exists $d_i \in T_i$ s.t. $(l, d_i)^{\mathcal{I}} \in U^{\mathcal{I}}$. Since F is unsatisfiable, there exists a clause C_k s.t. $\beta(C_k) = false$, i.e. $\beta(l^k_{1+}) = false, \beta(l^k_{2+}) = false, \beta(l^k_{1-}) = true, \beta(l^k_{2-}) = true$. Therefore, we know that for each d^1_j, d^2_j s.t. $(l^k_{1+}, d^1_j)^{\mathcal{I}}, (l^k_{1+}, d^1_j)^{\mathcal{I}} \in U^{\mathcal{I}}$ must be $d^1_j, d^2_j \in$

T_j and that exists such d_j^1, d_j^2. In addition, there must exists $d_i^1, d_i^2 \in T_i$ s.t. $(l_{1+}^i, d_i^1)^{\mathcal{I}}, (l_{1+}^i, d_i^1)^{\mathcal{I}} \in U^{\mathcal{I}}$.

Finally, there exists substitution $\theta = \{c \mapsto c_k, f_1 \mapsto l_{1+}^k, f_2 \mapsto l_{2+}^k, vf_1 \mapsto d_j^1, vf_2 \mapsto d_j^2, t_1 \mapsto l_{1-}^k, t_2 \mapsto l_{2-}^k, vt_1 \mapsto d_i^1, vt_2 \mapsto d_i^2\}$ that assigns variables in $q()$ s.t. $q()$ evaluates to true in $\mathcal{I}$.Notice that literals l_{true} and l_{false} complies with the definition of θ.

($\Leftarrow$:) Suppose that F is satisfiable, where $\beta : \mathcal{P} \to \{true, false\}$ is satisfiable assignment. Based on β we define a model $\mathcal{I}$ of $\mathcal{O}_F$ that falsifies $q()$.

$$
\begin{aligned}
O^{\mathcal{I}} =& \{l^{\mathcal{I}} | l \in \mathcal{P}\} \\
U^{\mathcal{I}} =& \{(l^{\mathcal{I}}, true_i^{\mathcal{I}}) | if\ \beta(l) = true\} \cup \{(l_{true}, true_i)\} \cup \\
& \{(l^{\mathcal{I}}, false_j^{\mathcal{I}}) | if\ \beta(l) = false\} \cup \{(l_{false}, false_i)\} \\
P^{\mathcal{I}} =& \{(a_1^{\mathcal{I}}, a_2^{\mathcal{I}}) | P(a_1, a_2) \in \mathcal{A}_F\}, \text{ for } P \in \{P_1, P_2, N_1, N_2\}
\end{aligned}
$$

It is easy to verify that $\mathcal{I}$ is a model of $\mathcal{O}_F$.

On the other hand, if $\mathcal{I}$ satisfies q there must be at least one clause C_k for which which $I \models P_1(c_k, f_1), U(f_1, vf_1), T_j(vf_1), \ldots, T_i(vt_2)$. $\mathcal{I}$ determines exact one substitution for variables of a fixed clause c_k. If we substitute variables in $q()$ with the only possible substitution, since $\beta(C_k) = true$ we know that at least of the following conjunctions will not hold in $\mathcal{I}$:

$$
\begin{aligned}
& P_1^{\mathcal{I}}(c_k^{\mathcal{I}}, (l_{1+}^k)^{\mathcal{I}}), U^{\mathcal{I}}((l_{1+}^k)^{\mathcal{I}}, vf_1^{\mathcal{I}}), T_j(vf_1^{\mathcal{I}}), \\
& P_2^{\mathcal{I}}(c_k^{\mathcal{I}}, (l_{2+}^k)^{\mathcal{I}}), U^{\mathcal{I}}((l_{2+}^k)^{\mathcal{I}}, vf_2^{\mathcal{I}}), T_j(vf_2^{\mathcal{I}}), \\
& N_1^{\mathcal{I}}(c_k^{\mathcal{I}}, (l_{1-}^k)^{\mathcal{I}}), U^{\mathcal{I}}((l_{1-}^k)^{\mathcal{I}}, vt_1^{\mathcal{I}}), T_i(vt_1^{\mathcal{I}}), \text{ or} \\
& N_2^{\mathcal{I}}(c_k^{\mathcal{I}}, (l_{2-}^k)^{\mathcal{I}}), U^{\mathcal{I}}((l_{2-}^k)^{\mathcal{I}}, vt_2^{\mathcal{I}}), T_i(vt_2^{\mathcal{I}}).
\end{aligned}
$$

