Metamodelling and metaquerying in OWL 2 QL: semantics and algorithms

Dipartimento di Ingegneria Informatica, Automatica
e Gestionale “Antonio Ruberti”
Dottorato di Ricerca in Ingegneria Informatica – XXVIII Ciclo

Candidate
Lorenzo Lepore
ID number 799912

Thesis Advisor
Prof. Maurizio Lenzerini
Co-Advisor
Prof. Daniele Nardi

A thesis submitted in partial fulfillment of the requirements
for the degree of PhD in Engineering in Computer Science
April 2017
Thesis not yet defended
To Marisa and Antonio
# Contents

1 Introduction and foundations

## 1 Introduction

1.1 Contributions of the thesis ........................................... 5
1.2 Organization of the thesis ........................................... 7

## 2 Description logics

2.1 The Description Logic SROIQ ........................................... 9
2.1.1 Syntax of SROIQ .................................................. 9
2.1.2 Semantics of SROIQ .............................................. 10
2.2 The DL-Lite family of Description Logics ......................... 12
2.2.1 Syntax of DL-LiteR .............................................. 12
2.2.2 Semantics of DL-LiteR .......................................... 13
2.3 Reasoning in Description Logics ................................. 13
2.3.1 Basic inference problems ..................................... 14
2.3.2 Answering (union of) conjunctive queries .................... 14
2.4 Complexity of reasoning in SROIQ .............................. 16
2.5 Complexity of reasoning in DL-LiteR ............................ 17

## 3 Data integration through OBDM

3.1 Data integration ..................................................... 19
3.2 The OBDM paradigm ................................................ 21

## 4 OWL 2

4.1 The Resource Description Framework ............................ 25
4.1.1 Simple semantics ............................................ 26
4.1.2 RDF semantics ............................................... 27
4.1.3 RDFS semantics .............................................. 28
4.2 OWL 2 syntax ....................................................... 29
4.2.1 OWL 2 entities ............................................... 29
4.2.2 OWL 2 expressions .......................................... 30
4.2.3 OWL 2 axioms ............................................... 31
4.2.4 OWL 2 DL syntactic restrictions ............................ 33
4.3 OWL 2 semantics .................................................. 33
4.3.1 Direct Semantics ........................................... 34
4.3.2 RDF-Based Semantics .................................... 39
4.4 OWL 2 QL .......................................................... 44
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4.1 OWL 2 QL syntax</td>
<td>44</td>
</tr>
<tr>
<td>4.4.2 Reasoning in OWL 2 QL</td>
<td>46</td>
</tr>
<tr>
<td>4.5 The SPARQL query language</td>
<td>47</td>
</tr>
<tr>
<td>4.6 Querying OWL 2 ontologies with SPARQL</td>
<td>48</td>
</tr>
<tr>
<td>4.6.1 The OWL 2 Direct Semantics Entailment Regime</td>
<td>48</td>
</tr>
</tbody>
</table>

## A semantics for metamodeling in the semantic web                       53

### 5 Motivating scenario

- 5.1 A real world OBDM scenario requiring metamodeling                  55
- 5.2 Metamodeling and metaquerying in OWL 2 standard semantics          58

## Adding proper metamodeling capabilities to OWL 2                         61

- 6.1 The language Hi(OWL 2)                                             61
  - 6.1.1 Syntax of Hi(OWL 2)                                            61
  - 6.1.2 Semantics of Hi(OWL 2)                                        61
- 6.2 The language Hi(OWL 2 QL)                                          70
- 6.3 Query language                                                     72

## Reasoning in Hi (OWL 2 QL)                                              77

### 7 Satisfiability and answering instance queries

- 7.1 The OWL 2 QL canonical pseudo-interpretation under MMS              79
  - 7.1.1 Chase procedure for OWL 2 QL under MMS                        79
  - 7.1.2 Canonical pseudo-interpretation of a OWL 2 QL ontology         85
  - 7.1.3 Properties of the canonical pseudo interpretation             87
- 7.2 Ontology satisfiability                                            93
- 7.3 Answering instance queries                                        98

### 8 Answering general queries

- 8.1 Answering general queries over TBox-complete ontologies            101
- 8.2 Answering general queries                                          127

## Complexity analysis                                                    131

- 9.1 Upper bounds                                                       131
  - 9.1.1 Ontology satisfiability                                       131
  - 9.1.2 Answering instance queries                                   132
  - 9.1.3 Answering queries over TBox-complete ontologies              132
  - 9.1.4 Answering queries over general ontologies                    133
- 9.2 Lower bounds                                                       133

## Practical applicability, implementation and evaluation                  147

### 10 An optimized algorithm for query answering

- 10.1 The LMG algorithm                                                149
- 10.2 Error calculation                                                 151
Contents

10.2 Properties of LMG ........................................... 158

11 Implementation and evaluation ................................. 163
  11.1 The MQ-MASTRO system ..................................... 163
  11.2 The MQ-MASTRO plugin for PROTEGE .................... 164
  11.3 Evaluation .................................................. 166

12 Conclusion and future work ................................ 171
List of Figures

3.1 Example database .................................................. 23

5.1 Chain of instanceof relations between entities in the Italian Public Debt domain .............................................. 57

6.1 Interpretation structure ................................................ 65

10.1 Computation of the labeled graph associated to the query of Example 77 ......................................................... 155

11.1 Architecture of the MQ-MASTRO system ......................... 164

11.2 Architecture of the MQ-MASTRO plugin for PROTEGE .............. 165

11.3 Graphical interface of the MQ-MASTRO plugin for PROTEGE .............. 165
## List of Tables

<table>
<thead>
<tr>
<th>Number</th>
<th>Table Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>RDFS vocabulary</td>
<td>28</td>
</tr>
<tr>
<td>4.2</td>
<td>RDFS axiomatic triples</td>
<td>28</td>
</tr>
<tr>
<td>4.3</td>
<td>OWL 2 predefined datatypes</td>
<td>30</td>
</tr>
<tr>
<td>4.4</td>
<td>OWL 2 Complex class expressions</td>
<td>30</td>
</tr>
<tr>
<td>4.5</td>
<td>OWL 2 Complex data range expressions</td>
<td>30</td>
</tr>
<tr>
<td>4.6</td>
<td>OWL 2 Declaration axioms</td>
<td>31</td>
</tr>
<tr>
<td>4.7</td>
<td>OWL 2 ABox axioms</td>
<td>32</td>
</tr>
<tr>
<td>4.8</td>
<td>OWL 2 TBox axioms</td>
<td>32</td>
</tr>
<tr>
<td>4.9</td>
<td>OWL 2 RDF-Based vocabulary</td>
<td>40</td>
</tr>
<tr>
<td>4.10</td>
<td>OWL 2 QL datatypes</td>
<td>45</td>
</tr>
<tr>
<td>4.11</td>
<td>OWL 2 QL ABox axioms</td>
<td>45</td>
</tr>
<tr>
<td>4.12</td>
<td>OWL 2 QL TBox axioms</td>
<td>46</td>
</tr>
<tr>
<td>5.1</td>
<td>OWL 2 QL Italian Public Debt ontology with no metamodeling features</td>
<td>55</td>
</tr>
<tr>
<td>5.2</td>
<td>OWL 2 QL Italian Public Debt ontology with metamodeling features</td>
<td>58</td>
</tr>
<tr>
<td>6.1</td>
<td>H1 (OWL 2 QL) class expressions (a is an atomic expression, op is an object property expression)</td>
<td>70</td>
</tr>
<tr>
<td>6.2</td>
<td>H1 (OWL 2 QL) axioms (a is an atomic expression, op is an object property expression), c is a class expression, sub is a subclass expression, lt is a literal)</td>
<td>71</td>
</tr>
<tr>
<td>7.1</td>
<td>Chase rules for H1 (OWL 2 QL) ontologies</td>
<td>85</td>
</tr>
<tr>
<td>8.1</td>
<td>Violation sets of negative axioms</td>
<td>128</td>
</tr>
<tr>
<td>10.1</td>
<td>Function f</td>
<td>150</td>
</tr>
<tr>
<td>11.1</td>
<td>Test ontologies features</td>
<td>167</td>
</tr>
<tr>
<td>11.2</td>
<td>Test queries</td>
<td>168</td>
</tr>
<tr>
<td>11.3</td>
<td>Performance of LMG algorithm</td>
<td>169</td>
</tr>
<tr>
<td>11.4</td>
<td>Performance of NAIVE algorithm</td>
<td>169</td>
</tr>
<tr>
<td>11.5</td>
<td>Test queries (witnessed) metavariables</td>
<td>170</td>
</tr>
</tbody>
</table>
Part I

Introduction and foundations
Chapter 1

Introduction

Numerous are the areas of the computer science where the need of defining formal languages with metamodeling capabilities has arisen. Among them we can list database management systems [71,1,50], object-oriented programming languages, logic (meta)programming [23,27,51,60], fields of AI as machine learning, concept learning and datamining [6,35,18,12,30,59], web data exchange technologies and conceptual modeling. In particular, recent papers point out the need of enriching conceptual models and ontology languages with metamodeling and metaquerying features, i.e., features for specifying and reasoning about metaclasses (also called metaconcepts) and metarelations (also called metaproperties) [5,16]. Roughly speaking, a metaclass is a class whose instances can be themselves classes, a metarelation is a relationship (or, property) between metaclasses, and a metaquery is a query possibly using metaclasses and metarelations, and whose variables may be bound to predicates. Since metaclasses can be themselves instances of other metaclasses, multi-level modeling is another term used for metamodeling.

Consider the following example (61,79,63), concerning the modeling of a portion of the zoology domain. In this domain, Bird is the class representing the set of all birds, Eagle is an obvious subclass of Bird, and Golden Eagle a subclass of Eagle. These subclass relations imply, for example, that all golden eagles are eagles, and all eagles are birds, so that, if the individual Harry is asserted to be an instance of Golden Eagle, then we can infer that it is also an instance of both the class Eagle, and the class Bird. However, in zoology, one typically is not only interested in describing properties of specific living beings, but also to make statements about species themselves, such as “golden eagle is listed in the IUCN Red Lists of endangered species”. Note that this statement is not about a particular individual, e.g., we are not asserting that Harry is in danger. It is the golden specie as a whole that is listed in the IUCN Red List. Therefore, to model this statement formally, we can introduce the class RedListSpecies representing all the species listed in the IUCN Red List, and then assert that Golden Eagle is an instance of such class. Having a class among its instances makes RedListSpecies a metaclass. Note that, depending on the expressive power of the modeling language, we may be able to assert relevant properties on metaclasses. In the zoology domain, an interesting example of such properties is described in [61]: “no one is allowed to hunt animals of the species listed in the IUCN Red List”. If the modeling language has sufficient expressivity, this can be formalized by asserting the following logical property of the metaclass RedListSpecies: no instance of the class MayBeHunted can be instance of an instance of the class RedListSpecies. Note that this allow us to deduce that
“no one is allowed to hunt Harry.” Finally, if the ontology language has suitable features for metamodeling, then it is natural to express queries exploiting such features. For example, one might be interested to ask for all instances $x$ of RedListSpecies that are subclasses of Bird and that include Harry among their instances. Clearly, this is a metaquery, in the sense that in any natural formulation of such query, variable $x$ will range over classes, rather than individuals, as in usual first-order queries.

As we said before, in the last years there has been a huge interest in issues related to metamodeling and metaquerying. Broadly speaking, the works on this subject can be classified into two categories: those addressing the principles underlying metamodeling, and those investigating languages for metamodeling and metaquerying.

Recent papers on the former aspect, for example [16, 40], concentrate on providing effective guidelines supporting the modeler in producing correct multi-level ontologies, i.e., ontologies where classes and metaclasses coexist. The methodological issues underlying such works, although extremely valuable for the conceptual modeling community, are actually outside the scope of our work, that focuses instead on logic-based ontology languages for expressing metamodeling constructs, especially languages based on OWL 2, and algorithms for answering metaqueries over ontologies formulated in such languages. Thus, in the rest of this section, we concentrate on papers on the latter aspect, and we refer in particular to the OWL 2 ontology language, which is the W3C standard ontology language, and its fragments. A group of recent papers addresses the issues of metamodeling and metaquerying for DL-Lite. For example in [33, 70] a new semantics for Description Logics is proposed that takes metaclasses and metaqueries into account. The new semantics is inspired by the Hilog-style semantics [27], where every object $o$ in an interpretation is seen as an individual, a class and a relation. Indeed, a total function $ext$ is defined on the interpretation domain, in such a way that $ext(o)$ is the set of instances of $o$ when seen as a class (analogous extension functions specify the instances of $o$ when seen as a relation). Also, the papers show that for the logics of the DL-Lite family, and for a significant subclass of conjunctive queries, a particular technique, called metagrounding, provides the right tool for deriving sound and complete algorithms for answering instance metaqueries. Roughly speaking, metagrounding computes all possible instantiations of a metaquery, where each instantiation is obtained by substituting the metavariables with all the classes of the ontology (similarly for relations), and then answering the original query is reduced to answering all the instantiations thus computed (which are traditional, first-order queries). The most powerful OWL 2 language is OWL 2 Full that does not enforce a strict separation of classes, properties, individuals and data values. So, metamodeling is taken for granted in OWL 2 Full. In [61], a landmark paper in OWL 2 metamodeling, it is shown, however, that one of the reasons why reasoning in OWL 2 Full is undecidable is the power of its metamodeling features. With the goal of achieving decidability, both the Description Logic and the Semantic Web community concentrated their attention on OWL 2 DL, that is essentially a first-order language, with no explicit semantic support for metamodeling. Indeed, virtually all systems developed in the context of Description Logics and the Semantic Web support ontologies constrained by the restrictions required for OWL 2 DL.

There exist a number of approaches that are based on the idea of using the whole power of OWL 2 DL, actually OWL 2, for expressing ontologies. However, it is well-known that when very powerful languages are used to express ontologies, query answering become computationally intractable, if not undecidable. This barrier is even more significant in the context of the Ontology-Based Data Access (OBDM) paradigm [20], where ontologies
1.1 Contributions of the thesis

are used in conjunction of typically large quantity of data. An ontology-based data access system has a three-level architecture, constituted by the ontology, the sources, and the mapping between the two [71]. The ontology is a conceptual, formal description of the domain of interest to a given organization (or, a community of users), expressed in terms of relevant classes, attributes of classes, relationships between classes, and logical assertions characterizing the domain knowledge. The data sources are the repositories accessible by the organization where data concerning the domain are stored. In the general case, such repositories are numerous, heterogeneous, each one managed and maintained independently from the others. The mapping is a precise specification of the correspondence between the data contained in the data sources and the elements of the ontology. The main purpose of an OBDM system is to allow information consumers to query the data using the elements in the ontology as predicates. In this sense, OBDM can be seen as a form of information integration, where the usual global schema is replaced by the conceptual model of the application domain, formulated as an ontology expressed in a logic-based language. With this approach, the integrated view that the system provides to information consumers is not merely a data structure accommodating the various data at the sources, but a semantically rich description of the relevant concepts in the domain of interest, as well as the relationships between such concepts. Note that in OBDM scenarios, metamodeling and metaquerying are essential both for correctly capturing complex domains, and for performing interesting analyses, such as, for example, “find all data sources that contribute, through mappings, to the instances of C, or any of its superclass”.

Although OWL 2 DL provides syntactic support for metamodeling through OWL 2 punning (by which the same name can be used to denote ontology elements of different categories, such as a class and an individual), we argue that the official semantics of OWL 2, the so-called Direct Semantics, treats punning in a way that is not adequate for metamodeling. The reason is simply that proper metamodeling requires that the same element plays the role of both individual and class (or, class and relation), while DS sanctions that an individual and a class with the same name are different elements. This is confirmed by the fact that the Direct Semantics Entailment Regime, which is the logic-based semantics of SPARQL 1.1, when applied to OWL 2 QL, forces queries to obey the so-called typing constraint, which rules out the possibility of using the same variable in incompatible positions (for example, in individual and in class position).

In this thesis, we are interested in exploring the issue of metamodeling and metaquerying in the context of OBDM. For this reason, we focus on the most popular ontology language for OBDM applications, namely OWL 2 QL. This is a fragment of OWL 2 derived from the DL-Lite family [23] of Description Logics, that shares with DL-Lite the characteristic of allowing tractable reasoning and tractable query answering, even in the presence of huge quantity of data. Indeed, it is well-known that the computational complexity of answering conjunctive queries (akin to select-project join of SQL) over OWL 2 QL ontologies is the same as the complexity of answering such queries over a relational database.

1.1 Contributions of the thesis

In this thesis we study the topics of metamodeling and metaquerying for ontology languages in the context of the semantic web with particular emphasis over the OBDM problem. Indeed, the general goal of this dissertation is to present a new metamodeling enabling semantics
for OWL 2 QL, study its complexity and devise systems providing practical algorithms for answering metaqueries over OWL 2 QL ontologies including metamodeling features.

In what follows we briefly summarize the main contributions we provide.

• We present the ontology language $\text{Hi(OWL 2 QL)}$, whose syntax is the same as of OWL 2 QL, and we describe its semantics, called MetaModeling semantics (MMS);

• We present a new SPARQL entailment regime allowing one to evaluate a broad range of metaqueries, expressed in SPARQL syntax, over $\text{Hi(OWL 2 QL)}$ ontologies. The class of queries taken into account corresponds to the class of union of conjunctive queries where, beside ABox atoms, also TBox atoms are allowed to occur and where variables may occur in predicate position, that is, in position where typically only constants are allowed to occur;

• We describe the notion of chase for our language, that forms the basis for all decidability and complexity results given in the thesis. We show, by exploiting the chase, that ontology satisfiability for $\text{Hi(OWL 2 QL)}$ has the same complexity as that of ontology satisfiability for OWL 2 QL. Moreover we show that answering instance queries (i.e., metaqueries with ABox atoms only where variables in predicate position may possibly occur) over $\text{Hi(OWL 2 QL)}$ ontologies has the same complexity as of answering standard first-order queries over OWL 2 QL ontologies;

• We study the problem of answering general queries (i.e., metaqueries with both ABox and TBox atoms where variables in predicate position may possibly occur) and we perform an extensive analysis of the computational complexity of such a problem. We present an algorithm, based on metagrounding and called N\textsc{aive}, for the case of a particular class of $\text{Hi(OWL 2 QL)}$ ontologies, called TBox-complete ontologies, where, in a nutshell, one does not have uncertainty about the negative axioms of the ontology, in the sense that for every possible negative statement (such as two classes are disjoint) about classes and relations of the ontology, either such a statement or its negation is logically implied by the ontology. We use the N\textsc{aive} algorithm to show that the computational complexity of answering general queries over TBox complete ontologies is the same as in the case of instance queries. As for the case of non-TBox-complete ontologies, we show that answering general queries is AC\textsc{0} w.r.t. data complexity (i.e., w.r.t. the size of the ABox only), coNP-complete w.r.t. to ontology complexity (i.e., w.r.t. the size of the ontology only) and Π\textsc{p}^2-complete w.r.t. combined complexity (i.e., w.r.t. to both the query and the queried ontology). For the best of our knowledge, these are the first decidability (even tractability w.r.t. data complexity) results for general metaqueries posed to ontologies expressed in (fragments of) OWL 2;

• We present an optimized sound and complete algorithm, called LMG, whose aim is to overcome the computational cost of N\textsc{aive} by replacing the blind instantiation process for metavariables residing behind the metagrounding technique by a more clever strategy that, guided by the structure of the query taken into account, considers only those instantiations that are potentially useful for producing query results;

• We present a prototype system, called MQ-MASTRO, that, for the best of our knowledge, is the first system implementing a sound and complete algorithm to answer a
broad range of metaqueries over ontologies expressed in (fragments of) \( \text{OWL} \, 2 \), and, by means of a series of experiments, we demonstrate that our system can be used in practice and that our query answering technique outperforms metagrounding in virtually all cases.

1.2 Organization of the thesis

The thesis is organized as follows. In Chapters 2, 3 and 4 we present some theoretical background. In particular:

- in Chapter 2 we present the Description Logics \( SROIQ \) and \( DL-Lite_R \), and we discuss some of the issues related to knowledge bases expressed in such languages, with emphasis on the query answering problem;

- in Chapter 3 we introduce the data integration problem and the ontology based data management paradigm;

- in Chapter 4 we introduce the languages \( \text{OWL} \, 2 \) and \( \text{SPARQL} \), that are, the W3C recommendations for respectively representing and querying ontologies in the context of the semantic web.