We select an arbitrary clause c_k it follows follows $() \notin q^{\mathcal{I}}$. To conclude, $() \notin cert(\mathcal{O}_F, q())$. □

Corollary 1. *Let $\mathcal{O}$ be a DL-Lite$_{core}^{(\mathcal{HF})} + \mathcal{D}$ ontology defined over a datatype lattice $\mathcal{D}$ where exists datatypes T_i and T_{sup}, s.t. $T_i \sqsubseteq T_{sup}$ and the number of values in T_{sup} that are not in T_i is finite. Then answering $\mathsf{UCQ}_{\mathcal{D}}$ over $\mathcal{O}$ is not FOL-rewritable.*

Proof. The proof follows from the proof of lemma21, by adoption of the ABox $\mathcal{A}_F$ and the query q. By assumption $VS_{rest} = VS_{\sup} \setminus VS_{T_i}$ is a finite set of the size k. $\mathcal{A}_F$ is extended with assertions $\{V(l, d) | l \in O \wedge d^{\mathcal{D}} \in VS_{rest}\}$. The size of a new TBox is fixed as before, while the size of $\mathcal{A}_F$ is linearly bigger for $k \times |\mathcal{A}_F| \in \mathcal{O}(|\mathcal{A}_F|)$ (data complexity). Query $q()$ is changed but is still a fixed size:

$$
\begin{aligned}
q() \leftarrow & P_1(c, f_1), U(f_1, vf_1), T_i(vf_1), \\
& P_2(c, f_2), U(f_2, vf_2), T_i(vf_2), \\
& N_1(c, t_1), U(t_1, vt_1), V(t_1, vt_1), \\
& N_2(c, t_2), U(t_2, vt_2), V(t_2, vt_2).
\end{aligned}
$$

The rest of the proof is the same as in Lemma 21. □

Corollary 2. *Let $\mathcal{O}$ be a DL-Lite$_{core}^{(\mathcal{HF})} + \mathcal{D}$ ontology defined over a datatype lattice $\mathcal{D}$ where exists three different datatypes T_{sup}, T_i, T_j s.t. $T_{sup} \not\sqsubseteq T_i$, $T_{sup} \not\sqsubseteq T_j$ and the number of values in T_{sup} that are not neither in T_i nor in T_j is a finite number. Then answering $\mathsf{UCQ}_{\mathcal{D}}$ over $\mathcal{O}$ is not FOL-rewritable.*

Proof. Follow from the proof of Lemma21, by adoption of an ABox $\mathcal{A}_F$ and TBox $\mathcal{T}$. By assumption $VS_{rest} = VS_{\text{sup}} \setminus (VS_{T_i} \cup VS_{T_j})$ is a finite set of the size k. $\mathcal{A}_F$ is extended with assertions $\{V(l,d) | l \in O \wedge d \in VS_{rest}\}$. TBox is extended with $\{V \sqsubseteq \neg U\}$. The size of a new TBox is fixed as before, while the size of $\mathcal{A}_F$ is linearly bigger for $k \times |\mathcal{A}_F| \in \mathcal{O}(|\mathcal{A}_F|)$ (data complexity).