In Part II we formally present our proposal for a language enabling proper metamodeling and metaquerying in the context of the semantic web. In particular:

- in Chapter 5 we motivate our research by presenting a real world OBDM scenario where metamodeling and metaquerying are necessary in order to respectively formalize and extract newsworthy patterns of the domain of interest. The domain of interest is actually that of the Italian Public Debt, that we have modeled in a joint project with the Italian Ministry of Economy and Finance;

- in Chapter 6 we introduce the language \( \text{Hi(OWL 2 QL)} \), we illustrate metaqueries and present their semantics.

In Part III we address the problem of reasoning in \( \text{Hi(OWL 2 QL)} \) and we provide a number of algorithms solving the considered inference tasks. In particular:

- in Chapter 7 we study the problem of ontology satisfiability and answering instance queries for \( \text{Hi(OWL 2 QL)} \);

- in Chapter 8 we study the problem of answering general metaqueries over \( \text{Hi(OWL 2 QL)} \) ontologies;

- in Chapter 9 we give a complete complexity characterization for the query answering problem for \( \text{Hi(OWL 2 QL)} \).

In Part IV we investigate the practical applicability of our techniques in real world scenarios. In particular:

- in Chapter 10 we present LMG, that is, an optimized algorithm for answering metaqueries and we prove its correctness;
• in Chapter [11] we describe a prototype system, called MQ-MASTRO, that, by implementing the algorithms described in the thesis, allows evaluating metaqueries over $\mathbb{H}i$ (OWL 2 QL) ontologies. Finally, we illustrate the results of various experiments we have carried out using the Italian Public Debt to test the performances of the system.
Chapter 2

Description logics

This chapter introduces the family of knowledge representation languages known as Description Logics (DLs). We describe the syntax and the semantics of the DL SROIQ in Section 2.1 and those of DL-LiteR in Section 2.2. Then in Section 2.3 we present a number of inference problems that are of interest in the context of DLs. Finally we briefly discuss the computational complexity of such problems with reference to SROIQ and DL-LiteR in Sections 2.4 and 2.5 respectively.

2.1 The Description Logic SROIQ

SROIQ [45] is one of the most expressive decidable DL, originally introduced as an extension of the DL SHOIN [46] by adding a number of expressive means such as negated role assertions, complex role inclusions and nominals. Semantics of SROIQ and SHOIN represent the theoretical foundations underpinning logic-based interpretations of W3C standard languages OWL 2 and OWL, respectively.

2.1.1 Syntax of SROIQ

In this section we introduce the syntax of SROIQ. Let $V_C$, $V_R$ and $V_I$ be three pairwise disjoint finite vocabularies of concept, role and individual names, respectively. We assume that both the top and bottom concepts, denoted by $\top$ and $\bot$ respectively, are included in $V_C$, and that $V_R$ contains the universal role, denoted by $U$. The set of SROIQ roles is $V_R \cup \{R^- \mid R \in V_R\}$ where $R^-$ is called the inverse role of $R$. In addition, we define the function $Inv$ such that $Inv(R) = R^-$ and $Inv(R^-) = R$. If $w = R_1 \ldots R_n$ is a string of roles $R_i$ for $1 \leq i \leq n$, then $Inv(w) = Inv(R_1) \ldots Inv(R_n)$. We now give the definitions of role inclusion axiom, role hierarchy, and role assertion.

Definition 1 (Role Inclusion Axiom and Role Hierarchy) A SROIQ role inclusion axiom (RIA) is an expression of the form $w \sqsubseteq R$, where $w$ is a finite string of roles not including $U$ and $R \neq U$ is in $V_R$. A role hierarchy $R_h$ is a finite set of RIAs.

Definition 2 (Role Assertion) If $R \neq U$ and $S \neq U$ are roles, then a SROIQ role assertion is an expression having one of forms $Ref(R)$, $Irr(R)$, $Sym(R)$, $Asy(R)$, $Tra(R)$ and $R \sqsubseteq \neg S$. 
2. Description logics

Given a role hierarchy \( \mathcal{R}_h \) and a set of role assertions \( \mathcal{R}_a \) the set of roles that are *simple* in \( \mathcal{R} = \mathcal{R}_h \cup \mathcal{R}_a \) is inductively defined as follows:

- if \( R \in V_R \) does not occur on the right hand side of a RIA in \( \mathcal{R}_h \), then \( R \) and \( R^{-} \) are simple roles;
- if \( R \in V_R \) occurs on the right hand side of a RIA in \( \mathcal{R}_h \), then \( R \) and \( R^{-} \) are simple roles if, for every \( w \subseteq R \), \( w = S \) and \( S \) is a simple role.

Moreover, \( \mathcal{R}_a \) is *simple* if all roles \( R \) and \( S \) occurring in expressions of the form \( \text{Irr}(R) \), \( \text{Asy}(R) \) and \( R \subseteq \neg S \) in \( \mathcal{R}_a \), are simple in \( \mathcal{R} \).

**Definition 3** (Role Box) A \( SROIQ \) role box (RBox) is a set \( \mathcal{R} = \mathcal{R}_h \cup \mathcal{R}_a \), where \( \mathcal{R}_h \) is a role hierarchy not containing cyclic dependencies and \( \mathcal{R}_a \) is a finite, simple set of role assertions.

Next, we define the set of \( SROIQ \) concepts, denoted by \( \text{Exp}_C \), that can be built over \( V_C, V_R \) and \( V_I \).

**Definition 4** The set of \( SROIQ \) concepts \( \text{Exp}_C \) is the smallest set such that:

- if \( A \in V_C \), then \( A \in \text{Exp}_C \);
- if \( o \in V_I \), then \( \{o\} \in \text{Exp}_C \);
- if \( C, D \in \text{Exp}_C, R \) is a (possibly inverse) role, \( S \) is a (possibly inverse) simple role, and \( n \) is a non-negative integer, then \( C \sqcap D, C \sqcup D, \neg C, \forall R.C, \exists R.C, \exists S.\text{Self}, \geq n.S.C \) and \( \leq n.S.C \) are also in \( \text{Exp}_C \).

Given two concepts \( C, D \in \text{Exp}_C \), a \( SROIQ \) concept inclusion axiom is an expression of the form \( C \sqsubseteq D \). A \( SROIQ \) TBox \( \mathcal{T} \) is obtained as the union of a finite set of concept inclusion axioms and of a \( SROIQ \) RBox. A \( SROIQ \) individual assertion is an expression having one of the forms \( C(a), R(a,b), \neg R(a,B), \) or \( a \neq b \), where \( a, b \in V_I \), \( C \in \text{Exp}_C \) and \( R \) is a (possibly inverse) role. A \( SROIQ \) ABox \( A \) is a finite set of individual assertions.

A \( SROIQ \) knowledge base (KB) \( K \) is a couple \( \langle \mathcal{T}, A \rangle \) where \( \mathcal{T} \) is a TBox and \( A \) is a ABox.

### 2.1.2 Semantics of \( SROIQ \)

In this section we introduce the semantics of \( SROIQ \). In what follows we use (i) \( C \) and \( D \) to denote concepts in \( \text{Exp}_C \), (ii) \( R \) and \( S \) to denote roles, (iii) \( w \) to denote a string of roles, and (iv) \( a \) and \( b \) to denote individual in \( V_I \). As usual in the context of DLs, semantics of \( SROIQ \) relies on the notion of *interpretation*. An interpretation \( \mathcal{I} \) is a couple \( \langle \Delta^I, \mathcal{I} \rangle \) where \( \Delta^I \), called the *domain* of \( \mathcal{I} \), is a non empty set, and \( \mathcal{I} \), called the *interpretation function* of \( \mathcal{I} \), is function mapping every \( A \in V_C \) into a set \( \mathcal{A}^I \subseteq \Delta^I \), every \( R \in V_R \) into a binary relation \( \mathcal{R}^I \subseteq \Delta^I \times \Delta^I \) and every \( o \in V_I \) into an element \( o^I \in \Delta^I \). The interpretation function is extended to constructs of \( SROIQ \) as follows, where \( \#B \) denotes the cardinality of a set \( B \) and \( \circ \) denotes composition of binary relations:

\[
(R^{-})^I = \{ \langle x, y \rangle \mid \langle y, x \rangle \in \mathcal{R}^I \}
\]
2.1 The Description Logic \( \mathbb{SROIQ} \)

\[
U^I = \Delta^I \times \Delta^I \\
(R_1 \ldots R_n)^I = R_1^I \circ \ldots \circ R_n^I \\
\top^I = \Delta^I \\
\bot^I = \emptyset \\
(\neg C)^I = \Delta^I \setminus C^I \\
(C \cap D)^I = C^I \cap C^I \\
(C \cup D)^I = C^I \cup C^I \\
(\exists R.C)^I = \{x \mid \exists y. \langle x, y \rangle \in R^I \text{ and } y \in C^I \} \\
(\forall R.C)^I = \{x \mid \forall y. \langle x, y \rangle \in R^I \text{ implies } y \in C^I \} \\
(\exists S.\text{Self})^I = \{x \mid \langle x, x \rangle \in R^I \} \\
\{(o)^I \} = \{o^I \} \\
(\geq nS.C)^I = \{x \mid \#\{y. \langle x, y \rangle \in R^I \text{ and } y \in C^I \} \geq n \} \\
(\leq nS.C)^I = \{x \mid \#\{y. \langle x, y \rangle \in R^I \text{ and } y \in C^I \} \leq n \}
\]

Satisfaction of (role or concept) inclusion axioms and of (role or individual) assertions w.r.t. an interpretation \( \mathcal{I} \) is defined as follows:

\[
\mathcal{I} \models C \subseteq D, \text{ if } C^I \subseteq D^I \\
\mathcal{I} \models w \subseteq R, \text{ if } w^I \subseteq R^I \\
\mathcal{I} \models \text{Ref}(R), \text{ if } \forall x. x \in \Delta^I, \langle x, x \rangle \in R^I \\
\mathcal{I} \models \text{Irr}(R), \text{ if } \forall x. x \in \Delta^I, \langle x, x \rangle \notin R^I \\
\mathcal{I} \models \text{Sym}(R), \text{ if } \langle x, y \rangle \in R^I \text{ implies } \langle y, x \rangle \in R^I \\
\mathcal{I} \models \text{Asy}(R), \text{ if } \langle x, y \rangle \in R^I \text{ implies } \langle y, x \rangle \notin R^I \\
\mathcal{I} \models R \sqsubseteq \neg S, \text{ if } R^I \cap S^I = \emptyset \\
\mathcal{I} \models C(a), \text{ if } a^I \in C^I \\
\mathcal{I} \models R(a, b), \text{ if } \langle a^I, b^I \rangle \in R^I \\
\mathcal{I} \models \neg R(a, b), \text{ if } \langle a^I, b^I \rangle \notin R^I \\
\mathcal{I} \models a \neq b, \text{ if } a^I \neq b^I
\]

Given a \( \mathbb{SROIQ} \) KB \( \mathcal{K} \), an interpretation \( \mathcal{I} \) is a model of \( \mathcal{K} \) (written \( \mathcal{I} \models \mathcal{K} \)) if \( \mathcal{I} \) satisfies all the inclusion axioms and assertions of \( \mathcal{K} \). A KB is satisfiable if has at least one model, otherwise it is unsatisfiable. A KB \( \mathcal{K} \) logically implies an inclusion axiom (or an assertion) \( \alpha \), (written \( \mathcal{K} \models \alpha \)) if all models of \( \mathcal{K} \) satisfy \( \alpha \).
2. Description logics

2.2 The DL-Lite family of Description Logics

Languages of the DL-Lite family [22] are DLs that have been designed to work in context where the need of managing huge amount of data arise. Indeed, their expressive power have been carefully tailored to ensure that evaluation of a certain class of queries, whose semantics corresponds to that of Select-Project-Join queries of the relational algebra, can be delegated to highly optimized relational database management systems. Among the numerous languages belonging to the DL-Lite family, here we introduce DL-Lite$_R$, as it is the member of the family which the W3C standard language OWL 2 QL was defined on.

2.2.1 Syntax of DL-Lite$_R$

As usual in the context of the DLs, DL-Lite$_R$ allows representing a domain of interest in terms of concepts, denoting sets of objects, and roles, denoting binary relations between objects. Let $V_C$, $V_R$ and $V_I$ be three pairwise disjoint finite vocabularies of concept, role and individual names, respectively. DL-Lite$_R$ concept and role expressions are formed according to the following grammar:

$$
B \rightarrow A \mid \exists R \\
C \rightarrow B \mid \neg B \mid \exists R.B \\
E \rightarrow R \mid \neg R
$$

where $A \in V_C$, $P \in V_R$, and $P^\leftarrow$ denotes the inverse of $P$, $B$ denotes a basic concept, that is, a concepts that can be either an element in $V_C$ or a concept of the form $\exists R$, where $R$ denotes a basic role, that is, a role that is either an element in $V_R$ or its inverse. Finally $C$ denotes a general concept, that is, a concepts that can be either a basic concept or its negation, and $E$ denotes a general role, that is, a role that can be either a basic role or its negation. Given a basic concept $B$ and a general concept $C$, a DL-Lite$_R$ concept inclusion axiom is an expression of the form

$$B \sqsubseteq C.$$

That is, only basic concepts can occur on the left-hand side of inclusion axioms, whereas general concepts can only occur on the right-hand side of inclusion axioms. Note that, as $B_1 \sqcup B_2 \sqsubseteq C$ is equivalent to the pair of axioms $B_1 \sqsubseteq C$ and $B_2 \sqsubseteq C$, DL-Lite$_R$ implicitly includes union of basic concepts in the constructs for the left-hand side of inclusion axioms. Similarly, as $B \sqsubseteq C_1 \cap C_2$ is equivalent to the pair of axioms $B \sqsubseteq C_1$ and $B \sqsubseteq C_2$, DL-Lite$_R$ implicitly includes conjunction of general concepts in the constructs for the right-hand side of inclusion axioms.

Given a basic role $R$ and a general role $E$, a DL-Lite$_R$ role inclusion axiom is an expression of the form

$$R \sqsubseteq E.$$
$$A(o) \quad P(o_1,o_2)$$

where $A \in V_C$, $P \in V_R$, and $o, o_1, o_2 \in V_I$.

A DL-Lite$_R$ ABox is a finite set of DL-Lite$_R$ individual assertions. A DL-Lite$_R$ knowledge base (KB) $K$ is a couple $\langle T, A \rangle$ where $T$ is a TBox and $A$ is an ABox.

### 2.2.2 Semantics of DL-Lite$_R$

As usual in the context of DLs, semantics of DL-Lite$_R$ relies on the notion of interpretation. An interpretation $I$ is a couple $\langle \Delta^I, \cdot^I \rangle$ where $\Delta^I$, called the domain of $I$, is a non empty set, and $\cdot^I$, called the interpretation function of $I$, is function mapping every $A \in V_C$ into a set $A^I \subseteq \Delta^I$, every $R \in V_R$ into a binary relation $R^I \subseteq \Delta^I \times \Delta^I$ and every $o \in V_I$ into an element $o^I \in \Delta^I$.

Note that DL-Lite$_R$ enforces the unique name assumption (UNA) on individuals in $V_I$, that is, for every couple of $o_1, o_2 \in V_I$, if $o_1 \neq o_2$, then $o_1^I \neq o_2^I$. In a nutshell, UNA ensures that every interpretation $I$ must assign to each individual $o$ a distinct element $o^I \in \Delta^I$. The interpretation function is extended to constructs of DL-Lite$_R$ as follows:

$$\begin{align*}
(P^\neg)^I &= \{\langle x, y \rangle \mid \langle y, x \rangle \in R^I \} \\
(\neg B)^I &= \Delta^I \setminus B^I \\
(\neg R)^I &= \Delta^I \times \Delta^I \setminus R^I \\
(\exists R)^I &= \{x \mid \exists y. \langle x, y \rangle \in R^I \} \\
(\exists R.B)^I &= \{x \mid \exists y. \langle x, y \rangle \in R^I \text{ and } y \in B^I \}
\end{align*}$$

Satisfaction of (role or concept) inclusion axioms individual assertions w.r.t. an interpretation $I$ is defined as follows:

$$\begin{align*}
I \models B \sqsubseteq C, \text{ if } B^I \subseteq C^I \\
I \models R \sqsubseteq E, \text{ if } R^I \subseteq E^I \\
I \models A(o), \text{ if } o^I \in A^I \\
I \models P(o_1,o_2), \text{ if } \langle o_1^I, o_2^I \rangle \in R^I
\end{align*}$$

The notions of model, satisfiability and unsatisfiability, and of logical implication of axioms and assertions that has been given for SROIQ KBs in Section 2.1.2 naturally apply as well in the case of DL-Lite$_R$ KBs.

### 2.3 Reasoning in Description Logics

A DL knowledge base $K$ is equipped with a semantics that make it possible to consider $K$ as equivalent to a suitable set of axioms of first-order logic (FOL). So, any other set of FOL axioms, a KB $K$ expressed in a DL language $L$ contains, in addition to the explicit knowledge represented by its axioms, a certain amount of implicit knowledge that can be
made explicit by performing inference processes over $\mathcal{K}$. Clearly, the cost of such inference processes increases with the increase in the expressive power of $\mathcal{L}$, so many efforts have been spent in designing DLs languages having enough expressive power to formalize interesting real-world scenarios while keeping the computational complexity of reasoning as low as possible.

2.3.1 Basic inference problems

The basic inference problem for a DL logic language $\mathcal{L}$ is that of checking satisfiability (SAT) of a given KB $\mathcal{K}$ expressed in $\mathcal{L}$, i.e. determining whether a model of $\mathcal{K}$ exists w.r.t. the semantic of $\mathcal{L}$.

Given a $\mathcal{L}$ knowledge base $\mathcal{K}$, there a number of interesting inference tasks that can be performed over $\mathcal{K}$ [10]:

- **Concept satisfiability**: given a concept $C$ verify whether there exists a model $\mathcal{M}$ of $\mathcal{K}$ such that $C^\mathcal{M} \neq \emptyset$.
- **Concept subsumption**: given two concepts $C$ and $D$ verify whether $\mathcal{K} \models C \subseteq D$.
- **Instance checking**: given an individual $o$ and a concept $C$, verify whether $\mathcal{K} \models C(o)$
- **Role checking**: given two individuals $o_1$ and $o_2$ and a role $R$, $\mathcal{K} \models R(o_1, o_2)$.

Note that, under certain semantics conditions on $\mathcal{L}$, all the above problems can be reduced to SAT. For example, in the case of instance checking, if $\mathcal{L}$ allows for negation in front of individual assertions, one can verify that $\mathcal{K} \models C(o)$ if and only if the KB $\mathcal{K}' = \mathcal{K} \cup \neg C(a)$ is unsatisfiable.

2.3.2 Answering (union of) conjunctive queries

In addition to the inference problems that have been described in the previous section, we are interested in the problem of computing answers to queries posed over a DL knowledge base. In particular, we consider a specific class of queries, namely (union of) conjunctive queries [26], that is equivalent to the class of select-project-join queries of the relational algebra [2].

The notion of (union of) conjunctive queries relies on that of query atom, that in turn is based on the notion of query term.

**Definition 5** (Query term and query atom) Given a KB $\mathcal{K}$, let $\mathcal{U}$ be a countably infinite set of variables disjoint with the vocabularies $V_C$, $V_R$ and $V_I$ of $\mathcal{K}$. A query term (or simply term) is an element in the set $\mathcal{V} \cup V_I$, i.e. it is either a variable or an individual name. A query atom (or simply atom) is an expression of the form $A(t)$ or $P(t_1, t_2)$, where $A$ is a concept name in $V_C$, $P$ is a role name in $V_R$, and $t$, $t_1$ and $t_2$ are terms.

With the definition of terms and atoms in hand, we can now give the definition of conjunctive query.
2.3 Reasoning in Description Logics

Definition 6 (Conjunctive query) Given a KB $K$, a conjunctive query (CQ) over $K$ is an expression of the form

$$q(\bar{x}) \leftarrow \exists \bar{y}. \text{Conj}(\bar{x}, \bar{y})$$

where $\text{Conj}(\bar{x}, \bar{y})$ is a conjunction of atoms built over terms $\bar{x}$, called the distinguished terms of $q$, and $\bar{y}$. A CQ with no distinguished terms is called a boolean CQ.

The definition of union of conjunctive queries is as follows.

Definition 7 (Union of conjunctive queries) Given a KB $K$, a union of conjunctive queries (UCQ) over $K$ is an expression of the form

$$Q(\bar{x}) \leftarrow \bigcup_{1 \leq i \leq n} \exists \bar{y}_i. \text{Conj}_i(\bar{x}, \bar{y}_i)$$

where every $\text{Conj}_i(\bar{x}, \bar{y}_i)$ is a conjunction of atoms built over terms $\bar{x}$, called the distinguished terms of $Q$, and $\bar{y}_i$. A UCQ with no distinguished terms is called a boolean UCQ.

Thus, a UCQ $Q(\bar{x})$ is defined to be the union of a set of CQs $\{q_1(\bar{x}), \ldots, q_n(\bar{x})\}$ with the same distinguished terms $\bar{x}$.

If $q(\bar{x})$ is a conjunctive query with distinguished terms $\bar{x}$, and $\mu$ is a substitution for the variables in $\bar{x}$ with individuals in $V_I$, then we denote by $\mu'$ the function defined as follows

$$\mu'(x) = \begin{cases} x & \text{for } x \in V_I \\ \mu(x) & \text{for } x \in V \end{cases}$$

Furthermore, $q(\bar{x}) \leftarrow \exists \bar{y}. \text{Conj}(\bar{x}, \bar{y})$ is a conjunctive query and $\mu$ is a substitution for the variables in $\bar{x}$, we denote by $\mu'(q)$ the boolean conjunctive query obtained by applying $\mu'$ to $\text{Conj}(\bar{x}, \bar{y})$.

The semantics of conjunctive queries relies on the notion of certain answer, that in turn is based on the well known notion of query homomorphism [26].

Definition 8 (Query homomorphism) Let $K$ be a knowledge base, $q() \leftarrow \exists \bar{y}. \text{Conj}(\bar{y})$ be a boolean conjunctive query over $K$, and let $M = \langle \Delta^M, \cdot^M \rangle$ be a model of $K$. A query homomorphism from $q$ to $M$ is a function $f$ from the terms in $\bar{y}$ to $\Delta^M$ such that for every $o \in \bar{y} \cap V_I$, $f(o) = \sigma^M$ and all the semantics properties that the atoms in $\text{Conj}(\bar{y})$ impose on terms in $\bar{y}$ are preserved under $f$, namely:

- if $A(y) \in \text{Conj}(\bar{y})$, then $f(y) \in A^M$;
- if $P(y_1, y_2) \in \text{Conj}(\bar{y})$, then $\langle f(y_1), f(y_2) \rangle \in P^M$.

Given a knowledge base $K$ and boolean conjunctive query $q$ over $K$, we say that $K$ entails $q$, written $K \models q$, if, for every model $M$ of $K$, there exists a query homomorphism from $q$ to $M$. Furthermore, if $Q$ is a union of boolean conjunctive queries, then any query homomorphism from a disjunct of $Q$ to a model $M$ of $K$ is said to be a query homomorphism from $Q$ to $M$. Given a knowledge base $K$ and a union of boolean conjunctive queries $Q$ over $K$, we say that $K$ entails $Q$, written $K \models Q$, if, for every model $M$ there exists a query homomorphism from $Q$ to $M$. By exploiting the above definition, we can now formally define the notion of certain answer as follows.
Definition 9 (Certain answer) Given a knowledge base $K$ and a union of conjunctive queries $Q$ over $K$ with distinguished terms $\bar{x} = \langle x_1, \ldots, x_m \rangle$ of individuals in $V_I$ is a certain answer of $Q$ over $K$ if and only if for every model $M = \langle \Delta_M, \cdot_M \rangle$ of $K$, there are a substitution $\mu$ such that $\mu'(\bar{x}) = \bar{o}$, and a query $q$ that is a disjunct in $Q$ such that there is a query homomorphism $f$ from $\mu'(q)$ to $M$.

Furthermore, given a knowledge base $K$, and a union of conjunctive queries $Q$ over $K$ with distinguished terms $\bar{x}$, we denote by $\text{ans}(Q, K)$ the set containing all the tuples $\bar{o}$ of individuals occurring in $K$ such that $\bar{o}$ is a certain answer of $Q$ over $K$.

2.4 Complexity of reasoning in $SROIQ$

Here we briefly report the known results about the computational complexity for $SROIQ$ knowledge bases of the reasoning tasks described in Section 2.3.

In [49] has been demonstrated that the problem of checking satisfiability of a $SROIQ$ knowledge base $K$ is N2ExpTime-complete. The computational characterization of other basic inference problems of Section 2.3.1 w.r.t. a $SROIQ$ KB $K$ are given by reduction to (un)satisfiability of a suitable $SROIQ$ KB $K'$ as follows, where $\iota$ denotes a fresh new individuals not occurring in $K$:

- **A concept $C$ is satisfiable w.r.t. $K$ if and only if $K' = K \cup C(\iota)$ is satisfiable. Thus deciding concept satisfiability in $SROIQ$ is N2ExpTime-complete.**

- **Concept subsumption:** given two concepts $C$ and $D$, $K \models C \sqsubseteq D$ if and only if $K' = K \cup \{ C \cap D(\iota) \}$ is unsatisfiable. Thus deciding concept subsumption in $SROIQ$ is CO-N2ExpTime-complete.

- **Instance checking:** given an individual $o$ and a concept $C$, $K \models C(o)$ if and only if $K' = K \cup \{ \neg C(o) \}$ is unsatisfiable. Thus deciding instance checking in $SROIQ$ is CO-N2ExpTime-complete.

- **Role checking:** given two individuals $o_1$ and $o_2$ and a role $R$, $K \models R(o_1, o_2)$ if and only if $K' = K \cup \{ \neg R(o_1, o_2) \}$ is unsatisfiable. Thus deciding role checking in $SROIQ$ is CO-N2ExpTime-complete.

As for query answering, a decision procedure for conjunctive query entailment in $SROIQ$ has not yet been established. The main issue preventing using the hereto known techniques to devise such a decision procedure has been recognized in the possibility in $SROIQ$ to use constructs allowing for the combinations of nominals ($O$), inverse roles ($I$) and cardinality restrictions ($Q$).

However, many efforts has been spent in last years in trying to find suitable techniques for languages whose expressive power is someway comparable to that of $SROIQ$, with the goal to help in solving the issues related to query answering in the DL underlying the OWL 2 W3C standard.

In [37] authors show a correct decision procedure for entailment of UCQs, in the case where non-simple roles are not allowed occurring as predicate in atoms of queries. Though, in such a paper no complexity bounds have been given. In [66] authors show that answering conjunctive queries in Horn-$SROIQ$, i.e. the language obtained from $SROIQ$
by disallowing any form of disjunction, is $2\text{ExpTime}$-complete in combined complexity and $\text{PTime}$-complete in the size of the ABox. Finally, in [24], authors show that answering positive 2-way regular path queries, actually a generalization of CQs, in the DLs $\text{SRIQ}$, $\text{SROI}$ and $\text{SROQ}$ is in $3\text{ExpTime}$ in the total size of the queried knowledge base.

### 2.5 Complexity of reasoning in $\text{DL-Lite}_R$

In this section we present the known results about computational complexity of reasoning in $\text{DL-Lite}_R$. In presenting such results we are interested in data and combined complexity [78], and in TBox complexity:

- **Data complexity** is the complexity w.r.t. the size of the ABox only;
- **TBox complexity** is the complexity w.r.t. the size of the TBox only;
- **Combined complexity** is the complexity w.r.t. the size of all inputs to the problem.

Let us recall the notion of FOL-rewritability for the problems of KB satisfiability and of query answering, as such notion is fundamental to give a precise computational characterization for $\text{DL-Lite}_R$.

First, given a nonempty ABox $\mathcal{A}$ containing only assertions of the form $A(o)$ and $P(o_1,o_2)$, we denote by $db(\mathcal{A}) = \langle \Delta^{db(\mathcal{A})}, o^{db(\mathcal{A})} \rangle$ the interpretation defined as follows:

- $\Delta^{db(\mathcal{A})}$ is the set containing all the individuals occurring in $\mathcal{A}$;
- $o^{db(\mathcal{A})}$ for every individual occurring in $\mathcal{A}$;
- $A^{db(\mathcal{A})} = \{ o \mid A(o) \in \mathcal{A} \}$ for every atomic concept $A$;
- $P^{db(\mathcal{A})} = \{ (o_1,o_2) \mid P(o_1,o_2) \in \mathcal{A} \}$ for every atomic role $P$;

Based on interpretation $db(\mathcal{A})$, the definitions of FOL-rewritability of SAT and query answering are as follows:

**Definition 10** (SAT FOL-rewritability) Satisfiability of KB $\mathcal{K}$ in a DL $\mathcal{L}$ 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 nonempty ABox $\mathcal{A}$, $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable if and only if $q$ evaluates to false in $db(\mathcal{A})$.

**Definition 11** (Query answering FOL-rewritability) Query answering in a DL $\mathcal{L}$ for UCQs is **FOL-rewritable** if, for every UCQ $Q$ and every TBox $\mathcal{T}$ expressed in $\mathcal{L}$, there exists a FOL query $q_1$, over the alphabet of $\mathcal{T}$, such that for every nonempty ABox $\mathcal{A}$, every tuple of individuals $\bar{o}$ occurring in $\mathcal{A}$, $\bar{o} \in \text{ans}(\langle \mathcal{T}, \mathcal{A} \rangle)$, if and only if $\bar{o}$ is answer to $q_1$ w.r.t. the minimal interpretation $db(\mathcal{A})$ under the standard FOL semantics.

In [22] authors show that SAT, instance and role checking, and answering union of conjunctive queries in $\text{DL-Lite}_R$ are FOL-rewritable and hence are all $\text{AC}^0$ in data complexity. They provide a query reformulation procedure, called PERFECTREF, that, given a UCQ $Q$ over a satisfiable KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, returns a UCQ $Q'$ "embedding" the knowledge residing in $\mathcal{T}$ that is relevant for computing the certain answers of $Q$ w.r.t. $\mathcal{K}$. In particular, they
show that \( \text{ans}(Q, \langle T, A \rangle) \) is equivalent to \( \text{ans}(Q', \langle \emptyset, A \rangle) \), thus allowing one to compute the certain answers of \( Q \) w.r.t. \( K \) by computing the answers to \( Q' \) w.r.t. \( db(A) \) under the FOL semantics, that is, by delegating the evaluation of \( Q' \) to a standard relational database management system.

The precise characterization of computational complexity of reasoning problems in \( DL\text{-}\text{Lite}_R \) is as follows:

- Knowledge base satisfiability is \( \text{PTIME} \) in combined complexity and \( \text{AC}^0 \) in data complexity;
- Concept subsumption is \( \text{PTIME} \) in TBox complexity;
- Instance and role checking are both \( \text{PTIME} \) in combined complexity and \( \text{AC}^0 \) in data complexity;
- Answering union of conjunctive queries is \( \text{PTIME} \) in TBox complexity, \( \text{AC}^0 \) in data complexity and \( \text{NP}\)-complete in combined complexity.
Chapter 3

Data integration through OBDM

In this chapter, we introduce the problem of data integration as the problem of granting access to heterogeneous data sources through a shared and unified view. We give a general theoretical definition of the problem in Section 3.1 and we show how DLs can come in handy in solving such a problem in Section 3.2.

3.1 Data integration

Data integration is the problem of providing an unified and transparent view to data residing in numerous, heterogeneous and autonomous sources [56]. Accessing multiple heterogeneous data sources by means of a shared view has been a hot research topic for many years, and has received attention of many researchers, especially from database and knowledge representation areas. From a theoretical perspective, a data integration system $\mathcal{J}$ can be represented as a triple $\langle G, S, M \rangle$ where:

- $G$ is the global schema, that is, a logical theory, expressed in a language $\mathcal{L}_G$ over an alphabet $A_G$, representing the unified and integrated view of the data sources;
- $S$ is the source schema, that is the schema, expressed in a language $\mathcal{L}_S$ over an alphabet $A_S$, describing the structure of the heterogeneous data sources where actual data reside;
- $M$ is the mapping between $S$ and $G$.

The mapping $M$ is formed by a set of mapping assertions, i.e. expressions having one the following forms

\[ q_G \leadsto q_S \]  

\[ q_S \leadsto q_G \]  

where $q_S$, expressed in a query language $\mathcal{L}_{M,S}$, and $q_G$, expressed in a query language $\mathcal{L}_{M,G}$, are two queries of the same arity, respectively over $S$ and over $G$. Assertions of the form (3.1) specifies that the concept of $G$ represented by $q_G$ corresponds to the concept of $S$ represented by $q_S$ (the description of assertions of the form (3.2) can be given similarly).
Intuitively, a data integration system represents a virtual database that users can exploit to extract reconciled data from the sources by posing queries expressed in a language $L_Q$ defined over the alphabet $A_G$.

In what follows, we make the following assumptions:

- if $\alpha$ is a schema, a database $DB$ for $\alpha$ is a set of collections of sets, one for every symbol in the alphabet of $\alpha$;
- if $DB$ is a database for a schema $\alpha$ and $q$ is a query of arity $n$ over $\alpha$, we denote by $q^{DB}$ the set of tuples of arity $n$ in $DB$ that satisfy $q$.

The semantics of a data integration system $J = \langle G, S, M \rangle$ relies on the notions of source database for $J$ and of global database for $J$ where:

- a source database $D$ for $J$ is a database that is instance of $S$;
- a global database $B$ for $J$ is a database that is instance of $G$.

We assume that both the source and the global databases are defined over a fixed domain $\Gamma$. We say that a global database $B$ for $J$ is legal with respect to a source database $D$, if $B$ satisfies all the constraints in $G$, and $B$ satisfies $M$ with respect to $D$, where the notion of satisfaction $M$ w.r.t. $D$ depends on how the assertions in $M$ are interpreted.

Let $J = \langle G, S, M \rangle$ be a data integration system, $D$ be a source database for $J$, and $q$ be a query expressed in the query language $L_Q$ defined over the alphabet $A_G$. The answer $q^{J,D}$ to $q$ over $J$ w.r.t. $D$, is the set of tuples $\vec{t}$ of objects in $\Gamma$ such that $\vec{t} \in q^B$ for every global database $B$ that is legal for $J$ with respect to $D$.

Consequently, computing the answer to queries posed to a data integration system corresponds in solving a logical implication problem: given a tuple $\vec{t}$ of objects in $\Gamma$, $\vec{t}$ is part of the answer to the query if and only if it is logically implied by the data in the sources that $\vec{t}$ satisfies the query.

It is worth noting that one of the fundamental aspect when designing a data integration system, is the specification of the mapping $M$ between $S$ and $G$, as it heavily influence the way answers to queries posed to the system are computed. There exist two basic approaches for the specification of $M$:

- **local-as-view approach (LAV)**: in systems based on LAV approach, $M$ contains a mapping assertion of the form $s \rightarrow q_g$ for every element $s$ of the schema $S$. Such an approach is based on the idea that the content of every element $s$ of $S$ is determined by a view $q_g$ over $G$;
- **global-as-view approach (GAV)**: in systems based on GAV approach, $M$ contains a mapping assertion of the form $g \rightarrow q_s$ for every element $g$ of the schema $G$. Such an approach is based on the idea that the content of every element $g$ of $G$ is determined by a view $q_s$ over $S$.

The choice between LAV and GAV approach is typically made by taking into account what part of the system, between $S$ and $G$, is thought to remain stable over time: a system based on LAV approach is preferable in situations where the global view $G$ is stable , whereas a system based on GAV approach is effective in situations where the stable part of the system is represented by the set of sources which $S$ is based on.
3.2 The OBDM paradigm

The data integration problem must be faced by numerous organizations needing advanced methods and systems for accessing their data. Indeed, nowadays data are often dispersed in heterogeneous and autonomously evolving systems, or have been adapted through the years to the needs of the applications they serve, which makes it difficult to extract them in an useful format for the business of the interested organizations.

In this section we present the Ontology Based data management (OBDM) paradigm [72, 21, 73, 19], which represent an useful tool in many situations where the need arises of accessing a bunch of pre-existing heterogeneous data sources by means of a shared and well understood view. To simplify the presentation, in what follows we assume that data in the sources are available in a single federated relational database. Note that we can make this assumption without loss of generality, as currently there exist numerous data federations tools that can be exploited to get to such a situation.

Intuitively, an OBDM specification is a data integration system where the global schema is given in the form of the intensional level of an ontology, typically a DL knowledge base, which is connected to a relational database through a set of GAV mapping assertions. More formally, an OBDM specification is a triple $\mathcal{J} = \langle \mathcal{T}, \mathcal{D}, \mathcal{M} \rangle$ where:

- $\mathcal{T}$ is a DL TBox, expressed in some language $\mathcal{L}$, representing the explicit formal representation of the domain of interest;
- $\mathcal{D}$ is a (federated) relational database representing the sources;
- $\mathcal{M}$ is a set of GAV mapping assertions of the form $\Phi \leadsto \Psi$, where $\Phi$ is an SQL query over $\mathcal{D}$, and $\Psi$ is a query over the alphabet of $\mathcal{T}$, without existential variables, of the same arity of $\Phi$. Thus $\mathcal{M}$ specifies how data in the sources should be used to populate the elements occurring in $\mathcal{T}$.

Consequently, by applying the mapping $\mathcal{M}$ over the data sources $\mathcal{D}$, we can obtain a set of DL ABox assertions $\mathcal{A}_V$, where the individual constants occurring in such assertions are obtained starting from data values in the sources. Thus, an OBDM specification $\mathcal{J} = \langle \mathcal{T}, \mathcal{D}, \mathcal{M} \rangle$, can be seen as a DL knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}_V \rangle$, where $\mathcal{A}_V = \mathcal{M}(\mathcal{D})$ is called the virtual ABox of $\mathcal{K}$. Note that the ABox $\mathcal{A}_V = \mathcal{M}(\mathcal{D})$ is virtual as it does not exist as an independent syntactic object, rather it is just a representation of the data in $\mathcal{D}$ in terms of the vocabulary of $\mathcal{T}$, that is obtained by applying $\mathcal{M}$ over such data. Note that, in general, there is an impedance mismatch between the way in which data is represented in a relational database, and the way in which the corresponding information must be rendered in a DL knowledge base. Indeed, while the relational database $\mathcal{D}$ stores data values, the ABox $\mathcal{A}_V$ obtained from $\mathcal{D}$ stores instances of DL concepts represented by objects, each one denoted by an object identifier, not to be confused with a data value. The impedance mismatch problem is solved by defining a mapping language that use constructors to obtain objects of the knowledge base from tuples of values in the source database. Such constructors are modeled by defining an alphabet $\Lambda$ of Skolem functions symbols, where each symbol $\lambda$ in $\Lambda$ has an associated arity, specifying the number of arguments it accepts. The set of function symbols in $\Lambda$ are applied to variables occurring in the righ hand side of the mappings assertions. Let $\Phi$ be an SQL query over $\mathcal{D}$, $\Psi$ be a query over the alphabet of $\mathcal{T}$,
a tuple of variables, and \( \bar{\lambda} \) a tuple of Skolem function symbols. The mapping assertions in \( \mathcal{M} \) are expressions of the form

\[
\Phi(\bar{x}) \leadsto \Psi(\bar{\lambda}, \bar{x})
\]

We now give an example of OBDM specification.

Example 12 Let \( \mathcal{T}^{ex} \) be the DL TBox formed by the following axioms

1. \( \text{Employee} \sqsubseteq \text{Person} \)
2. \( \text{Executive} \sqsubseteq \text{Employee} \)
3. \( \text{Employee} \sqsubseteq \exists \text{works in} \)
4. \( \exists \text{works in}^- \sqsubseteq \text{Department} \)
5. \( \text{Executive} \sqsubseteq \exists \text{managed by}^- \)
6. \( \exists \text{managed by}^- \sqsubseteq \text{Department} \)
7. \( \text{Department} \sqsubseteq \exists \text{managed by} \)
8. \( \exists \text{managed by}^- \sqsubseteq \exists \text{Executive} \)
9. \( \text{Department} \sqsubseteq \exists \text{located in} \)
10. \( \exists \text{located in}^- \sqsubseteq \text{City} \)

The above TBox \( \mathcal{T} \) models information about employees and departments they work in. Specifically axioms in \( \mathcal{T} \) state the following. Every employee is a person \( [1] \), and every executive is an employee \( [2] \). Every employee works in at least one department \( [3][4] \). Every executive manages at least one department \( [5][6] \), and every department is managed by at least one executive \( [7][8] \). Finally every department is located in at least one city \( [9][10] \).