The rest of the proof is the same as in Lemma 21. Intuitively V will determine inclusion $O \sqsubseteq \exists U$ to "assigns" $l \in O$ to values from T_i or T_j (and not their difference with T_{sup}) which leads exactly to the case we have in Lemma 21. □

Corollary 3. *Let $\mathcal{O}$ be a DL-Lite$_{core}^{(\mathcal{HF})} + \mathcal{D}$ ontology defined over a datatype lattice $\mathcal{D}$ where $\mathcal{D} = \{T_i, T_j\}$ and $T_i \not\sqsubseteq T_j$,$T_j \not\sqsubseteq T_i$. Then answering $\mathsf{UCQ}_{\mathcal{D}}$ over $\mathcal{O}$ is not FOL-rewritable.*

Proof. The proof follows from the proof of lemma 21., by replacing $\mathsf{Rng}(U) \sqsubseteq T_{sup}$ with $\mathsf{Rng}(()U) \sqsubseteq T_d$. The rest of the proof proceed in the same fashion. □

Lemma 22. *Let $\mathcal{O}$ be a DL-Lite$_{core}^{(\mathcal{HF})}$+$\mathcal{D}$ ontology defined over a datatype lattice $\mathcal{D}$ where $\mathcal{D} = \{T_1, \ldots, T_k\}$, for $k \geq 3$, where at least two are not subtype of any other. Then answering $\mathsf{UCQ}_{\mathcal{D}}$ over $\mathcal{O}$ is not FOL-rewritable.*

Proof. We assume, that each datatype T_i $(1 \leq i \leq k)$ is not a sub-type of any other datatype T_j $(i \neq j)$ in $\mathcal{D}$. If this is not a case then we can extract a subset of datatypes from $\mathcal{D}$ that posses this property. If there is exactly two datatypes with such property then we can apply lemma 3. Otherwise we set k to the subset size and in the following operate only on the subset.

The k-coloring problem is problem of asking whether an undirected graph $G = \langle V, E \rangle$ admits coloring with $k \geq 3$ colors. A graph admits k-coloring if exists a coloring function $\beta : V \to \{c_1, \ldots, c_n\}$ s.t. for each $(v_i, v_j) \in E$ holds $\beta(v_i) \neq \beta(v_j)$. In simple words, if there exists coloring of the graph vertices such that no two adjacent vertices have the same color. Deciding whether an undirected graph $G = \langle V, E \rangle$ admits k-coloring $(k \geq 3)$ in NP-complete. Then deciding opposite, whether an undirected graph does not admit k-coloring $(k \geq 3)$ in coNP-hard.

We will reduce the k-coloring complement problem of a graphs $G = \langle V, E \rangle$ to the problem of answering $\mathsf{UCQ}_{\mathcal{D}}s$ over DL-Lite$_{core}^{(\mathcal{HF})} + \mathcal{D}$ ontologies. With $\mathcal{O}_G = \langle \mathcal{T}, \mathcal{A}_G \rangle$ we denote a DL-Lite$_{core}^{(\mathcal{HF})} + \mathcal{D}$ ontology that encodes the graph. For every graph only $\mathcal{A}_G$(data complexity measure) will change linearly in the size of the input G, while the TBox $\mathcal{T}$ and the $\mathsf{UCQ}_{\mathcal{D}}$ $q()$ are fixed. As a consequence, data complexity of answering $\mathsf{UCQ}_{\mathcal{D}}s$ will be coNP-hard. Then $\mathsf{UCQ}_{\mathcal{D}}$ answering is not FOL-rewritable in DL-Lite$_{core}^{(\mathcal{HF})} + \mathcal{D}$.

Assume we are given an undirected graph $G = \langle V, E \rangle$. We define an ontology $\mathcal{O}_G = \langle \mathcal{T}, \mathcal{A}_G \rangle$.

$$\mathcal{T} = \{V \sqsubseteq \exists U\},$$

$$\mathcal{A}_G = \{V(v_i) | v_i \in V\} \cup \{E(v_i, v_j) | (v_i, v_j) \in E\}$$

and a $\mathsf{UCQ}_{\mathcal{D}}$:

$$q() \leftarrow U(x_1, c_1), U(x_2, c_2), E(x_1, x_2), T_1(c_1), T_1(c_2).$$
$$\dots$$
$$q() \leftarrow U(x_1, c_1), U(x_2, c_2), E(x_1, x_2), T_k(c_1), T_k(c_2).$$

Where U is an atomic attribute, V is an atomic concept, and E is an atomic role. Intuitively, $V \sqsubseteq \exists U \in \mathcal{T}$ will simulate an assignment of the colors to the vertices. Each datatype T_i simulates a color c_i. In addition we distinguish k constants for each datatype: $c_i \in LS_{T_i}$.

Now we claim:

$$G \text{ does not admit k-coloring iff } () \in \mathit{cert}(\mathcal{O}_G, q())$$

($\Rightarrow$:) Suppose $() \in \mathit{cert}(\mathcal{O}_G, q())$. Now let $\beta : V \to \{c_1, \dots, c_n\}$ be an arbitrary color assignment. Now we construct a model $\mathcal{I}_\beta$ of $\mathcal{O}_G$. For every vartex $v^{\mathcal{I}_\beta} = v$. Furthermore,

$$V^{\mathcal{I}_\beta} = \{v | v \in V\}$$
$$U^{\mathcal{I}_\beta} = \{(v, c_i) | \mathit{if}\ \beta(v) = c_i\}$$
$$E^{\mathcal{I}_\beta} = \{(v_i, v_j) | (v_i, v_j) \in E\}$$

Obviously, $\mathcal{I}_\beta \models \mathcal{O}_G$. By assumption $q^{\mathcal{I}_\beta} = \{()\}$, so there exists i $(1 \leq i \leq k)$, s.t.

$$\mathcal{I}_\beta \models \exists x_1, x_2, c_1, c_2 . U(x_1, c_1), U(x_2, c_2), E(x_1, x_2), T_i(c_1), T_i(c_2).$$

In other words, there exists vertices v_i and v_j s.t. $(v_i, v_j) \in E$ and $\beta(v_i) = c_k = \beta(v_j)$. So β is not proper coloring. On the other hand, we select β arbitrary, so G does not admit k-coloring.

($\Leftarrow$:) Assume G does not admit k-coloring and let $\mathcal{I} \models O_G$. Now we define color assignment β based on the $\mathcal{I}$ interpretation of $\mathcal{U}$.

For every $v \in V$ define $\beta(v)$:

$$\begin{aligned}
&\text{If exists } c \in VS_{T_1} \text{ s.t. } (v^{\mathcal{I}}, c) \in U^{\mathcal{I}} && \text{then } \beta(v) = c_1 \\
&\text{else if exists } c \in VS_{T_2} \text{ s.t. } (v^{\mathcal{I}}, c) \in U^{\mathcal{I}} && \text{then } \beta(v) = c_2 \\
&\qquad \dots \\
&\text{else if exists } c \in VS_{T_{k-1}} \text{ s.t. } (v^{\mathcal{I}}, c) \in U^{\mathcal{I}} && \text{then } \beta(v) = c_{k-1} \\
&\text{else if exists } c \in VS_{T_k} \text{ s.t. } (v^{\mathcal{I}}, c) \in U^{\mathcal{I}} && \text{then } \beta(v) = c_k.
\end{aligned}$$

β is correctly defined as for each $v \in V$ it assigns exactly one color and for each $v \in V$ there must exists at least one value c s.t. $(v^{\mathcal{I}}, c) \in U^{\mathcal{I}}$. By assumption G is not k-colorable so there exists v_i and v_j s.t. $(v_i, v_j) \in E$ and $\beta(v_i) = c_z = \beta(v_j)$ $(1 \leq z \leq k)$.

On the other hand, $\mathcal{I}$ is a model of $\mathcal{O}_G$, so $(v_i^{\mathcal{I}}, v_j^{\mathcal{I}}) \in E^{\mathcal{I}}$, and by definition of β exists $c_z^1, c_z^2 \in VS_{T_z}$ s.t. $(v_i^{\mathcal{I}}, c_z^1) \in U^{\mathcal{I}}$ and $(v_j^{\mathcal{I}}, c_z^2) \in U^{\mathcal{I}}$. Then $() \in q()^{\mathcal{I}}$.