Let \( \mathcal{D}^{ex} \) be the relational database formed by the tables \( \text{dept_loc}, \text{dept_emp}, \text{dept_exec} \) that are depicted in Figure 3.1 where

- \( \text{dept_loc} \) stores tuples \( \langle c_1, c_2 \rangle \) where \( c_1 \) is the code of a department located in the city \( c_2 \);
- \( \text{dept_emp} \) stores tuples \( \langle c, s \rangle \) where \( s \) is the social security number of an employee working in the department identified by the code \( c \);
- \( \text{dept_exec} \) stores tuples \( \langle c, s \rangle \) where \( s \) is the social security number of an executive managing the department identified by the code \( c \);

Let \( \text{pers}, \text{dept}, \text{cty} \) be three Skolem function symbols of arity 1, and let \( \mathcal{M}^{ex} \) be the GAV mapping formed by the following assertions
3.2 The OBDM paradigm

M₁: Select code,city  \implies Department(dept(code))  
From dept_loc  
      City(cty(city))  
      located_in(dept(code),cty(city))

M₂: Select code,SSN  \implies Employee(pers(SSN))  
From dept_emp  
      Department(dept(code))  
      works_in(pers(SSN),dept(code))

M₃: Select code,SSN  \implies Executive(pers(SSN))  
From dept_exec  
      Department(dept(code))  
      managed_by(dept(code),pers(SSN))

M₁ maps every tuple \(⟨s,n⟩\) in employee to an employee \(\text{pers}(s)\). M₂ maps every 
tuple \(⟨c,s⟩\) in dept_emp to an employee \(\text{pers}(s)\) working in the department \(\text{dept}(c)\). M₂ 
maps every tuple \(⟨c,s⟩\) in dept_exec a department \(\text{dept}(c)\) which is managed by the 
executive \(\text{pers}(s)\).

Let \(J^{ex} = (T^{ex}, D^{ex}, M^{ex})\) be an OBDM specification. Given a mapping assertion 
\(M \in \mathcal{M}^{ex}\), we denote by \(M(D^{ex})\) the set of individual assertions that is possible to obtain 
by the application of \(M\) over \(D^{ex}\). We now show how the virtual ABox \(A^{ex}_{V}\) can be obtained 
by applying \(M^{ex}\) over \(D^{ex}\). \(A^{ex}_{V}\) is the union of the following set of individual assertions

\[
\begin{align*}
M₁(D^{ex}) &= \{ \text{Department}(\text{dept}(d_1)), \text{Department}(\text{dept}(d_2)), \\
&\quad \text{located_in}(\text{dept}(d_1),\text{cty}(\text{Rome})), \text{City}(\text{cty}(\text{Rome})), \\
&\quad \text{located_in}(\text{dept}(d_2),\text{cty}(\text{Milan})), \text{City}(\text{cty}(\text{Milan})) \}; \\
M₂(D^{ex}) &= \{ \text{Employee}(\text{pers}(1155)), \text{Employee}(\text{pers}(2266)), \\
&\quad \text{works_in}(\text{pers}(1155),\text{dept}(d_1)), \text{Department}(\text{dept}(d_1)), \\
&\quad \text{works_in}(\text{pers}(2266),\text{dept}(d_2)), \text{Department}(\text{dept}(d_2)) \}; \\
M₂(D^{ex}) &= \{ \text{Executive}(\text{pers}(3377)), \text{Executive}(\text{pers}(5599)), \\
&\quad \text{managed_by}(\text{dept}(d_1),\text{pers}(3377)), \text{Department}(\text{dept}(d_1)), \\
&\quad \text{managed_by}(\text{dept}(d_3),\text{pers}(5599)), \text{Department}(\text{dept}(d_3)) \};
\end{align*}
\]

The main reason to build an OBDM system is to provide high-level services to the clients 
of the information system. The most important service is query answering. Clients express 
their queries in terms of the conceptual view (the DL knowledge base) in the form of UCQs, 
and the system should reason about the knowledge base and the mapping and should translate
the request into suitable queries posed to the sources. Given that the amount of data stored at
the sources can be very large, OBDM systems must rely on DLs for which reasoning remains
computationally tractable with respect to data complexity. This consideration rules out the
possibility of using, for example, the DL SROIQ, as the problem of entailment of CQs in
such a language is not even known to be decidable, or any language allowing to express a
form of disjunction, as the problem of answering conjunctive queries in such languages is
known to be coNP-hard w.r.t. data complexity \[64, 65, 36\]. In literature there exist a number
of DLs which guarantee that reasoning problems are decidable in polynomial time, as, for
example, EL \[9\], Horn-SH\(I\)\(Q \[47\], and the languages of the DL-Lite family introduced
in Section 2.2. However, only the languages of the DL-Lite family allow for answering
UCQs in AC\(0\) with respect to data complexity, and, consequently, for delegating query
processing to the relational DBMS managing the data layer thanks to FOL-rewritability of
the query answering problem. Indeed, let \(J = \langle T, D, M \rangle\) be an OBDM specification such
that \(T\) is a DL-Lite\(R\) TBox, and let \(Q\) be an UCQ defined over the vocabulary of \(T\). In \[72\],
authors show that the certain answers of \(Q\) w.r.t. \(J\) can be computed by executing three
main steps:

1. \(Q\) is rewritten with respect to \(T\) into a new UCQ \(Q'\). \(Q'\) embed all the knowledge
   residing in \(T\) that is necessary to compute the certain answer to \(Q\);
2. the mapping assertions in \(M\) are used to unfold the UCQ \(Q'\) into an SQL query \(Q''\);
3. the answers obtained by evaluating the SQL query \(Q''\) over \(D\) are returned as the
certain answers of \(Q\) w.r.t. \(J\).
Chapter 4

OWL 2

The Web Ontology Language (OWL 2) is a W3C recommendation and the de facto standard language for the definition of ontologies in the context of the semantic web. We start by introducing the Resource Description Framework, which is the standard model for data interchange on the web and one of the fundamental block of the semantic web tower, in Section 4.1. In Section 4.2 we describe the syntax of OWL 2, and in Section 4.3 we introduce the two semantics under which is possible to interpret OWL 2 ontologies. In Section 4.4 we present the language OWL 2 QL as a fragment of OWL 2 particularly suited for data intensive scenarios, and we show how the reasoning techniques devised for DL-Lite^R can be adapted to the case of OWL 2 QL. Finally, in Section 4.5 we introduce (a subset) of the SPARQL 1.1 query language, that is, the W3C standard for query answering over OWL 2 ontologies.

4.1 The Resource Description Framework

The Resource Description Framework (RDF) allows one to describe a domain of interest as a set of RDF triples. Each "thing" in the domain (objects, sets of objects, data values, datatypes, etc.,...) is called a resource and can be denoted by either an international resource identifier (IRI)\(^1\) or a literal. An IRI is a character string designed to identify resources in the context of the semantic web and typically presents a structure of type NameSp-Res_ID, where NameSp, called namespace of the IRI, is a string which fixes a common context for all IRIs sharing it (similar to a common sub-URL for different web pages hosted on the same server), and Res_ID is a string associated to a particular object in that context. A namespace may have an associated namespace prefix which is a short string that can be used in practice to make the IRI more readable. Literals are represented by a pair \((l,d)\), where \(l\) is a string and \(d\) is an IRI.

Let Iri denote the set of all international resource identifiers, \(L_{rdf}\) denote the set of all the RDF literals, \(B_{rdf}\) denote the set of all blank nodes (strings disjoint from both Iri and \(L_{rdf}\)), and let Terms_{rdf} denote the union of of the three previous sets. We refer to the elements of Terms_{rdf} as the RDF terms.

A RDF triple is a statement of the form \{subject predicate object\} where:

- subject \(\in\) Iri \(\cup\) \(B_{rdf}\);
- predicate \(\in\) Iri;

\(^1\)http://www.ietf.org/rfc/rfc3987.txt
26

• object ∈ Terms_{\text{rdf}}.

An RDF triple is \textit{ground} if no blank nodes occur in it. A \textit{RDF graph} is a set of RDF triples. An RDF graph is \textit{ground} if all its triples are ground. Given an RDF graph $G$, the vocabulary $V_G$ of $G$ is the set of all the RDF terms that occur in $G$.

According to the W3C standards, RDF graphs can be interpreted in numerous ways, based on different sets of semantic conditions that are taken into account to define the notion of \textit{satisfaction} of a given graph. Starting from the most simple type of interpretation, not assigning predefined meaning to any term that may occur in a graph, each type of interpretation is obtained as a monotonic extension of the previous one, by progressively extending the semantic conditions to be satisfied.

\subsection{Simple semantics}

An \textit{RDF simple interpretation} is a 6-tuple $I = (U, \mathcal{P}, \cdot^{\text{Ext}}, \cdot^{\text{IS}}, \cdot^{\text{IL}})$ where:

\begin{itemize}
  \item $U$ is a non-empty set of resources, called the \textit{domain} of $I$;
  \item $\mathcal{P}$ is a set resources, called the \textit{set of properties} of $I$;
  \item $\cdot^{\text{Ext}} : \mathcal{P} \rightarrow U \times U$;
  \item $\cdot^{\text{IS}} : \text{Iri} \rightarrow U \cup \mathcal{P}$;
  \item $\cdot^{\text{IL}} : \mathcal{L}_{\text{rdf}} \rightarrow U$;
\end{itemize}

A simple interpretation $I$ interprets IRIs and literals and assigns truth values to ground triples and graph according to the following semantic conditions:

\begin{itemize}
  \item if $e \in \mathcal{L}_{\text{rdf}}$, then $I(e) = (e)^{\text{IL}}$;
  \item if $e \in \text{Iri}$, then $I(e) = (e)^{\text{IS}}$;
  \item if $e = s \ p \ o$ is a ground triple, then
    \begin{itemize}
      \item if $I(p) \in \mathcal{P}$ and $\langle I(s), I(o) \rangle \in I(p)^{\text{Ext}}$, $I(e) = \text{true}$;
      \item $I(e) = \text{false}$ otherwise.
    \end{itemize}
  \item if $e$ is a ground graph, then
    \begin{itemize}
      \item if $e$ contains a triple $e'$ s.t. $I(e') = \text{false}$, then $I(e) = \text{false}$;
      \item $I(e) = \text{true}$ otherwise.
    \end{itemize}
\end{itemize}

To interpret non-ground graphs we need manage the presence of blank nodes, that are treated as simply indicating the existence of something that is not precisely identified, similar to labeled nulls in database theory \cite{29}. Given a graph $G$, let $\mathcal{BL}(G)$ be the set of blank nodes occurring in $G$ and let $sk$ be a mapping from $\mathcal{BL}(G)$ to $U$, and let $I^{sk}$ be the extension of the simple interpretation $I$ such that $I^{sk}(e) = sk(e)$, for all $e$ in $\mathcal{BL}(G)$. The semantic conditions considering blank nodes are as follows:

\begin{itemize}
  \item if $e$ is a graph, then
\end{itemize}
4.1 The Resource Description Framework

- if there exist some mapping sk such that $I^{sk}(e) = \text{true}$, then $I(e) = \text{true}$;
- $I(e) = \text{false}$ otherwise.

Given two graph $G$ and $G'$ and a simple interpretation $I$ we say that:

1. $I$ satisfies $G$ if $I(G) = \text{true}$;
2. $G$ is simply satisfiable if there exists some simple interpretation satisfying it;
3. $G$ entails $G'$ if all the simple interpretations satisfying $G$ also satisfies $G'$.

All semantics extensions of the simple interpretation use datatypes and literals to represent data values. Each literal is a pair $(l, d)$ such that $l$ is a string, called lexical form and $d$ is an IRI identifying a datatype. Each datatype $d$ is defined by an associated set of lexical forms (i.e., a set of character strings), that define the lexical space of $d$, a non-empty set called the value space and a mapping $\cdot^d$ from its lexical space to its value space.

**Definition 13** Given a set $\mathcal{D}$ of IRIs denoting datatypes, a $\mathcal{D}$-interpretation is a simple interpretation $I$ satisfying the following semantic conditions:

- for every IRI $d \in \mathcal{D}$, $I(d)$ coincides with the datatype denoted by $d$;
- $\forall (l, d) \mid d \in \mathcal{D}, (l, d)^{IL} = (l)^d$.

Given a set $\mathcal{D}'$ of IRIs denoting datatypes, a RDF datatype map is the restriction of a $\mathcal{D}$-interpretation to the set $\mathcal{D}'$. In a nutshell, a RDF datatype map defines a global and unique mapping from the set of IRIs to datatypes and to the set of literals into data values.

Each semantic extension $E$ of the simple semantics (e.g. RDF, RDFS, OWL 2 RDF-Based) is defined over a vocabulary $V_E$ containing an infinite set of IRIs with predefined semantics and, to impose the desired meaning to such a vocabulary, imposes a set of semantic conditions for IRIs in $V_E$ that a $E$-interpretation has to satisfy.

### 4.1.2 RDF semantics

The first semantic extension is called RDF semantics and adds semantics conditions on the set $V_{\text{RDF}}$ containing part of the infinite set of IRIs with prefix rdf including for example rdf:type and rdf:Property. Given a set of datatypes $\mathcal{D}$ including xsd:string, an RDF interpretation is a $\mathcal{D}$-interpretation satisfying an infinite set of axiomatic triples such as the triple $\text{rdf:type } \text{rdf:type } \text{rdf:Property}$ stating that $\text{rdf:type}$ is an instance of $\text{rdf:Property}$, and the following semantic conditions:

- $x \in P$ if and only if $(x, I(\text{rdf:Property})) \in I(\text{rdf:type})^{Ext}$;
- for every $d \in \mathcal{D}$, $(x, I(d)) \in I(\text{rdf:type})$ if and only if $x$ is in the value space of $I(d)$.

\[https://www.w3.org/TR/rdf11-mt/\]
4.1.3 RDFS semantics

So far we have seen that simple semantics can only interpret the basic structure of a graph, whereas RDF semantic conditions can only induce a very limited notion of entailment. More complex forms of entailment are enabled by the RDFS semantics. RDFS semantics introduces the notion of class as a resource representing a set of objects in the universe and considers a vocabulary $V_{RDFS}$ that extends $V_{RDF}$. A subset of the IRIs having a predefined semantics is reported in Table 4.1. An RDFS interpretation is an RDF interpretation, extended by adding a function $\cdot^C$ and two sets $C$ and $V$, satisfying, among the others, the triples in Table 4.2 and the following semantic conditions:

- $(C)^C = \{ x | \langle x, e \rangle \in (\mathcal{I}(\text{rdf:type}))^\text{Ext} \}$;
- $C = (\mathcal{I}(\text{rdfs:Class}))^C$;
- $V = (\mathcal{I}(\text{rdfs:Literal}))^C$;
- $U = (\mathcal{I}(\text{rdfs:Resource}))^C$;
- if $\langle x, y \rangle \in (\mathcal{I}(\text{rdfs:domain}))^\text{Ext}$ and $\langle z, w \rangle \in (x)^\text{Ext}$, then $z \in (y)^C$;
- if $\langle x, y \rangle \in (\mathcal{I}(\text{rdfs:range}))^\text{Ext}$ and $\langle z, w \rangle \in (y)^\text{Ext}$, then $w \in (y)^C$;
- $(\mathcal{I}(\text{rdfs:subPropertyOf}))^\text{Ext}$ is transitive and reflexive on $P$;
- if $\langle x, y \rangle \in (\mathcal{I}(\text{rdfs:subPropertyOf}))^\text{Ext}$, then $x, y \in P$ and $(x)^\text{Ext} \subseteq (y)^\text{Ext}$;
- $(\mathcal{I}(\text{rdfs:subClassOf}))^\text{Ext}$ is transitive and reflexive on $C$;
- if $x \in C$, then $\langle x, \mathcal{I}(\text{rdfs:subPropertyOf}) \rangle \in (\mathcal{I}(\text{rdfs:subClassOf}))^\text{Ext}$;
- if $\langle x, y \rangle \in (\mathcal{I}(\text{rdfs:subClassOf}))^\text{Ext}$, then $x, y \in C$ and $(x)^C \subseteq (y)^C$;
- if $x \in \text{rdfs:Datatype}$, then $\langle x, \mathcal{I}(\text{rdfs:Literal}) \rangle \in (\mathcal{I}(\text{rdfs:subClassOf}))^\text{Ext}$.

<table>
<thead>
<tr>
<th>rdfs:subClassOf</th>
<th>rdfs:subPropertyOf</th>
<th>rdfs:Class</th>
<th>rdfs:domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs:range</td>
<td>rdfs:Resource</td>
<td>rdfs:Datatype</td>
<td>rdfs:Literal</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 4.1. RDFS vocabulary

<table>
<thead>
<tr>
<th>rdf:type</th>
<th>rdfs:domain</th>
<th>rdfs:Resource</th>
<th>rdfs:domain</th>
<th>rdfs:domain</th>
<th>rdf:Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs:range</td>
<td>rdfs:domain</td>
<td>rdf:Property</td>
<td>rdfs:subPropertyOf</td>
<td>rdfs:domain</td>
<td>rdf:Property</td>
</tr>
<tr>
<td>rdfs:subClassOf</td>
<td>rdfs:domain</td>
<td>rdfs:Class</td>
<td>rdfs:type</td>
<td>rdfs:range</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>rdfs:domain</td>
<td>rdfs:range</td>
<td>rdfs:Class</td>
<td>rdfs:range</td>
<td>rdfs:range</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>rdfs:subPropertyOf</td>
<td>rdfs:range</td>
<td>rdf:Property</td>
<td>rdfs:subClassOf</td>
<td>rdfs:range</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 4.2. RDFS axiomatic triples
An OWL 2 ontology is a formal description of a domain of interest and can be defined by using four distinct syntactic categories:

- **Literals**, that, as in RDF, are used to denote data values;
- **Entities**, also known as *atomic expressions*, that are the fundamental elements of an ontology;
- **Complex expressions** representing complex elements in the domain of interest;
- **Axioms** representing statements that are asserted to be true in the domain of interest.

In Sections 4.2.1, 4.2.2 and 4.2.3 we describe entities, complex expressions and axioms respectively. Note that syntax of literals will be given later together with their semantics. Finally, in Section 4.2.4 we list the syntactic restrictions that has to be satisfied by a OWL 2 ontology to be declared as a OWL 2 DL ontology. Note that there exist several syntaxes that can be used to define an OWL 2 ontology. In what follows we describe the OWL 2 *functional-style syntax* [15].

### 4.2.1 OWL 2 entities

The fundamental building blocks of an OWL 2 ontology are the *entities* defining its signature. Classes, datatypes, object properties, data properties and named individuals are entities, and they are all uniquely identified by IRIs. Classes represent sets of individuals; datatypes represent sets of data values such as strings or integers; object and data properties can be used to represent relationships in the domain between pairs of individuals and between an individual and a literal, respectively; named individuals represent actual objects in the domain. Apart from named individuals, OWL 2 also provides for anonymous individuals that, similarly to blank nodes in RDF, can be used locally in one ontology to represent objects of the domain that are known to exist but for which an identifier is not known.

Entities are identified by using IRIs. Subsets of IRIs of particular interest to us are:

- $R_{owl}$, containing the IRIs in the reserved vocabulary of OWL 2 (*i.e.*, the set of all IRIs with prefixes: :rdf, :rdfs, :xsd and :owl);

- $D_{owl}$, that is the subset of $R_{owl}$ containing the IRIs listed in Table 4.3;

- $T_{owl}$, that is the subset of $R_{owl}$ containing IRIs owl:Thing, owl:Nothing, owl:topObjectProperty, owl:bottomObjectProperty, owl:topDataProperty, and owl:bottomDataProperty;

- $D'_{owl}$, which is defined as $D_{owl} \setminus \{\text{rdfs:Literal}\}$.

Note that in general an IRI can be used in the same context to refer to more than one type of entity. This native syntactic support for metamodeling in OWL 2 is called *punning*. 
### 4.2.2 OWL 2 expressions

**OWL 2 expressions** are built upon entities and can be split into class expressions, data range expressions, object property expressions and data properties. A class expression is either a class or a complex class expression. An object property expression is either an object property or a complex object property expression. A data range is either a datatype or a complex data range expression. Complex class expressions are listed in Table 4.4 where: (i) \( ce \) (possibly with subscript) denotes a class expression; (ii) \( ind \) (possibly with subscript) denotes an individual; (iii) \( ope \) denotes an object property expression; (iv) \( dp \) denotes a data property; (v) \( lt \) denotes a literal; (vi) \( k \) denotes a non negative integer. Complex object property expressions are expressions of the form \( \text{ObjectInverseOf}(op) \), where \( op \) denotes an object property.