We selected $\mathcal{I}$ arbitrary, so $() \in \mathit{cert}(\mathcal{O}_G, q())$. □

Corollary 4. *Let $\mathcal{O}$ be a DL-Lite$_{core}^{(\mathcal{HF})} + \mathcal{D}$ ontology defined over a datatype lattice $\mathcal{D}$ where exists $T_{sup}, T_1, \dots, T_k$ ($k \geq 2$) s.t. $T_{sup} \not\sqsubseteq T_i$ ($1 \leq i \leq k$), $VS_{T_{sup}} \setminus \bigcup_{i=1}^{k} VS_{T_i}$ is finite. Then query answering over $\mathcal{O}$ is not FOL-rewritable.*

Proof. We distinguish three special cases.

If exists T_i that contains all other T_i's, then we apply the lemma 1.

Otherwise, if there exists exactly two T_i's that contain all other T_i's, then we apply lemma 3.

Otherwise, there must exists at least three different T_i's that are not proper sub-type of some other T_i. Then we adopt the proof of lemma 22., by adding $\{\rho(U) \sqsubseteq T_{sup}, U \sqsubseteq \neg U_{diff}\}$ to TBox and $\{U_{diff}(v,d) | v \in V \wedge d \in VS_{T_{sup}} \setminus \bigcup_{i=1}^{k} VS_{T_i}\}$ to ABox. In the rest, we can proceed in the same fashion as in the proof of the lemma. □

Chapter 7

Conclusions and Future work

7.1 Conclusion

In this thesis we were aiming at providing a solid base for the investigation of datatypes in the context of *DL-Lite* DLs. For this purpose we define a language for describing datatypes, as well as the constructors for defining new datatypes based on defined ones.

Additionally, we were aiming at exploring the computational effects that datatypes can have on the desired properties. The main computational properties of our concern are FOL-rewritability of ontology satisfiability and FOL-rewritability of query answering. We wanted to explore necessary and sufficient conditions over datatypes so that a DL preserves "nice" computational properties. Or on the contrary, we were trying to realize the source of the problems that increase the complexity.

For these reasons, we have studied restrictions imposed over datatypes under which we can guarantee the desired computational properties. These conditions are formalized through the structure of a datatype lattice, a finite set of primitive and derived datatypes. We classify datatype lattices, in three classes named $\mathcal{D}_0$, $\mathcal{D}_1$ and $\mathcal{D}_2$ ($\mathcal{D}_0 \supset \mathcal{D}_1 \supset \mathcal{D}_2$), based on the conditions that a datatype lattice satisfies or not.

Traditionally, *DL-Lite* candidates for OBDA were investigated based only on the ontology expressibility [10, 4]. Furthermore, they were investigated wrt a fixed query language (UCQ) and without considering datatypes. We define a more abstract view over the OBDA scenario, by identifying datatypes and query language as important parts of OBDA, that can vary in expressivity based on the needs. We denote such a framework with $\mathcal{DL}+\mathcal{D}+\mathcal{Q}$. The first component is a language $\mathcal{DL}$, that describes the ontology language together with the interface for combining ontologies with datatype lattices. The second component $\mathcal{D}$ is the type of admissible datatype lattices. The last component is a query language $\mathcal{Q}$. In this terminology the traditional view on OBDA can be described as $\mathcal{DL} + \emptyset + \mathsf{UCQ}$.

To support our investigation, we explored two significant candidates for the OBDA framework. Namely, $\textit{DL-Lite}_{core}^{(\mathcal{HF})} + \mathcal{D}_1 + \mathsf{UCQ}$ and $\textit{DL-Lite}_{core}^{(\mathcal{HF})} + \mathcal{D}_2 +$

$\mathsf{UCQ}_{\mathcal{D}}$. In both cases we proceed in two steps. In the first one, we prove that satisfiability of ontologies is FOL-rewritable. And in the second one, we show that a framework is eligible for the OBDA scenario, i.e., that is FOL-rewritable. For the second case we extend standard UCQs with datatypes (constraints) and obtain a query language called $\mathsf{UCQ}_{\mathcal{D}}$.

Moreover, we showed that in case of $\mathit{DL\text{-}Lite}_{core}^{(\mathcal{HF})} + \mathcal{D}_1 + \mathsf{UCQ}$ datatype conditions defined over $\mathcal{D}_1$ are necessary if we want to preserve FOL-rewritability. We proved the same for the case of $\mathcal{D}_2$ conditions in $\mathit{DL\text{-}Lite}_{core}^{(\mathcal{HF})} + \mathcal{D}_2 + \mathsf{UCQ}_{\mathcal{D}}$.