<table>
<thead>
<tr>
<th>Complex class expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectIntersectionOf(( ce_1 ) ( ce_2 ) ... ( ce_n ))</td>
</tr>
<tr>
<td>ObjectComplementOf(( ce ))</td>
</tr>
<tr>
<td>ObjectSomeValuesFrom(( ope ) ( ce ))</td>
</tr>
<tr>
<td>ObjectHasSelf(( ope ))</td>
</tr>
<tr>
<td>ObjectMinCardinality(( k ) ( ope ) ( ce ))</td>
</tr>
<tr>
<td>ObjectMaxCardinality(( k ) ( ope ) ( ce ))</td>
</tr>
<tr>
<td>DataAllValuesFrom(( dp ) ( dr ))</td>
</tr>
<tr>
<td>DataHasValue(( dp ) ( lt ))</td>
</tr>
<tr>
<td>DataMaxCardinality(( k ) ( dp ) ( dr ))</td>
</tr>
<tr>
<td>DataMinCardinality(( k ) ( dp ) ( dr ))</td>
</tr>
<tr>
<td>DataExactCardinality(( k ) ( dp ) ( dr ))</td>
</tr>
</tbody>
</table>

| Table 4.4. OWL 2 Complex class expressions |

Complex data range expressions are listed in Table 4.5 where: (i) \( dr \) (possibly with subscript) denotes a data range; (ii) \( lt \) (possibly with subscript) denotes a literal; (iii) \( dt \) is a datatype; (iv) \( F_i \) denotes a constraining facet.

<table>
<thead>
<tr>
<th>Complex data range expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataIntersectionOf(( dr_1 ) ( dr_2 ) ... ( dr_n ))</td>
</tr>
<tr>
<td>DataOneOf(( lt_1 ) ( lt_2 ) ... ( lt_n ))</td>
</tr>
<tr>
<td>DatatypeRestriction(( dt F_1 ) ( lt_1 ) ... ( F_n ) ( lt_n ))</td>
</tr>
</tbody>
</table>

| Table 4.5. OWL 2 Complex data range expressions |

The structure of complex expressions naturally induces to define the notion of type of position of entities in complex expression.

**Definition 14** (Type of position for entities in complex expressions) Given an entity \( e \) and an expression \( E \), we say that:
• e occurs in class position in E if E has one of the forms in Table 4.4 and e occurs in E in a position denoted by ce (possibly with subscript);
• e occurs in object property position in E if either E has one of the forms in Table 4.4 and e occurs in E in a position denoted by ope, or E=ObjectInverseOf(e);
• e occurs in data property position in E if E has one of the forms in Table 4.4 and e occurs in E in a position denoted by dp;
• e occurs in individual position in E if E has one of the forms in Table 4.4 and e occurs in E in a position denoted by ind (possibly with subscript);
• e occurs in data range position in E if E has either one of the forms in Table 4.4 or one in Table 4.5 and e occurs in E in a position denoted by dr (possibly with subscript).

4.2.3 OWL 2 axioms

An OWL 2 ontology is a finite set of OWL 2 axioms. Axioms can be partitioned into declaration axioms and logical axioms. Declaration axioms are listed in Table 4.6. Declaration axioms are used to indicate that a given entity e is part of the signature of the considered ontology and to associate it with an entity type, i.e., declaration axioms states whether e is used in the as a class, a datatype, an object (resp. data) property an individual or a combination thereof in the considered ontology.

Definition 15 (Declaration of entities) Let Ax be an axiom having one of the forms Table 4.6. We say that an entity e
• is declared as a class by Ax, if Ax = Declaration(Class(e));
• is declared as an object property by Ax, if Ax=Declaration(ObjectProperty(e));
• is declared as data property by Ax, if Ax = Declaration(DataProperty(e));
• is declared as a datatype by Ax, if Ax = Declaration(Datatype(e));
• is declared as an individual by Ax, if Ax = Declaration(NamedIndividual(e)).

Logical axioms provide a mean to formally describe relationships occurring between elements of the domain and are divided in ABox axioms and TBox axioms. Logical axioms are listed in Tables 4.7 and 4.7 where (i) c denotes a class; (ii) ce (possibly with subscript) denotes a class expression; (iii) ind (possibly with subscript) denotes an individual; (iv) ope (possibly with subscript) denotes an object property expression; (v) dp (possibly with subscript) denotes a data property; (vi) lt denotes a literal; (vii) dr denotes a data range expression; (viii) dt denotes a datatype.

<table>
<thead>
<tr>
<th>Declaration(Class(e))</th>
<th>Declaration(ObjectProperty(e))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration(DataProperty(e))</td>
<td>Declaration(Datatype(e))</td>
</tr>
<tr>
<td>Declaration(NamedIndividual(e))</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.6. OWL 2 Declaration axioms

As in the case of complex expressions, the structure of axioms naturally induces to define the notion of type of position of entities in axioms.
Table 4.7. OWL 2 ABox axioms

<table>
<thead>
<tr>
<th>ABox Axiom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DifferentIndividuals(ind_1 ... ind_n)</td>
<td>SubClassOf(ce_1 ce_2)</td>
</tr>
<tr>
<td>SubObjectPropertyOf(dp_1 dp_2)</td>
<td>SameIndividual(ind_1 ... ind_n)</td>
</tr>
<tr>
<td>ObjectPropertyDomain(ope ce)</td>
<td>ObjectPropertyRange(ope ce)</td>
</tr>
<tr>
<td>DataPropertyDomain(dp ce)</td>
<td>DataPropertyRange(dp dr)</td>
</tr>
<tr>
<td>DisjointUnion(ce_1 ... ce_n)</td>
<td>DisjointClasses(ce_1 ... ce_n)</td>
</tr>
<tr>
<td>DisjointObjectProperties(ope_1 ... ope_n)</td>
<td>DisjointDataProperties(dp_1 ... dp_n)</td>
</tr>
<tr>
<td>EquivalentClasses(ce_1 ... ce_n)</td>
<td>EquivalentObjectProperties(ope_1 ... ope_n)</td>
</tr>
<tr>
<td>IrreflexiveObjectProperty(ope)</td>
<td>SubObjectPropertyOf(ope_1 ope_2)</td>
</tr>
<tr>
<td>SymmetricObjectProperty(ope)</td>
<td>AsymmetricObjectProperty(ope)</td>
</tr>
<tr>
<td>InverseObjectProperties(ope_1 ope_2)</td>
<td>FunctionalObjectProperty(ope)</td>
</tr>
<tr>
<td>TransitiveObjectProperty(ope)</td>
<td>InverseFunctionalObjectProperty(ope)</td>
</tr>
<tr>
<td>FunctionalDataProperty(dp)</td>
<td>HasKey(ce (ope_1 ... ope_n) (dp_1 ... dp_n))</td>
</tr>
<tr>
<td>EquivalentDataProperties(dp_1 ... dp_n)</td>
<td>ReflexiveObjectProperty(ope)</td>
</tr>
<tr>
<td>DatatypeDefinition(dt dr)</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.8. OWL 2 TBox axioms

Definition 16 (Type of position for entities in axioms) Given an entity \( e \) and an axiom \( Ax \), we say that:

- \( e \) occurs in class position in \( Ax \) if either \( e \) is declared as a class by \( Ax \), or \( Ax \) has one of the forms in Tables 4.7 and 4.8 and \( e \) either occurs in \( Ax \) in a position denoted by \( ce \) (possibly with subscript), or it occurs in class position in some of the arguments of \( Ax \);

- \( e \) occurs in object property position in \( Ax \) if either is declared as an object property by \( Ax \), or \( Ax \) has one of the forms in Tables 4.7 and 4.8 and \( e \) either occurs in \( Ax \) in a position denoted by \( ope \), or it occurs in object property position in some of the arguments of \( Ax \);

- \( e \) occurs in data property position in \( Ax \) if either is declared as an data property by \( Ax \), or \( Ax \) has one of the forms in Tables 4.7 and 4.8 and \( e \) either occurs in \( Ax \) in a position denoted by \( dp \) (possibly with subscript) or it occurs in data property position in some of the arguments of \( Ax \);

- \( e \) occurs in individual position in \( Ax \) if either is declared as an individual by \( Ax \), or \( Ax \) has one of the forms in Tables 4.7 and 4.8 and \( e \) either occurs in \( Ax \) in a position denoted by \( ind \) (possibly with subscript), or it occurs in individual position in some of the arguments of \( Ax \);

- \( e \) occurs in data range position in \( Ax \) if either is declared as a datatype by \( Ax \), or \( Ax \) has one of the forms in Tables 4.7 and 4.8 and \( e \) either occurs in \( Ax \) in a position denoted by either \( dr \) or \( dt \), or it occurs in data range position in some of the arguments of \( Ax \).

Furthermore, given an OWL 2 ontology \( O \) and an entity \( e \), we say that \( e \) occurs in class (resp. object property, data property, datatype) position in \( O \) if \( e \) occurs in class (resp. object
property, data property, datatype) position in some of the axioms of $O$.

### 4.2.4 OWL 2 DL syntactic restrictions

Ontologies in OWL 2 DL represent the subset of OWL 2 ontologies for which it is possible to use reasoning techniques that have been designed in the context of DLs. Intuitively, the satisfaction of the OWL 2 DL syntactic restrictions from an OWL 2 ontology $O$, ensures that $O$ can be unambiguously mapped to a corresponding SROIQ(D) knowledge base $K_O$, where SROIQ(D) is the language obtained by adding to SROIQ the support for (a restricted form) of concrete domains [11]. More formally, to be an OWL 2 DL ontology, a OWL 2 ontology $O$ has to be such that:

- no entity in $\mathcal{R}_{owl}$ other than owl:Thing and owl:Nothing occurs in class position in $O$;
- no entity in $\mathcal{R}_{owl}$ other than owl:topObjectProperty and owl:bottomObjectProperty occurs in object property position in $O$;
- no entity in $\mathcal{R}_{owl}$ other than owl:topDataProperty and owl:bottomDataProperty occurs data property position in $O$;
- no entity in $\mathcal{R}_{owl}$, other than those in $\mathcal{D}_{owl}$, occurs in data range position in $O$;
- no entity in $\mathcal{R}_{owl}$ occurs in individual position in $O$;
- all the entities occurring as object properties in axioms of the form ObjectMinCardinality, ObjectMaxCardinality and ObjectExactCardinality (i.e., axioms imposing number restrictions) are simple. Where the notion of simple object property can be straightforwardly derived from notion of simple roles given in Section 2.1.1 for SROIQ KBs;
- does not exist any entity $e$ occurring both in object property position and in data property position in $O$;
- does not exist any entity $e$ occurring both in class position and in data range position in $O$.

The last two conditions listed above are called typing constraints of OWL 2 DL.

### 4.3 OWL 2 semantics

In this section we introduce the semantics of OWL 2. There exist two distinct and alternative ways to assign formal meaning to an OWL 2 ontology:

- by using the Direct Semantics, that interprets directly the ontology structure and is based on the model theoretic semantics of the description logic SROIQ introduced in Section 2.1;
- RDF-Based Semantics, that is defined as an extension of the RDFS semantics described in Section 4.1.3 and interprets the RDF graph which the ontology can be mapped to [69].
There exist several fundamental differences between the two semantics, but the notions of entailment for both of them are strongly connected by a relationship stating that the RDF-Based Semantics is able to reflect all the logical conclusions which can be derived by considering the Direct Semantics \[34\]. As the Direct Semantics can be used to interpret only a proper subset of the OWL 2 ontologies that can be interpreted using the RDF-Based Semantics, i.e. OWL 2 DL ontologies only, the Direct Semantics can be considered as a semantic subset of the RDF-Based Semantics only when OWL 2 DL ontologies are interpreted. However, as the RDF-Based Semantics is a non-standard semantics, and we are interested only in semantics being model theoretic and logic-based, we fully describe the Direct Semantics in Section 4.3.1 whereas we just give a brief description of RDF-Based one in Section 4.3.2.

4.3.1 Direct Semantics

In this section we present the OWL 2 direct semantics (DS). Such a semantics can be used to interpret only OWL 2 DL ontologies, and it is defined as an extension of the semantics of SROIQ that has been given in Section 2.1.2. Indeed, for many of the OWL 2 constructs that have been described in Section 4.2, one can find a natural counterpart among the constructs of SROIQ that have been described in Section 4.2. However, as SROIQ does not provide support for punning, data properties and datatypes, the OWL 2 DS is here described fully instead of by reference to semantics of SROIQ.

First of all we introduce the notion of OWL 2 datatype map, whose aim, intuitively, is to fix the interpretation of OWL 2 datatypes. Like RDF, OWL 2 uses literals to represent data values, where each literal is a pair \((l, d)\) such that \(l\) is a string, called lexical form, and \(d\) is an OWL 2 datatype. Every OWL 2 datatype \(d\) is defined by: (i) the set of data values represented by \(d\), called the value space of \(d\); (ii) the set of lexical forms that can be used to refer to data values in \(d\); (iii) the set of pairs \(\langle F, v \rangle\), where \(F\) is an Iri identifying a constraining facet and \(v\) is an arbitrary data value called the constraining value, that can be used to refer to specific subsets of the value space of \(d\).

Definition 17 (OWL 2 datatype map)

A OWL 2 datatype map \(DM = (N_{DT}, N_{LS}, N_{FS}, DT, LS, FS)\) is a 6-tuple where

- \(N_{DT}\) is a set of OWL 2 datatypes not containing rdfs:Literal;
- \(N_{LS}\) is the function that assigns to each \(d \in N_{DT}\) the lexical space of \(d\) consisting of the strings, called lexical forms, used in OWL 2 ontologies to refer to data values of \(d\);
- \(N_{FS}\) is a function that assigns to each \(d \in N_{DT}\) a set of constraining facets, where a constraining facet is an operator imposing restriction on the value space of datatypes;
- \(DT\) is a function that assigns to each \(d \in N_{DT}\) the so-called value space, i.e. the set of data values represented by \(d\) in all OWL 2 ontologies;
- \(LS\) is a function that specifies, for each literal \((l, d)\) which is the data value in \(d^{DT}\) denoted by such literal;
- \(FS\) is a function that, given a constraining facet \(CF\) for a datatype \(d \in D\), assigns to \(CF\) the set \(CF^{FS} \subseteq d^{DT}\).
Note that, for every \( dt \in D'_{owl} \), if a datatype map \( DM = (N_{DT}, N_{LS}, N_{FS}, \cdot_{DT}, \cdot_{LS}, \cdot_{FS}) \) is such that \( dt \in N_{DT} \), then all the functions of \( DM \) when applied to \( dt \) are required to coincide with the corresponding functions specified by the documents defining \( dt \), namely \( \text{[32]} \) for datatypes whose IRIs start with prefix \( xsd \) and \( \text{[15]} \) for datatypes whose IRIs start with prefix \( owl \). This means that interpretations of datatypes listed in Table 4.3 are fixed once and for all in any \( OWL 2 \) DL ontology.

Now we are ready to introduce the notion of \( OWL 2 \) vocabulary.

**Definition 18** Given an \( OWL 2 \) datatype map \( DM \), a \( OWL 2 \) vocabulary \( V=(V_C, V_OP, V_DP, V_I, V_{DT}, V_LT, V_{FA}) \) over \( DM \) is a 7-tuple where:

- \( V_C \) is a set of classes containing \( owl:Thing \) and \( owl:Nothing \);
- \( V_OP \) is a set of object properties containing \( owl:topObjectProperty \) and \( owl:bottomObjectProperty \);
- \( V_DP \) is a set of data properties containing \( owl:topDataProperty \) and \( owl:bottomDataProperty \);
- \( V_{DT} \) is a set of datatypes such that \( \{rdfs:Literal\} \cup N_{DT} \subseteq V_{DT} \);
- \( V_LT \) is a set of literals that are lexical forms of datatypes defined by \( DM \);
- \( V_{FA} \) is the set of pairs \((F,lt)\) such that the literal \( lt \) satisfies the constraining facet \( F \) according to \( DM \);
- \( V_OP \cap R_{owl} = \{owl:topObjectProperty, owl:bottomObjectProperty\} \);
- \( V_DP \cap R_{owl} = \{owl:topDataProperty, owl:bottomDataProperty\} \);
- \( V_I \cap R_{owl} = \emptyset \);
- \( V_{DT} \cap R_{owl} \subseteq D_{owl} \);
- \( V_C \cap R_{owl} = \{owl:Thing, owl:Nothing\} \);
- \( V_C \cap V_{DT} = \emptyset \);
- \( V_{OP} \cap V_{DP} = \emptyset \);

Note that the conditions imposed over the sets forming a \( OWL 2 \) vocabulary are given to make the vocabulary satisfy the restrictions given in Section 4.2.4 for \( OWL 2 \) DL ontologies.

Given an \( OWL 2 \) vocabulary \( V \) over a \( OWL 2 \) datatype map \( DM \), a direct interpretation for \( DM \) and \( V \) is a 10-tuple \( I=(\Delta_o, \Delta_v, C, OP, DP, I, DT, LT, FA, \text{NAMED}) \) where:

- \( \Delta_o \), the object domain, and \( \Delta_v \), the value domain are two disjoint non-empty sets;
- \( C \) is a function s.t.:
  - \( \forall C \in V_C, C \subseteq \Delta_o \);
  - \( owl:Thing^C = \Delta_o \);
  - \( owl:Nothing^C = \emptyset \).
\( \mathcal{OP} \) is a function s.t.:
- \( \forall \mathcal{OP} \in V_{\mathcal{OP}}, \mathcal{OP}^{\mathcal{OP}} \subseteq \Delta_o \times \Delta_o; \)
- \( \text{owl:topObjectProperty}^{\mathcal{OP}} = \Delta_o \times \Delta_o; \)
- \( \text{owl:bottomObjectProperty}^{\mathcal{OP}} = \emptyset. \)

\( \mathcal{DP} \) is a function s.t.:
- \( \forall \mathcal{DP} \in V_{\mathcal{DP}}, \mathcal{DP}^{\mathcal{DP}} \subseteq \Delta_o \times \Delta_v; \)
- \( \text{owl:topDataProperty}^{\mathcal{DP}} = \Delta_o \times \Delta_v; \)
- \( \text{owl:bottomDataProperty}^{\mathcal{DP}} = \emptyset. \)

\( \mathcal{I} \) is a function s.t. \( \forall \text{ind} \in V_{\mathcal{I}}, \text{ind}^{\mathcal{I}} \in \Delta_o; \)

\( \mathcal{DT} \) is a function s.t.:
- \( \forall \text{DT} \in V_{\mathcal{DT}}, \text{DT}^{\mathcal{DT}} \subseteq \Delta_v; \)
- \( \text{rdfs:Literal}^{\mathcal{DT}} = \Delta_v. \)

\( \mathcal{LT} \) is a function s.t. \( \forall \text{lt} \in V_{\mathcal{LT}}, \text{lt}^{\mathcal{LT}} = (LV, DT)^{LS}; \)

\( \mathcal{FA} \) is a function s.t. \( \forall (F, \text{lt}) \in V_{\mathcal{FA}}, (F, \text{lt})^{\mathcal{FA}} = (F, (\text{lt})^{\mathcal{LT}})^{FS}; \)

\( \text{NAMED} \) is a set s.t. \( \text{NAMED} \subseteq \Delta_o. \) If \( \text{ind} \) is a named individual s.t. \( \text{ind} \in V_{\mathcal{I}}, \) then \( (\text{ind})^{\mathcal{I}} \in \text{NAMED}. \)

The above functions are then extended to interpret complex expressions as follows, where \( \#B \) denotes the cardinality of a set \( B: \)

- \( \text{ObjectIntersectionOf}(ce_1 \ldots ce_n)^C = (ce_1)^C \cap \ldots \cap (ce_n)^C; \)
- \( \text{ObjectUnionOf}(ce_1 \ldots ce_n)^C = (ce_1)^C \cup \ldots \cup (ce_n)^C; \)
- \( \text{ObjectOneOf}(\text{ind}_1 \ldots \text{ind}_n)^C = \{ (\text{ind}_1)^I, \ldots, (\text{ind}_n)^I \} \)
- \( \text{ObjectComplementOf}(ce)^C = \Delta_o \setminus (ce)^C; \)
- \( \text{ObjectSomeValuesFrom}(\text{ope}\ ce)^C = \{ \text{ind}_1 | \exists \text{ind}_2, (\text{ind}_1, \text{ind}_2) \in (\text{ope})^{\mathcal{OP}}, \text{ind}_2 \in (ce)^C \}; \)
- \( \text{ObjectAllValuesFrom}(\text{ope}\ ce)^C = \{ \text{ind}_1 | \forall \text{ind}_2, (\text{ind}_1, \text{ind}_2) \in (\text{ope})^{\mathcal{OP}}, \text{ind}_2 \in (ce)^C \}; \)
- \( \text{ObjectHasValue}(\text{ope}\ ind_1)^C = \{ \text{ind}_2 | (\text{ind}_2, (\text{ind}_1)^I) \in (\text{ope})^{\mathcal{OP}} \}; \)
- \( \text{ObjectHasSelf}(\text{ope})^C = \{ \text{ind} | (\text{ind}, \text{ind}) \in (\text{ope})^{\mathcal{OP}} \}; \)
- \( \text{ObjectMinCardinality}(k\ \text{ope}\ ce)^C = \{ \text{ind} | \# \{ \text{ind}_i | (\text{ind}, \text{ind}_i) \in (\text{ope})^{\mathcal{OP}}, \text{ind}_i \in (ce)^C \} \geq k \}; \)
- \( \text{ObjectMaxCardinality}(k\ \text{ope}\ ce)^C = \{ \text{ind} | \# \{ \text{ind}_i | (\text{ind}, \text{ind}_i) \in (\text{ope})^{\mathcal{OP}}, \text{ind}_i \in (ce)^C \} \leq k \}; \)
• (ObjectExactCardinality\((k \text{ ope ce})\))^C = \{\text{ind} \mid \#\{\text{ind}_i \mid (\text{ind}, \text{ind}_i) \in \text{(ope)}^{DP}, \text{ind}_i \in (\text{ce})^C\} = k\};