7.2 Future work

Extensions of the work presented in the thesis can go in two directions. The first is further exploring and characterizing the $\mathcal{DL} + \mathcal{D} + \mathcal{Q}$ framework. In particular, finding maximal triples for query languages, DL languages , and datatype lattices , that are FOL-rewritable. The second direction is an implementation of $\mathcal{DL} + \mathcal{D} + \mathcal{Q}$ implementation. We present each in a more detail:

Extending $\mathcal{DL} + \mathcal{D} + \mathcal{Q}$

Already in the *DL-Lite* family we have DL constructors that we haven't considered in the thesis. For example, an expressive *DL-Lite* member is $\mathit{DL\text{-}Lite}_{horn}^{(\mathcal{HN})}$ [4], which includes concept expressions of the form $\geq k\ R$ (generalize $\exists R$), where $\geq k\ R$ means that there exists at least k succors via role R. In addition, $\mathit{DL\text{-}Lite}_{horn}^{(\mathcal{HN})}$ allows conjunction of concepts on the lhs of a concept inclusion, $B_1 \sqcap \ldots \sqcap B_k \sqsubseteq B$. Most importantly, $\mathit{DL\text{-}Lite}_{horn}^{(\mathcal{HN})}$ is FOL-rewritable for UCQs.

Further more, one can extend the datatype interface. For example, by allowing the expressions of the form $B \sqsubseteq \forall U.T_i$. Even modest usage of the $\forall$ quantifier over roles would spoil FOL-rewritability of query answering for *DL-Lite* members (NLogSpace-hard for data complexity [10]). However, defined over attributes only, it has potential to be introduced for "free".

Another expressive feature in DLs are datatype linear equations. They are used to express dependencies between the values on different attributes over a single individual. They have been theoretically founded in $\mathcal{EL}$ by means of Concrete Domains [8]. For example, consider an ontology that models substances, where an attribute *MeltingPoint* defines the melting temperature of a substance and an attribute *BoilingPoint* defines the boiling temperature of a substance. To relate those two we add an inclusion

$$\mathit{NormalSupstance} \sqsubseteq \forall(\mathit{MeltingPoint} \leq \mathit{BoilingPoint}),$$

which states that for each normal substance the melting point should always be less or equal than the boiling point. Datatype linear equations are included in OWL2 syntax, equipped with standard arithmetic comparisons and arithmetic operators. [1]

Lastly, we can consider extending the query language $\mathcal{Q}$. We notice that a perfect reformulation of a UCQ (resp., a $\mathsf{UCQ}_{\mathcal{D}}$) is again a UCQ (resp., a $\mathsf{UCQ}_{\mathcal{D}}$), which is less expressive then FOL queries (full SQL). Then we can ask, what would be queries of maximal expressiveness that are still FOL-rewritable over

[1] http://www.w3.org/TR/owl2-dr-linear/

particular *DL-Lite* members. For example, a simple extension of CQs can be obtained by adding concepts of the form $\geq k\ R$ (similar the to SQL count(*) function).

Implementing $\mathcal{DL} + \mathcal{D} + \mathcal{Q}$

In a real scenario, data in a repository are not stored as ABox instances, i.e., tables with one or two columns. Additionally, data in a repository are typed and do not represent objects directly. So in order to implement OBDA we will have to define a mapping language that will solve the problem, so called *impedance mismatch* problem. The authors in [10], propose a GAV mapping language between databases and ontologies. In short, the language has two components, the mapping definitions that assign database columns to ontology datatypes, and the second one that defines mappings from database data to ontology ABox. In such settings, there is a need for verification methods over mapping definitions (without employing ABox), that can be in collision with the TBox and datatypes dependencies.