• (DataSomeValuesFrom\((dp \text{ dr})\))^C = \{\text{ind} \mid \exists \text{lt}. (\text{ind}, \text{lt}) \in (\text{dp})^{DP}, \text{lt} \in (\text{dr})^{DT}\};

• (DataAllValuesFrom\((dp \text{ dr})\))^C = \{\text{ind} \mid \forall \text{lt}. (\text{ind}, \text{lt}) \in (\text{dp})^{DP} \implies \text{lt} \in (\text{dr})^{DT}\};

• (DataHasValue\((dp \text{ lt})\))^C = \{\text{ind} \mid (\text{ind}, \text{lt})^{LT} \in (\text{dp})^{DP}\};

• (DataMinCardinality\((k \text{ dp dr})\))^C = \{\text{ind} \mid \#\{\text{lt} \mid (\text{ind}, \text{lt}) \in (\text{dp})^{DP}, \text{lt} \in (\text{dr})^{DT}\} \geq k\};

• (DataMaxCardinality\((k \text{ dp dr})\))^C = \{\text{ind} \mid \#\{\text{lt} \mid (\text{ind}, \text{lt}) \in (\text{dp})^{DP}, \text{lt} \in (\text{dr})^{DT}\} \leq k\};

• (DataExactCardinality\((k \text{ dp dr})\))^C = \{\text{ind} \mid \#\{\text{lt} \mid (\text{ind}, \text{lt}) \in (\text{dp})^{DP}, \text{lt} \in (\text{dr})^{DT}\} = k\};

• (ObjectInverseOf\((\text{op})\))^{OP} = \{\{\text{ind}_1, \text{ind}_2\} \mid (\text{ind}_2, \text{ind}_1) \in (\text{op})^{OP}\};

• (DataIntersectionOf\((\text{dr}_1 \ldots \text{dr}_n)\))^{DT} = (\text{dr}_1)^{DT} \cap \ldots \cap (\text{dr}_n)^{DT};

• (DataUnionOf\((\text{dr}_1 \ldots \text{dr}_n)\))^{DT} = (\text{dr}_1)^{DT} \cup \ldots \cup (\text{dr}_n)^{DT};

• (DataComplementOf\((\text{dr})\))^{DT} = \Delta_v \setminus (\text{dr})^{DT};

• (ObjectOneOf\((\text{lt}_1 \ldots \text{lt}_n)\))^{DT} = \{(\text{lt}_1)^{LT}, \ldots, (\text{lt}_n)^{LT}\};

• (DatatypeRestriction\((\text{DT} F_1 \text{ lt}_1 \ldots F_n \text{ lt}_n)\))^{DT} = (\text{DT})^{DT} \cap (F_1, \text{lt}_1)^{FA} \cap \ldots \cap (F_n, \text{lt}_n)^{FA}.

Now we turn our attention to the notion of satisfaction of an axiom with respect to a direct interpretation \(\mathcal{I}\). We say that a direct interpretation \(\mathcal{I}\) satisfy a logical axiom \(Ax\), written \(\mathcal{I} \models_{DS} Ax\) if one of the following conditions holds:

• \(\mathcal{I} \models_{DS} \text{ClassAssertion}(\text{ce ind}, \text{if} (\text{ind})^I \in (\text{ce})^C)\);

• \(\mathcal{I} \models_{DS} \text{ObjectPropertyAssertion}(\text{ope ind}_1 \text{ ind}_2), \text{if} (\text{(ind}_1)^I, (\text{ind}_2)^I) \in (\text{ope})^{OP})\);

• \(\mathcal{I} \models_{DS} \text{DataPropertyAssertion}(\text{dp ind lt}), \text{if} ((\text{ind})^I, (\text{lt})^{LT}) \in (\text{dp})^{DP})\);

• \(\mathcal{I} \models_{DS} \text{NegativeObjectPropertyAssertion}(\text{ope ind}_1 \text{ ind}_2), \text{if} ((\text{ind}_1)^I, (\text{ind}_2)^I) \notin (\text{ope})^{OP})\);

• \(\mathcal{I} \models_{DS} \text{NegativeDataPropertyAssertion}(\text{dp ind lt}), \text{if} ((\text{ind})^I, (\text{lt})^{LT}) \notin (\text{dp})^{DP})\);

• \(\mathcal{I} \models_{DS} \text{SameIndividual(\text{ind}_1 \ldots \text{ind}_n}), \text{if} (\text{ind}_i)^I = (\text{ind}_j)^I, \text{for} i, j \in \{1, \ldots, n\}\).
• $I \models_{DS} \text{DifferentIndividuals}(ind_1 \ldots ind_n)$, 
  if $(ind_i)^I \neq (ind_j)^I$, for $i, j \in \{1, \ldots, n\}$;

• $I \models_{DS} \text{SubClassOf}(ce_1 \ ce_2)$, if $(ce_1)^C \subseteq (ce_2)^C$;

• $I \models_{DS} \text{SubObjectPropertyOf}(ope_1 \ ope_2)$, if $(ope_1)^{OP} \subseteq (ope_2)^{OP}$;

• $I \models_{DS} \text{SubObjectPropertyOf}(\text{ObjectPropertyChain}(op_1 \ldots op_n) \ op)$, 
  if, for all $\{ind_0, \ldots, ind_n\}$ | $(ind_0, ind_1) \in (op_1)^{OP}$ and 
  $(ind_1, ind_2) \in (op_2)^{OP}$ and \ldots and $(ind_{n-1}, ind_n) \in (op_n)^{OP}$, 
  it is implied that $(ind_0, ind_n) \in (op)^{OP}$;

• $I \models_{DS} \text{InverseObjectProperties}(ope_1 \ ope_2)$, 
  if $(ope_1)^{OP} = \{\langle ind_1, ind_2 \rangle | \langle ind_2, ind_1 \rangle \in (ope_2)^{OP}\}$;

• $I \models_{DS} \text{SubDataPropertyOf}(dp_1 \ dp_2)$, if $(dp_1)^{DP} \subseteq (dp_2)^{DP}$;

• $I \models_{DS} \text{EquivalentClasses}(ce_1 \ldots ce_n)$, if $(ce_i)^C = (ce_j)^C$, 
  for $i, j \in \{1, \ldots, n\}$;

• $I \models_{DS} \text{EquivalentObjectProperties}(ope_1 \ldots ope_n)$, 
  if $(ope_i)^{OP} = (ope_j)^{OP}$, for $i, j \in \{1, \ldots, n\}$;

• $I \models_{DS} \text{EquivalentDataProperties}(dp_1 \ldots dp_n)$, 
  if $(dp_i)^{DP} = (dp_j)^{DP}$, for $i, j \in \{1, \ldots, n\}$;

• $I \models_{DS} \text{DisjointClasses}(ce_1 \ldots ce_n)$, if $(ce_i)^C \cap (ce_j)^C = \emptyset$, 
  for $i, j \in \{1, \ldots, n\}$;

• $I \models_{DS} \text{DisjointObjectProperties}(ope_1 \ldots ope_n)$, 
  if $(ope_i)^{OP} \cap (ope_j)^{OP} = \emptyset$, for $i, j \in \{1, \ldots, n\}$;

• $I \models_{DS} \text{DisjointDataProperties}(dp_1 \ldots dp_n)$, 
  if $(dp_i)^{DP} \cap (dp_j)^{DP} = \emptyset$, for $i, j \in \{1, \ldots, n\}$;

• $I \models_{DS} \text{DisjointUnion}(ce_1 \ldots ce_n)$, if $(c)^C = (ce_1)^C \cup \ldots \cup (ce_n)^C$ 
  and $(ce_i)^C \cap (ce_j)^C = \emptyset$ for $i, j \in \{1, \ldots, n\}$ and $i \neq j$;

• $I \models_{DS} \text{ObjectPropertyDomain}(ope ce)$, if for all $ind_1, ind_2$ such that 
  $\langle ind_1, ind_2 \rangle \in (ope)^{OP}$ it is implied that $ind_1 \in (ce)^C$;

• $I \models_{DS} \text{ObjectPropertyRange}(ope ce)$, if for all $ind_1, ind_2$ such that 
  $\langle ind_1, ind_2 \rangle \in (ope)^{OP}$ it is implied that $ind_2 \in (ce)^C$;

• $I \models_{DS} \text{DataPropertyDomain}(dp ce)$, if for all $ind, lt$ such that 
  $\langle ind, lt \rangle \in (dp)^{DP}$ it is implied that $ind \in (ce)^C$;

• $I \models_{DS} \text{DataPropertyRange}(dp dr)$, if for all $ind, lt$ such that 
  $\langle ind, lt \rangle \in (dp)^{DP}$ it is implied that $lt \in (dr)^{DT}$;

• $I \models_{DS} \text{ReflexiveObjectProperty}(ope)$, if for all $ind$ in $\Delta_o$ it is implied 
  that $\langle ind, ind \rangle \in (ope)^{OP}$;
4.3 OWL 2 Semantics

- $\mathcal{I} \models_{DS} \text{IrreflexiveObjectProperty}(\text{ope})$, if for all $\text{ind}$ in $\Delta_\circ$, it is implied that $\langle \text{ind}, \text{ind} \rangle \notin (\text{ope})^{OP}$;

- $\mathcal{I} \models_{DS} \text{SymmetricObjectProperty}(\text{ope})$, if for all $\text{ind}_1, \text{ind}_2$ such that $\langle \text{ind}_1, \text{ind}_2 \rangle \in (\text{ope})^{OP}$ it is implied that $\langle \text{ind}_2, \text{ind}_1 \rangle \in (\text{ope})^{OP}$;

- $\mathcal{I} \models_{DS} \text{AsymmetricObjectProperty}(\text{ope})$, if for all $\text{ind}_1, \text{ind}_2, \text{ind}_3$ such that $\langle \text{ind}_1, \text{ind}_2 \rangle \in (\text{ope})^{OP}$ it is implied that $\langle \text{ind}_2, \text{ind}_1 \rangle \notin (\text{ope})^{OP}$;

- $\mathcal{I} \models_{DS} \text{TransitiveObjectProperty}(\text{ope})$, if for all $\text{ind}_1, \text{ind}_2$ such that $\langle \text{ind}_1, \text{ind}_2 \rangle \in (\text{ope})^{OP}$ and $\langle \text{ind}_2, \text{ind}_3 \rangle \in (\text{ope})^{OP}$ it is implied that $\langle \text{ind}_1, \text{ind}_3 \rangle \in (\text{ope})^{OP}$;

- $\mathcal{I} \models_{DS} \text{FunctionalObjectProperty}(\text{ope})$, if for all $\text{ind}, \text{ind}_1, \text{ind}_2$ such that $\langle \text{ind}, \text{ind}_1 \rangle \in (\text{ope})^{OP}$ and $\langle \text{ind}, \text{ind}_2 \rangle \in (\text{ope})^{OP}$ it is implied that $\text{ind}_1 = \text{ind}_2$;

- $\mathcal{I} \models_{DS} \text{InverseFunctionalObjectProperty}(\text{ope})$, if for all $\text{ind}, \text{ind}_1, \text{ind}_2$ such that $\langle \text{ind}, \text{ind}_1 \rangle \in (\text{ope})^{OP}$ and $\langle \text{ind}, \text{ind}_2 \rangle \in (\text{ope})^{OP}$ it is implied that $\text{ind}_1 = \text{ind}_2$;

- $\mathcal{I} \models_{DS} \text{FunctionalDataProperty}(\text{dp})$, if for all $\text{ind}, \text{lt}_1, \text{lt}_2$ such that $\langle \text{ind}, \text{lt}_1 \rangle \in (\text{dp})^{DP}$ and $\langle \text{ind}, \text{lt}_2 \rangle \in (\text{dp})^{DP}$ it is implied that $\text{lt}_1 = \text{lt}_2$;

- $\mathcal{I} \models_{DS} \text{HasKey}(\text{ce} (\text{ope}_1 \ldots \text{ope}_n) (\text{dp}_1 \ldots \text{dp}_n))$, if for all $\text{ind}_1, \text{ind}_2, a_1, \ldots, a_m, \text{lt}_1, \ldots, \text{lt}_n$ such that $\text{ind}_1 \in (\text{ce})^C$, $\text{ind}_2 \in \text{NAMED}$, $a_i \in \text{NAMED}$, and $\langle \text{ind}_1, a_i \rangle \in (\text{ope}_i)^{OP}$, $\langle \text{ind}_2, a_i \rangle \in (\text{ope}_i)^{OP}$ and $\langle \text{ind}_1, \text{lt}_i \rangle \in (\text{dp}_i)^{DP}$ and $\langle \text{ind}_2, \text{lt}_j \rangle \in (\text{dp}_j)^{DP}$ for $i, j \in \{1, \ldots, n\}$, it is implied that $\text{ind}_1 = \text{ind}_2$;

- $\mathcal{I} \models_{DS} \text{DatatypeDefinition}(\text{dt} \text{ dr})$, if $(\text{dt})^{DT} = (\text{dr})^{DT}$.

We say that a direct interpretation $\mathcal{I}$ satisfies an ontology $\mathcal{O}$, written $\mathcal{I} \models_{DS} \mathcal{O}$, if it satisfies all the logical axioms of $\mathcal{O}$. In this case we call $\mathcal{I}$ a direct model of $\mathcal{O}$. An ontology is satisfiable under DS if it admits at least one direct model, otherwise is unsatisfiable under DS. Finally we say that an axiom $Ax$ is entailed by an ontology $\mathcal{O}$ under DS, written $\mathcal{O} \models_{DS} Ax$, if $Ax$ is satisfied by every direct model of $\mathcal{O}$, whereas $Ax$ is violated under DS by $\mathcal{O}$ if $\mathcal{O} \cup Ax$ is unsatisfiable under DS. In what follows, we refer with OWL 2 DL to the language that is obtained by interpreting OWL 2 DL ontology by using the direct semantics.

All the inference problem that have been described in the context of DL knowledge bases in Section 2.3 can be taken into account also for OWL 2 DL ontologies. Moreover, as the expressive power of OWL 2 DL coincides with that of SROIQ, the results of computational complexity given in Section 2.4 for the case of SROIQ, hold for OWL 2 DL as well.

4.3.2 RDF-Based Semantics

The OWL 2 RDF-Based Semantics is defined as a semantic extension of the RDFS semantics described in Section 4.1.3. Like any other RDF semantic extension, such a semantics is
defined by assigning a predefined meaning to a suitable set of IRIs representing the base OWL 2 vocabulary for RDF, and by specifying a set of axiomatic triples taking into account such a vocabulary. Then, given any OWL 2 ontology \( \mathcal{O} \), instead of assigning meaning directly to the set of axioms of \( \mathcal{O} \), the RDF-Based Semantics interprets the RDF graph \( G(\mathcal{O}) \) which \( \mathcal{O} \) can be mapped to. The OWL 2 RDF-Based vocabulary, denoted by \( V_{\text{rdf}}^{\text{owl}} \), extends the RDFS vocabulary \( V_{\text{RDFS}} \) defined in Section 4.1.3 with the IRIs listed in Table 4.9.

<table>
<thead>
<tr>
<th>IRIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) owl:AllDifferent</td>
</tr>
<tr>
<td>(2) owl:AllDisjointClasses</td>
</tr>
<tr>
<td>(3) owl:AllDisjointProperties</td>
</tr>
<tr>
<td>(4) owl:AsymmetricProperty</td>
</tr>
<tr>
<td>(5) owl:Axiom</td>
</tr>
<tr>
<td>(6) owl:Class</td>
</tr>
<tr>
<td>(7) rdfs:Datatype</td>
</tr>
<tr>
<td>(8) owl:DatatypeProperty</td>
</tr>
<tr>
<td>(9) owl:FunctionalProperty</td>
</tr>
<tr>
<td>(10) owl:InverseFunctionalProperty</td>
</tr>
<tr>
<td>(11) owl:IrreflexiveProperty</td>
</tr>
<tr>
<td>(12) owl:NamedIndividual</td>
</tr>
<tr>
<td>(13) owl:NegativePropertyAssertion</td>
</tr>
<tr>
<td>(14) owl:Nothing</td>
</tr>
<tr>
<td>(15) owl:ObjectProperty</td>
</tr>
<tr>
<td>(16) owl:ObjectProperty</td>
</tr>
<tr>
<td>(17) owl:Restriction</td>
</tr>
<tr>
<td>(18) owl:Restriction</td>
</tr>
<tr>
<td>(19) owl:Restriction</td>
</tr>
<tr>
<td>(20) owl:Restriction</td>
</tr>
<tr>
<td>(21) owl:restriction</td>
</tr>
<tr>
<td>(22) owl:restriction</td>
</tr>
<tr>
<td>(23) owl:unionOf</td>
</tr>
<tr>
<td>(24) owl:unionOf</td>
</tr>
<tr>
<td>(25) owl:unionOf</td>
</tr>
<tr>
<td>(26) owl:unionOf</td>
</tr>
<tr>
<td>(27) owl:unionOf</td>
</tr>
<tr>
<td>(28) owl:unionOf</td>
</tr>
<tr>
<td>(29) owl:unionOf</td>
</tr>
<tr>
<td>(30) owl:unionOf</td>
</tr>
<tr>
<td>(31) owl:unionOf</td>
</tr>
<tr>
<td>(32) owl:unionOf</td>
</tr>
<tr>
<td>(33) owl:unionOf</td>
</tr>
<tr>
<td>(34) owl:unionOf</td>
</tr>
<tr>
<td>(35) owl:unionOf</td>
</tr>
<tr>
<td>(36) owl:unionOf</td>
</tr>
<tr>
<td>(37) owl:unionOf</td>
</tr>
<tr>
<td>(38) owl:unionOf</td>
</tr>
<tr>
<td>(39) owl:unionOf</td>
</tr>
<tr>
<td>(40) owl:unionOf</td>
</tr>
<tr>
<td>(41) owl:unionOf</td>
</tr>
<tr>
<td>(42) owl:unionOf</td>
</tr>
<tr>
<td>(43) owl:unionOf</td>
</tr>
<tr>
<td>(44) owl:unionOf</td>
</tr>
<tr>
<td>(45) owl:unionOf</td>
</tr>
<tr>
<td>(46) owl:unionOf</td>
</tr>
<tr>
<td>(47) owl:unionOf</td>
</tr>
<tr>
<td>(48) owl:unionOf</td>
</tr>
<tr>
<td>(49) owl:unionOf</td>
</tr>
<tr>
<td>(50) owl:unionOf</td>
</tr>
<tr>
<td>(51) owl:unionOf</td>
</tr>
<tr>
<td>(52) owl:unionOf</td>
</tr>
<tr>
<td>(53) owl:unionOf</td>
</tr>
<tr>
<td>(54) owl:unionOf</td>
</tr>
</tbody>
</table>

Table 4.9. OWL 2 RDF-Based vocabulary

In the following we denote by \( \mathbb{N} \) the value space containing all the non-negative integers and, by \( \mathcal{S}_U \) the set containing all the possible sequences of elements in \( U \). A OWL 2 RDF-Based interpretation \( \mathcal{I} = (U, P, C, V, DTP, DT, Ext, C, IS, IL) \) is a RDFS interpretation extended with sets \( DTP \) and \( DT \) satisfying the following semantic conditions:

- \( C \subseteq U \);
- \( V \subseteq U \);
- \( P \subseteq U \);
- \( DTP \subseteq C \);
- \( \mathcal{S}_U = (\mathcal{I}(\text{rdf:List}))^C \);
- \( DTP \subseteq P \);
- if \( x \in C \), then \( (x)^C \subseteq U \);
- if \( x \in P \), then \( (x)^{Ext} \subseteq U \times U \);
- if \( x \in DT \), then \( (x)^C \subseteq V \);
- if \( x \in DTP \), then \( (x)^{Ext} \subseteq U \times V \);
- \( DT = (\mathcal{I}(\text{rdfs:Datatype}))^C \);
- \( P = (\mathcal{I}(\text{rdf:Property}))^C \);
- \( DTP = (\mathcal{I}(\text{rdfs:Datatype}))^C \);
• if \( x \) is an IRI between (1) and (20) in Table 4.9 then \( \mathcal{I}(x) \in C \);
• if \( x \) is an IRI between (21) and (55) in Table 4.9 then \( \mathcal{I}(x) \in P \);
• \((\mathcal{I}(\text{owl:AllDifferent}))^C \subseteq U\);
• \((\mathcal{I}(\text{owl:AllDisjointClasses}))^C \subseteq U\);
• \((\mathcal{I}(\text{owl:AllDisjointProperties}))^C \subseteq U\);
• \((\mathcal{I}(\text{owl:AsymmetricProperty}))^C \subseteq P\);
• \((\mathcal{I}(\text{owl:Axiom}))^C \subseteq U\);
• \((\mathcal{I}(\text{owl:Class}))^C \subseteq \times U\);
• \((\mathcal{I}(\text{rdfs:Datatype}))^C = V\);
• \((\mathcal{I}(\text{owl:DatatypeProperty}))^C \subseteq \text{DTP}\);
• \((\mathcal{I}(\text{owl:FunctionalProperty}))^C \subseteq P\);
• \((\mathcal{I}(\text{owl:InverseFunctionalProperty}))^C \subseteq P\);
• \((\mathcal{I}(\text{owl:IrreflexiveProperty}))^C \subseteq P\);
• \((\mathcal{I}(\text{owl:NamedIndividual}))^C \subseteq U\);
• \((\mathcal{I}(\text{owl:NegativePropertyAssertion}))^C \subseteq U\);
• \((\mathcal{I}(\text{owl:Nothing}))^C = \emptyset\);
• \((\mathcal{I}(\text{owl:ObjectProperty}))^C = P\);
• \((\mathcal{I}(\text{owl:ReflexiveProperty}))^C \subseteq P\);
• \((\mathcal{I}(\text{owl:Restriction}))^C \subseteq C\);
• \((\mathcal{I}(\text{owl:SymmetricProperty}))^C \subseteq P\);
• \((\mathcal{I}(\text{owl:Thing}))^C = U\);
• \((\mathcal{I}(\text{owl:TransitiveProperty}))^C \subseteq P\);
• \((\mathcal{I}(\text{owl:allValuesFrom}))^\text{Ext} \subseteq (\mathcal{I}(\text{owl:Restriction}))^C \times C\);
• \((\mathcal{I}(\text{owl:bottomDataProperty}))^\text{Ext} = \emptyset\);
• \((\mathcal{I}(\text{owl:bottomObjectProperty}))^\text{Ext} = \emptyset\);
• \((\mathcal{I}(\text{owl:cardinality}))^\text{Ext} \subseteq (\mathcal{I}(\text{owl:Restriction}))^C \times \mathbb{N}\);
• \((\mathcal{I}(\text{owl:complementOf}))^\text{Ext} \subseteq C \times C\);
• \((\mathcal{I}(\text{owl:datatypeComplementOf}))^\text{Ext} \subseteq \text{DT} \times \text{DT}\);
• \((\mathcal{I}(\text{owl:differentFrom}))^\text{Ext} \subseteq \text{U} \times \text{U}\);
• \((I(owl:disjointUnionOf))^{Ext} \subseteq C \times S_U;\)
• \((I(owl:disjointWith))^{Ext} \subseteq C \times C;\)
• \((I(owl:distinctMembers))^{Ext} \subseteq (I(owl:AllDifferent))^{C} \times S_U;\)
• \((I(owl:equivalentClass))^{Ext} \subseteq C \times C;\)
• \((I(owl:hasKey))^{Ext} \subseteq C \times S_U;\)
• \((I(owl:hasSelf))^{Ext} \subseteq (I(owl:Restriction))^{C} \times U;\)
• \((I(owl:hasValue))^{Ext} \subseteq (I(owl:Restriction))^{C} \times U;\)
• \((I(owl:intersectionOf))^{Ext} \subseteq C \times S_U;\)
• \((I(owl:inverseOf))^{Ext} \subseteq P \times P;\)
• \((I(owl:maxCardinality))^{Ext} \subseteq (I(owl:Restriction))^{C} \times N;\)
• \((I(owl:maxQualifiedCardinality))^{Ext} \subseteq (I(owl:Restriction))^{C} \times N;\)
• \((I(owl:minCardinality))^{Ext} \subseteq (I(owl:Restriction))^{C} \times N;\)
• \((I(owl:minQualifiedCardinality))^{Ext} \subseteq (I(owl:Restriction))^{C} \times N;\)
• \((I(owl:onClass))^{Ext} \subseteq (I(owl:Restriction))^{C} \times C;\)
• \((I(owl:onDataRange))^{Ext} \subseteq (I(owl:Restriction))^{C} \times DT;\)
• \((I(owl:onDatatype))^{Ext} \subseteq DT \times DT;\)
• \((I(owl:oneOf))^{Ext} \subseteq C \times S_U;\)
• \((I(owl:propertyDisjointWith))^{Ext} \subseteq P \times P;\)
• \((I(owl:sameAs))^{Ext} \subseteq U \times U;\)
• \((I(owl:someValuesFrom))^{Ext} \subseteq (I(owl:Restriction))^{C} \times C;\)
• \((I(owl:topDataProperty))^{Ext} \subseteq U \times V;\)
• \((I(owl:topObjectProperty))^{Ext} \subseteq U \times U;\)
• \((I(owl:unionOf))^{Ext} \subseteq C \times S_U;\)
• if \(\langle z, c \rangle \in (I(owl:someValuesFrom))^{Ext}\) and \(\langle z, p \rangle \in (I(owl:onProperty))^{Ext}\),
  then \(z)^C = \{ x \mid \exists y : (x, y)(p)^{Ext} \text{ and } y \in (c)^C\};
• if \(\langle z, c \rangle \in (I(owl:allValuesFrom))^{Ext}\) and \(\langle z, p \rangle \in I(owl:onProperty))^{Ext}\),
  then \(z)^C = \{ x \mid \forall y : (x, y)(p)^{Ext} \text{ implies } \in (c)^C\};
• if $(z,a) \in (I(\text{owl:hasValue}))^{Ext}$ and $(z,p) \in (I(\text{owl:onProperty}))^{Ext}$, then $(z)^C = \{ x \mid \langle x, a \rangle \in (p)^{Ext} \}$;

• if $(z,a) \in (I(\text{owl:hasSelf}))^{Ext}$ and $(z,p) \in (I(\text{owl:onProperty}))^{Ext}$, then $(z)^C = \{ x \mid \langle x, x \rangle \in (p)^{Ext} \}$;

• if $(z,n) \in (I(\text{owl:minCardinality}))^{Ext}$ and $(z,p) \in (I(\text{owl:onProperty}))^{Ext}$, then $(z)^C = \{ x \mid \# \{ y \mid \langle x, y \rangle \in (p)^{Ext} \} \geq n \}$;

• if $(z,n) \in (I(\text{owl:maxCardinality}))^{Ext}$ and $(z,p) \in (I(\text{owl:onProperty}))^{Ext}$, then $(z)^C = \{ x \mid \# \{ y \mid \langle x, y \rangle \in (p)^{Ext} \} \leq n \}$;

• if $(z,n) \in (I(\text{owl:minQualifiedCardinality}))^{Ext}$ and $(z,p) \in (I(\text{owl:onProperty}))^{Ext}$ and $(z,c) \in (I(\text{owl:onClass}))^{Ext}$, then $(z)^C = \{ x \mid \# \{ y \mid \langle x, y \rangle \in (p)^{Ext} \text{ and } y \in (c)^C \} \geq n \}$;

• if $(z,n) \in (I(\text{owl:maxQualifiedCardinality}))^{Ext}$ and $(z,p) \in (I(\text{owl:onProperty}))^{Ext}$ and $(z,c) \in (I(\text{owl:onClass}))^{Ext}$, then $(z)^C = \{ x \mid \# \{ y \mid \langle x, y \rangle \in (p)^{Ext} \text{ and } y \in (c)^C \} \leq n \}$;

• if $(a_1,a_2) \in (I(\text{owl:sameAs}))^{Ext}$, then $a_1 = a_2$;

• if $(a_1,a_2) \in (I(\text{owl:differentFrom}))^{Ext}$, then $a_1 \neq a_2$;

• if $(c_1,c_2) \in (I(\text{owl:equivalentClass}))^{Ext}$, then $c_1,c_2 \in C$ and $(c_1)^C = (c_2)^C$;

• if $(c_1,c_2) \in (I(\text{owl:disjointWith}))^{Ext}$, then $c_1,c_2 \in C$ and $(c_1)^C \cap (c_2)^C = \emptyset$;

• if $(p_1,p_2) \in (I(\text{owl:equivalentProperty}))^{Ext}$, then $p_1,p_2 \in P$ and $(p_1)^{Ext} = (p_2)^{Ext}$;

• if $(p_1,p_2) \in (I(\text{owl:propertyDisjointWith}))^{Ext}$, then $p_1,p_2 \in P$ and $(p_1)^{Ext} \cap (p_2)^{Ext} = \emptyset$;

• $\langle p_1,p_2 \rangle \in (I(\text{owl:inverseOf}))^{Ext}$ if and only if $p_1,p_2 \in P$ and $(p_1)^{Ext} = \{ \langle x, y \rangle, \langle y, x \rangle \in (p_2)^{Ext} \}$;

• $p \in (I(\text{owl:FunctionalProperty}))^{C}$ if and only if $p \in P$ and $\forall x,y_1,y_2 : \langle x, y_1 \rangle, \langle x, y_2 \rangle \in (p)^{Ext}$ implies $y_1 = y_2$;

• $p \in (I(\text{owl:InverseFunctionalProperty}))^{C}$ if and only if $p \in P$ and $\forall x,y_1,y_2 : \langle y_1, x \rangle, \langle y_2, x \rangle \in (p)^{Ext}$ implies $y_1 = y_2$;

• $p \in (I(\text{owl:ReflexiveProperty}))^{C}$ if and only if $p \in P$ and $\forall x : \langle x, x \rangle \in (p)^{Ext}$;

• $p \in (I(\text{owl:IrreflexiveProperty}))^{C}$ if and only if $p \in P$ and $\forall x : \langle x, x \rangle \notin (p)^{Ext}$;

• $p \in (I(\text{owl:SymmetricProperty}))^{C}$ if and only if $p \in P$ and $\forall x,y : \langle x,y \rangle \in (p)^{Ext}$ implies $\langle y, x \rangle \in (p)^{Ext}$;
• \( p \in (\mathcal{I}(\text{owl:AsymmetricProperty}))^C \) if and only if \( p \in \mathbb{P} \) and \( \forall x,y : \langle x,y \rangle \in (p)^{\text{Ext}} \implies \langle y,x \rangle \notin (p)^{\text{Ext}} \);

• \( p \in (\mathcal{I}(\text{owl:TransitiveProperty}))^C \) if and only if \( p \in \mathbb{P} \) and \( \forall x,y,z : \langle x,y \rangle \in (p)^{\text{Ext}} \text{ and } \langle y,z \rangle \in (p)^{\text{Ext}} \implies \langle x,z \rangle \in (p)^{\text{Ext}} \);

The notions of satisfaction, consistency and entailment under the \textsc{owl 2} RDF-Based Semantics is derived from those of satisfaction, consistency and entailment given for RDF simple semantics with datatype support given in Section 4.1.1.

**Definition 19** Let \( G \) be an RDF graph, \( D \) be an \textsc{owl 2} RDF-Based datatype map and \( V \) be a vocabulary including the \textsc{owl 2} RDF-Based vocabulary together with the IRIs in \( D_{\text{owl}} \) and let \( \mathcal{I} \) be an \textsc{owl 2} RDF-Based interpretation of \( V \) with respect to \( D \). \( \mathcal{I} \) \textsc{owl 2} RDF-Based satisfies \( G \) with respect to \( D \) and \( V \) if and only if \( \mathcal{I} \) satisfies \( G \) as a \( D \)-interpretation.

**Definition 20** Let \( S \) be a collection of RDF graphs and \( D \) be an \textsc{owl 2} RDF-Based datatype map. \( S \) is \textsc{owl 2} RDF-Based consistent with respect to \( D \) if and only if there exist some \textsc{owl 2} RDF-Based interpretation with respect to \( D \) of some vocabulary \( V \) that includes the \textsc{owl 2} RDF-Based vocabulary together with the IRIs in \( D_{\text{owl}} \).

**Definition 21** Let \( S_1 \) and \( S_2 \) be collections of RDF graphs and \( D \) be an \textsc{owl 2} RDF-Based datatype map. \( S_1 \) \textsc{owl 2} RDF-Based entails \( S_1 \), written \( S_1 \models_{\text{RDF-B}} S_2 \), with respect to \( D \) if and only if for every \textsc{owl 2} RDF-Based interpretation \( \mathcal{I} \) with respect to \( D \) of any vocabulary \( V \) that includes the \textsc{owl 2} RDF-Based vocabulary together with the IRIs in \( D_{\text{owl}} \) the following holds: if \( \mathcal{I} \) \textsc{owl 2} RDF-Based satisfies all the graphs in \( S_1 \) with respect to \( D \) and \( V \), then \( \mathcal{I} \) \textsc{owl 2} RDF-Based satisfies all the graphs in \( S_2 \) with respect to \( D \) and \( V \).

In general, the problem of deciding RDF-Based entailment between two RDF graphs is known to be undecidable [61].

### 4.4 \textsc{OWL 2 QL}

In this section we present syntax and semantics of the \textsc{OWL 2 QL} language. Such a language is a W3C standard defined as a \textsc{owl 2} profile, where a profile is a fragment of \textsc{owl 2} obtained by suitably decreasing its expressive power. In particular, \textsc{owl 2 QL} is obtained by decreasing the expressive power of \textsc{owl 2} DL as to match that of DL-Lite\(_R\), thus resulting in a language where sound and complete query answering under the direct semantics is \(\text{AC}^0\) in data complexity.

#### 4.4.1 \textsc{OWL 2 QL} syntax

The syntax of \textsc{owl 2 QL} is obtained from that of \textsc{owl 2} by imposing several restrictions on it. First of all, \textsc{owl 2 QL} does not support anonymous individuals, and limits the set of datatypes that can occur in an ontology. We denote by \( D_{\text{ql}} \) the set of datatypes supported by \textsc{owl 2 QL}. The complete list of datatypes that are in \( D_{\text{ql}} \) is depicted in Table 4.10.

As for data range expressions, \textsc{owl 2 QL} allows for using only expressions of the form \text{DataIntersectionOf}(dt_1 \ dt_2 \ldots \ dt_n), where each \( dt_i \) is a datatype in \( D_{\text{ql}} \setminus \{\text{rdfs:Literal}\} \).
An important restriction imposed by OWL 2 QL regards the allowed class expressions. Similarly to the case of basic and general concepts in DL-LiteR, OWL 2 QL makes a distinction between subclass expressions and general class expressions.

**Definition 22** (Subclass expression) An OWL 2 QL subclass expression is either a class, or an expression having one of the following forms:

- DataSomeValuesFrom($dp\ dr$), where $dp$ is a data property and $dr$ is an OWL 2 QL data range;

- ObjectSomeValuesFrom($ope\ owl:Thing$), where $ope$ is an object property expression.

**Definition 23** (General class expression) An OWL 2 QL general class expression (simply class expression) is either a OWL 2 QL subclass expression, or an expression having one of the following forms:

- ObjectSomeValuesFrom($ope\ cl$), where $ope$ is an object property expression and $cl$ is a class;

- ObjectIntersectionOf($ce_1\ ce_2\ ...\ ce_n$), where each $ce_i$ is a OWL 2 QL general class expression;

- ObjectComplementOf($sce$), where $sce$ is an OWL 2 QL subclass expression.

As for axioms, a OWL 2 QL axiom is either a declaration axiom from Table 4.6 or one of the logical axioms listed in Tables 4.11 and 4.12 where (i) $c$ denotes a class; (ii) $ce$ (possibly with subscript) denotes a OWL 2 QL general class expression; (iii) $sce$ (possibly with subscript) denotes a OWL 2 QL subclass expression; $ind$ (possibly with subscript) denotes a named individual; (iv) $ope$ (possibly with subscript) denotes an object property expression; (v) $dp$ (possibly with subscript) denotes a data property; (vi) $lt$ denotes a literal; (vii) $dr$ denotes a OWL 2 QL data range; (viii) $dt$ denotes a datatype in $\{\text{rdfs:Literal}\}$. Finally a OWL 2 QL ontology is a finite set of OWL 2 QL axioms.

---

**Table 4.10. OWL 2 QL datatypes**

<table>
<thead>
<tr>
<th>rdfs:Literal</th>
<th>rdf:PlainLiteral</th>
<th>rdf:XMLLiteral</th>
<th>owl:real</th>
</tr>
</thead>
<tbody>
<tr>
<td>owl:rational</td>
<td>xsd:nonNegativeInteger</td>
<td>xsd:hexBinary</td>
<td>xsd:decimal</td>
</tr>
<tr>
<td>xsd:string</td>
<td>xsd:normalizedString</td>
<td>xsd:dateTime</td>
<td>xsd:Name</td>
</tr>
<tr>
<td>xsd:NCName</td>
<td>xsd:base64Binary</td>
<td>xsd:int</td>
<td>xsd:NMTOKEN</td>
</tr>
<tr>
<td>xsd:anyURI</td>
<td>xsd:dateTimeStamp</td>
<td>xsd:token</td>
<td></td>
</tr>
</tbody>
</table>

**Table 4.11. OWL 2 QL ABox axioms**

| ClassAssertion($c\ ind$) | ObjectPropertyAssertion($ope\ ind_1\ ind_2$) | DataPropertyAssertion($dp\ ind\ lt$) |

---
4.4.2 Reasoning in OWL 2 QL

Just like any other OWL 2 ontology, OWL 2 QL ontologies can be interpreted by both the Direct Semantics and the RDF-Based Semantics. However, OWL 2 QL has been designed with the goal of allowing one to use, also in the context of the semantic web, the highly optimized techniques for query answering originally devised for the languages of the DL-Lite family. As such techniques can be used only with a DL-compatible semantics, in this section we only comment the known computational complexity results regarding reasoning problems for OWL 2 QL ontologies interpreted by the Direct Semantics. Inference processes in OWL 2 QL are heavily conditioned by the way datatypes are managed, so we first introduce the OWL 2 QL datatype map, denoted by $DM_{ql}$, that is, the OWL 2 datatype map used to interpret only the datatypes in $D_{ql}$. In fact, note that we can assume without loss of generality that OWL 2 QL ontologies contain only datatypes in $D_{ql}$, as the expressive power of OWL 2 QL allows one to use axioms of the form $\text{DatatypeDefinition}(dt \, dr)$ only to rename predefined datatypes from Table 4.10. Indeed, $D_{ql}$ has been designed such that the intersection of the value spaces of any set of these datatypes is either empty or infinite. Such conditions imposed on $D_{ql}$ are fundamental to ensure the FOL-rewritability of both ontology satisfiability and query answering in OWL 2 QL, as discussed in [74].

The definition of the OWL 2 QL datatype map is as follows:

**Definition 24** (OWL 2 QL datatype map) The OWL 2 QL datatype map is the OWL 2 datatype map $DM_{ql} = (D_{ql}, N_{LS}, N_{FS}, DT, LS, FS)$.

Furthermore, a OWL 2 QL vocabulary is a OWL 2 vocabulary defined over $DM_{ql}$.

In [31] authors show how algorithms and techniques originally devised for deciding inference problems in DL-Lite$_R$ can be adapted to OWL 2 QL interpreted by the direct semantics, thus, the complexity of such problems are at follows:

- Ontology satisfiability is PTIME in the total number of logical axioms and $AC^0$ in the number of ABox axioms;
- Class expression subsumption is PTIME in the number of TBox axioms;
- Instance checking is PTIME in the number of logical axioms and $AC^0$ in the number of ABox axioms;
- Answering union of conjunctive queries is PTIME in the number of TBox axioms, $AC^0$ in the number of ABox axioms and NP-complete in the total number of logical axioms.