On the other hand, a serious issue of OBDA applicability is the size of a perfect reformulation, which is exponential in the size of the input query (eq. 5.2 and 6.2). However, potentially many CQs in a perfect reformulation are inconsistent wrt terminological part. It means they are redundant in query evaluation over the ABox and we can discard them without evaluation. Different optimization techniques have been employed to reduce the number of redundant queries in perfect reformulation [20]. By introducing datatypes, there is a potential to enhance optimization techniques, based on both the inner properties of a datatype lattice and on the mapping definitions that include datatypes.

Bibliography

[1] 11404:2007, I. Information technology : General purpose datatypes (gpd). Tech. rep., CH-1211 Geneva 20, Switzerland, 2007.

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

[3] Artale, A., Calvanese, D., Kontchakov, R., Ryzhikov, V., and Zakharyaschev, M. Reasoning over extended er models. In *Proc. of the 26th Int. Conf. on Conceptual Modeling (ER 2007)* (2007), vol. 4801 of *Lecture Notes in Computer Science*, Springer, pp. 277–292.

[4] Artale, A., Calvanese, D., Kontchakov, R., and Zakharyaschev, M. The *DL-Lite* family and relations. 1–69.

[5] Artale, A., Ibáñez-García, Y. A., Kontchakov, R., and Ryzhikov, V. Dl-lite with attributes and sub-roles (extended abstract). In *Description Logics* (2011).

[6] Baader, F., Brand, S., and Lutz, C. Pushing the el envelope. In *In Proc. of IJCAI 2005* (2005), Morgan-Kaufmann Publishers, pp. 364–369.

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

[8] Baader, F., and Hanschke, P. A scheme for integrating concrete domains into concept languages. Tech. Rep. RR-91-10, 1991.

[9] Berardi, D., Calvanese, D., and De Giacomo, G. Reasoning on uml class diagrams. *Artificial Intelligence 168*, 1-2 (2005), 70–118.

[10] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodríguez-Muro, M., and Rosati, R. Ontologies and databases: The *DL-Lite* approach. vol. 5689. 2009, pp. 255–356.

[11] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., and Rosati, R. Inconsistency tolerance in p2p data integration: an epistemic logic approach. *Information Systems 33*, 4 (2008), 360–384.

[12] Calvanese, D., De Giacomo, G., and Lenzerini, M. Description logics for information integration. In *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski*, A. Kakas and F. Sadri, Eds., vol. 2408 of *Lecture Notes in Computer Science*. Springer, 2002, pp. 41–60.

[13] Chandra, A. K., and Merlin, P. M. Optimal implementation of conjunctive queries in relational data bases. pp. 77–90.

[14] Lenzerini, M. Data integration: A theoretical perspective. pp. 233–246.

[15] Lutz, C. Description logics with concrete domains-a survey. In *Advances in Modal Logic* (2002), pp. 265–296.

[16] Magka, D., Kazakov, Y., and Horrocks, I. Tractable extensions of the description logic el with numerical datatypes. In *Description Logics* (2010).

[17] Motik, B., and Horrocks, I. Owl datatypes: Design and implementation. In *Proceedings of the 7th International Conference on The Semantic Web* (Berlin, Heidelberg, 2008), ISWC '08, Springer-Verlag, pp. 307–322.

[18] Papadimitriou, C. H. *Computational Complexity*. 1994.

[19] Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., and Rosati, R. Linking data to ontologies. *J. on Data Semantics X* (2008), 133–173.

[20] Rodriguez-Muro, M., and Calvanese, D. Dependencies to optimize ontology based data access. In *Proc. of the 24th Int. Workshop on Description Logics (DL 2011)* (2011), vol. 745 of *CEUR Electronic Workshop Proceedings, http://ceur-ws.org/*.

[21] Rodriguez-Muro, M., Lubyte, L., and Calvanese, D. Realizing ontology based data access: A plug-in for protg. In *Proc. of the ICDE Workshop on Information Integration Methods, Architectures, and Systems (IIMAS 2008)* (2008), IEEE Computer Society Press, pp. 286–289.

[22] Schaerf, A. On the complexity of the instance checking problem in concept languages with existential quantification. *J. Intell. Inf. Syst. 2* (September 1993), 265–278.

[23] Vollmer, H. *Introduction to Circuit Complexity: A Uniform Approach*. 1999.