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Semantic Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DifferentIndividuals($ind_1$ ... $ind_n$)</td>
<td>SubClassOf($soc , ce$)</td>
</tr>
<tr>
<td>SubDataPropertyOf($dp_1 , dp_2$)</td>
<td>SubObjectPropertyOf($ope_1 , ope_2$)</td>
</tr>
<tr>
<td>ObjectPropertyDomain($ope , ce$)</td>
<td>ObjectPropertyRange($ope , ce$)</td>
</tr>
<tr>
<td>DataPropertyDomain($dp , ce$)</td>
<td>DataPropertyRange($dp , dr$)</td>
</tr>
<tr>
<td>DisjointObjectProperties($ope_1$ ... $ope_n$)</td>
<td>DisjointClasses($soc_1$ ... $soc_n$)</td>
</tr>
<tr>
<td>DisjointDataProperties($dp_1$ ... $dp_n$)</td>
<td>EquivalentClasses($soc_1$ ... $soc_n$)</td>
</tr>
<tr>
<td>EquivalentDataProperties($dp_1$ ... $dp_n$)</td>
<td>EquivalentObjectProperties($ope_1$ ... $ope_n$)</td>
</tr>
<tr>
<td>IrreflexiveObjectProperty($ope$)</td>
<td>ReflexiveObjectProperty($ope$)</td>
</tr>
<tr>
<td>SymmetricObjectProperty($ope$)</td>
<td>AsymmetricObjectProperty($ope$)</td>
</tr>
<tr>
<td>InverseObjectProperty($ope$)</td>
<td>DatatypeDefinition($dt , dr$)</td>
</tr>
</tbody>
</table>

**Table 4.12.** OWL 2 QL TBox Axioms
4.5 The SPARQL query language

In this chapter we introduce the basic features of the SPARQL 1.1 (from now on just SPARQL) query language, that is the most recent version of the W3C standard pattern-matching language for querying RDF graphs in the context of the Semantic Web. Note that the focus of this thesis is on evaluating (a generalization of) the class of UCQs over OWL 2 DL ontologies by using a query language whose syntax was widely recognized as a standard. Thus, as we use SPARQL simply as syntactic mean to express (a generalization) of UCQs, we only introduce the SPARQL constructs needed to reach such a goal. Readers that are interested in detailed descriptions of SPARQL queries and of their underlining algebra may refer to [68], or to [43].

We start by introducing the syntax of the very basic building blocks of SPARQL, the so called basic graph patterns, that are in turn based on the notion of triple pattern. Intuitively, a triple pattern is a RDF triple where variables may occur.

Let $\mathcal{I}_r$, $\mathcal{L}_{rdf}$, $\mathcal{B}_{rdf}$ and $\text{Terms}_{rdf}$ be the sets defined at the beginning of Section 4.1 and let $\mathcal{V}$ denote a countably infinite set of query variables disjoint with $\text{Terms}_{rdf}$. We assume that elements in $\mathcal{V}$ starts with the symbol $\$$. A triple pattern is an expression of the form

$$(\text{Terms}_{rdf} \cup \mathcal{V}) \times (\mathcal{I}_r \cup \mathcal{V}) \times (\text{Terms}_{rdf} \cup \mathcal{V})$$

and a basic graph pattern (BGP) is a conjunction of triple patterns. Given a BGP $B$, we denote by $\text{Var}(B)$ the set of variables occurring in $\text{Var}(B)$. Following the notation used in of [68], if $\text{Tr}_1, \text{Tr}_2, \ldots$ and $\text{Tr}_n$ are triple patterns, then we denote the BGP formed by $\text{Tr}_1, \text{Tr}_2, \ldots$ and $\text{Tr}_n$ with an expression having the following form

$$\text{Tr}_1 \text{ AND } \text{Tr}_2 \text{ AND } \ldots \text{ AND } \text{Tr}_n$$

The basic semantics of SPARQL BGPs relies on subgraph matching. Given a RDF graph $G$ and a BGP $B$, the triple patterns of $B$ are seen as nodes and edges of a graph $G(B)$, and finding answers to $B$ mainly consists in matching $G(B)$ to $G$, where variables in $G(B)$ are interpreted as wild cards. Each instantiation of the variables in $G(B)$ leading to match a subgraph of $G$ constitutes a solution.

**Definition 25** (BGP solution mapping) The evaluation of a SPARQL BGP $B$ over a graph $G$ consists in finding the set of all possible bindings of variables in $\text{Var}(B)$ to RDF terms in the queried graph. Such bindings are represented by functions $\mu : \text{Var}(B) \rightarrow \text{Terms}_{rdf}$, called solution mappings.

BGPs can be combined using the UNION operator to form query patterns, where the set of query patterns is inductively defined as follows:

- A BGP $B$ is a query pattern;
- If $Pt_1$ and $Pt_2$ are query patterns, then $Pt_1 \text{ UNION } Pt_2$ is a query pattern.

A SPARQL query is an expression of the form

```
SELECT $x_1 \ldots x_m$ WHERE { $QPT$ }
```
where $QPT$ is a query pattern and $x_1 \ldots x_m$ are variables occurring in $QPT$, called the **result variables** of the query, such that each $x_i$, for $i \in \{1, \ldots, m\}$, occurs in at least one of the BGPs forming $QPT$. Intuitively, the answers to a SPARQL query over a RDF graph $G$, consists of the projection over (possibly a subset of) the result variables, of all the solution mappings that is possible to obtain for the BGPs in $QPT$. Note that (i) each BGP may provide solution mappings involving only a subset of the distinguished variables of the query, (ii) the UNION operator simply combines the solutions obtained by evaluating the single BGPs [76].

A SPARQL **boolean query** is an expression of the form

$$\text{ASK}\{ QPT \}$$

where $QPT$ is a query pattern. A SPARQL boolean query $q$, when evaluated over a RDF graph $G$, simply returns true, if there exists a BGP $B$ in $QPT$ such that the set of solution mappings of $B$ w.r.t. $G$ is nonempty, otherwise it returns false.

### 4.6 Querying OWL 2 ontologies with SPARQL

Evaluating SPARQL queries by simple subgraph matching, as described in the previous Section, corresponds to interpret the queried RDF graph by the simple semantics introduced in Section 4.1.1 as all the knowledge represented by the queried graph is supposed to be explicitly given. However, if one wants to associate to a SPARQL query a more powerful notion of entailment, allowing for retrieving solution mappings that are implicitly contained in the queried graph, then she has to rely on the notion of SPARQL entailment regime. An entailment regime specify how the evaluation of BGPs has to be redefined in order to take into account knowledge implicitly following from the queried graph. Given that there exist two possible semantics to interpret OWL 2 ontologies and derive implicit knowledge from them, then, when using SPARQL to query a RDF graph $G$ corresponding to an OWL 2 ontology $O(G)$, one can choose between two entailment regimes corresponding to such two semantics, namely, the Direct Semantics Entailment Regime (DSER) and the RDF-Based Entailment Regime. As we are only interested in the theoretical logic-based semantics for OWL 2, in the following we only introduce DSER.

#### 4.6.1 The OWL 2 Direct Semantics Entailment Regime

The OWL 2 Direct Semantics Entailment Regime (DSER) specifies how solutions mappings must be computed, when evaluating SPARQL queries over OWL 2 ontologies interpreted by the Direct Semantics. Consequently in the context of DSER we can make the following assumptions [13]:

- The queried RDF graph must correspond to a OWL 2 DL ontology;
- The set of triple patterns composing each of the BGPs in the query pattern, must correspond to a set of OWL 2 DL query atoms, where a OWL 2 DL query atom has the same form of a OWL 2 DL axiom, with the difference that variables from a set can occur in place of entities and literals. The notion of type of position for variables and entities in OWL 2 query atoms, can be simply derived from the notion of type of position for entities in OWL 2 axioms given in Definition 16 of Section 4.2.3.
As a consequence, without loss of generality, in the context of DSER we can introduce a UCQ-like syntax for SPARQL queries that, used instead of the the RDF triples-based syntax given in the previous section, make our exposition much simpler and more intuitive.

A DSER BGP is an expression of the form

\[ At_1 \text{ AND } \ldots \text{ AND } At_n \]

where each \( At_i \) is an OWL 2 DL atom. A DSER BGP \( B \) is defined to be a legal DSER BGP for DSER, if every variable \( \$x \) occurring in \( B \) satisfies following condition called typing constraint of DSER:

- there is no couple \( At_1 \) and \( At_2 \) of atoms of \( B \) such that \( \$x \) occurs in \( At_1 \) and \( At_2 \) in positions of different types (for example, in individual position in \( At_1 \) and in class position in \( At_2 \)).

If \( B = At_1 \text{ AND } \ldots \text{ AND } At_n \) is a legal DSER BGP, \( Var(B) \) is the set containing all the variables occurring in \( B \), and \( O \) is a OWL 2 DL ontology defined over a vocabulary \( V_O \), then a solution mapping for \( B \) w.r.t. \( O \), is a total function \( \mu : Var(B) \rightarrow V_O \) such that \( O |= \mu(At_i) \), for \( i \in \{1, \ldots, n\} \).

A DSER query is an expression of the form

\[
\text{SELECT} \; \$x_1 \ldots \$x_m \; \text{WHERE} \; \{ \; B_1 \; \text{UNION} \; \ldots \; \text{UNION} \; B_n \; \}
\]

where every \( B_i \), for \( i \in \{1, \ldots, n\} \), is a legal DSER BGP, and every \( \$x_j \), for \( j \in \{1, \ldots, m\} \) is a result variable occurring in at least one \( B_i \). Given a DSER query \( Q \) with result variables \( \$x_1 \ldots \$x_m \) and a OWL 2 DL ontology \( O \), the answer of \( Q \) w.r.t. \( O \) consists of the projection over to obtain for the DSER BGPs in \( Q \).

A DSER boolean query is an expression of the form

\[
\text{ASK} \; \{ \; B_1 \; \text{UNION} \; \ldots \; \text{UNION} \; B_n \; \}
\]

where every \( B_i \), for \( i \in \{1, \ldots, n\} \), is a legal DSER BGP. A DSER boolean query \( Q \), when evaluated over a OWL 2 DL ontology \( O \), simply returns true, if there exists a DSER BGP \( B \) in \( Q \) such there is a solution mapping for \( B \) w.r.t. \( O \), otherwise it returns false.

At first glance, the class of DSER queries might seem very similar to the class of UCQs introduced in Section 2.3. Indeed, the syntax of both classes are based on similar notions of query atom, that is, an axiom where also variables can occur, and their semantics both relies on the notion of entailment of axioms w.r.t. to an ontology (or a knowledge base). However, a number of both syntactic and, above all, semantic differences exists between the two classes of queries:

- **Type of atoms**: BGP\( s \) in a DSER query can be formed by atoms obtained by allowing the occurrence of variables in both TBox and ABox OWL 2 DL axioms, whereas atoms of conjunctive queries can have the forms of unary and binary DL ABox assertions only;

- **Position of variables**: variables can occur in any position (class, individual, object property, etc.) in OWL 2 DL atoms, thus variables occurring in a DSER query can be
mapped to ontology predicates representing set of objects in the domain, whereas variables of UCQs can only range over domain objects. Moreover, variables in OWL 2 DL atoms can occur as arguments inside complex class (or object property, or data property, or data range) expressions of the language, whereas this is not possible for variables occurring in UCQs;

- **Pure existential variables**: non-distinguished variables of CQs are equivalent to pure existential variables of open FOL formulas, thus they are not requested to be assigned mandatorily to known individuals in the queried knowledge base \([2]\), whereas the definition of solution mapping given above, requests that all the variables occurring in DSER BGP must be mapped to known entities of the queried ontology;

- **Interpretation of "union"**: UCQs are defined as to be the logical union of FOL formulas, meaning that a boolean UCQ \(Q\), considered as a whole FOL formula, evaluates to \(true\) in every model of the queried knowledge base, whereas, in the case of DSER queries, the \(UNION\) operator of SPARQL simply combines the solution mappings that can be associated to DSER BGPs.

We now show, through two examples, how the different interpretation of non-distinguished variables and of "union", may impact on query answering.

**Example 26** Let \(K = (T, A)\) be the \(SROIQ\) knowledge base such that

\[
T = \{ \text{Person} \sqsubseteq \text{Man} \sqcup \text{Woman}, \ \text{Man} \sqsubseteq \neg \text{Woman} \}
\]

\[
A = \{ \text{Person}(\text{andrea}) \}
\]

and let \(Q\) the following union of boolean conjunctive queries over \(K\), where \(x\) is a variable

\[
Q() \leftarrow \text{Man}(x) \cup \text{Woman}(x)
\]

Let \(Mod(K)\) denote the set of all models of \(K\). \(Mod(K)\) can be split in two disjoint subsets \(Mod_m(K)\) and \(Mod_w(K)\) as follows:

- \(Mod_m(K)\) contains all the models \(M_m\) of \(K\) such that \(M_m \models \text{Man}(\text{andrea})\);

- \(Mod_w(K)\) contains all the models \(M_w\) of \(K\) such that \(M_w \models \text{Woman}(\text{andrea})\).

So, for every \(M_m\) in \(Mod_m(K)\) there exists a query homomorphism for the disjunct \(\text{Man}(x)\) of \(Q\), and for every \(M_w\) in \(Mod_w(K)\) there exists a query homomorphism for the disjunct \(\text{Woman}(x)\) of \(Q\). Thus, \(K \models Q\), and, consequently, the evaluation of \(Q\) over \(K\) returns \(true\) as answer.

Now, let \(O\) be the OWL 2 DL ontology formed by the following three axioms

\[
\text{SubClassOf}(\text{ex:Person} \ \text{ObjectUnionOf}(\text{ex:Man} \ \text{ex:Woman}))
\]

\[
\text{DisjointClasses}(\text{ex:Man} \ \text{ex:Woman})
\]

\[
\text{ClassAssertion}(\text{ex:Man} \ \text{ex:andrea})
\]

and let \(Q_{dser}\) be the following DSER query over \(O\), where $x$ denotes a variable
4.6 Querying OWL 2 ontologies with SPARQL

\[ \text{ASK} \{ \text{ClassAssertion(ex:Man } \$x) \cup \text{ClassAssertion(ex:Woman } \$x) \} \]

The reader can easily verify that, when interpreted by the Direct Semantics, is semantically equivalent to \( K \), as it is possible to show that, given a model \( M \) of \( K \), one can always build a direct model \( M' \) of \( O \), and viceversa. Let \( Mod(O) \) denotes the set of all direct models of \( O \). Like \( Mod(K) \), \( Mod(O) \) can be split in two disjoint subsets \( Mod_m(O) \) and \( Mod_w(O) \) as follows:

- \( Mod_m(O) \) contains all the direct models \( M_m \) of \( O \) such that \( M|_{\text{DS}} \text{ClassAssertion(ex:Man ex:Andrea)} \);
- \( Mod_w(O) \) contains all the direct models \( M_w \) of \( O \) such that \( M|_{\text{DS}} \text{ClassAssertion(ex:Woman ex:Andrea)} \).

Thus, one might expect that, similarly to the case of \( Q \) and \( K \), \( Q_{dser} \) evaluates to \text{true} over \( O \). However, this is not the case, and actually \( Q_{dser} \) evaluates to \text{false} over \( O \). The reason is that there not exists a DSER BGP \( B \) in \( Q_{dser} \) for which a solution mapping exists. Indeed, the mapping \( \mu : \$x \rightarrow \text{ex:Andrea} \) is such that the two axioms \( \mu(\text{ClassAssertion(ex:Man } \$x)) \) and \( \mu(\text{ClassAssertion(ex:Woman } \$x)) \) are satisfied by only half of the direct models of \( O \).

Example 27 Let \( K = (T, A) \) be the SROIQ knowledge base such that

\[ T = \{ \text{Person} \sqsubseteq \exists \text{hasFather}.\text{Man} \} \]
\[ A = \{ \text{Person}(\text{andrea}) \} \]

and let \( Q \) the following union of boolean conjunctive queries over \( K \), where \( x \) is a variable, asking if an instance \( x \) of the concept \text{Man} exists

\[ Q() \leftarrow \text{Man}(x) \]

Let \( Mod(K) \) denote the set of all models of \( K \). Every \( M = (\Delta^M, \cdot^M) \) in \( Mod(K) \) is such that \( (\text{andrea})^M \in (\text{Person})^M \), and there exists an element \( i \) in \( \Delta^M \) such that (i) \( ((\text{andrea})^M, i) \in (\text{hasFather})^M \), and (ii) \( i \in (\text{Man})^M \). Consequently, for every \( M \) in \( Mod(K) \) there exists a query homomorphism from \( Q \) to \( M \), and the evaluation of \( Q \) over \( K \) returns true as answer.

Now, let \( O \) be the OWL 2 DL ontology formed by the following two axioms

\[ \text{SubClassOf(ex:Person ObjectSomeValuesFrom(ex:hasFather ex:Man))} \]
\[ \text{ClassAssertion(ex:Person ex:Andrea)} \]

and let \( Q_{dser} \) be the following DSER query over \( O \), where \( \$x \) denotes a variable, asking if an instance \( \$x \) of the class \text{ex : Man} exists

\[ \text{ASK} \{ \text{ClassAssertion(ex:Man } \$x) \} \]
The reader can easily verify that $\mathcal{O}$, when interpreted by the Direct Semantics, is semantically equivalent to $\mathcal{K}$, as it is possible to show that, given a model $\mathcal{M}$ of $\mathcal{K}$, one can always build a direct model $\mathcal{M}'$ of $\mathcal{O}$, and viceversa. Let $\text{Mod}(\mathcal{O})$ denotes the set of all direct models of $\mathcal{O}$. Every $\mathcal{M} = (\Delta_o, \Delta_v, C, \cdot OP, \cdot DP, \cdot I, \cdot DT, \cdot LT, \cdot FA, \text{NAMED})$ in $\text{Mod}(\mathcal{O})$ is such that $(\text{ex: andrea})^I \in (\text{ex: Person})^C$, and there exists an element $i$ in $\Delta_o$ such that (i) $((\text{ex: andrea})^I, i) \in (\text{ex: hasFather})^{OP}$, and (ii) $i \in (\text{ex: Man})^C$. Thus, one might expect that, similarly to the case of $\mathcal{Q}$ and $\mathcal{K}$, $Q_{dser}$ evaluates to $\text{true}$ over $\mathcal{O}$. However, this is not the case, and actually $Q_{dser}$ evaluates to $\text{false}$ over $\mathcal{O}$. The reason is that there not exists any entity $e$ occurring in $\mathcal{O}$, such that $(e)^I = i$, thus no solution mapping exists for the DSER BGP $\text{ClassAssertion(ex:Man \$x\)}$.

The two above examples show that, in general, answering union of conjunctive queries over $\text{SROIQ}$ KBs is more difficult then answering SPARQL DSER queries over OWL 2 DL ontologies. Indeed, if $Q$ is a SPARQL DSER query over a OWL 2 DL ontology $\mathcal{O}$, and $Q$ is formed by the DSER BGPs $B_1, \ldots, B_n$, an algorithm for computing the answers of $Q$ w.r.t. $\mathcal{O}$ is simply based on testing, for every possible solution mapping $\mu$, if $\mathcal{O} \models \mu(B_i)$ for some $i \in \{1, \ldots, n\}$. By the fact that only entities and literals occurring in $\mathcal{O}$ can be in the range of a mapping $\mu$ for $Q$ over $\mathcal{O}$, the number of solution mappings that such an algorithm has to take into account is finite. However, the number of solution mappings that have to be checked can be exponential in the number of variables occurring in each DSER BGP, thus it is unlikely that such a naive algorithm can be used in real world scenarios. To overcome such an issue, in [52] authors propose an optimization strategy based on determining a good execution order for atoms in BGPs by considering the selectivity of the atoms. However, motivated by the fact that, by evaluating SPARQL queries under DSER one cannot fully exploit the knowledge residing in the queried ontology, many researchers in the last years started investigating the problems of enlarging the SPARQL semantics with stronger reasoning capabilities [14, 53, 8, 54, 3], and of dropping the active domain restriction of SPARQL thus enabling the definition queries where pure existential variables occur and consequently reconciling SPARQL solution mappings with the usual notion of certain answers [39, 41, 4].
Part II

A semantics for metamodelling in the semantic web
Chapter 5

Motivating scenario

In this chapter we motivate our work aiming at adding proper metamodeling capabilities to OWL 2. In Section 5.1, we present a real-world OBDM scenario needing metamodeling features, and we show in Section 5.2 that the limited metamodeling capabilities of OWL 2 DL are not sufficient to match such needs.

5.1 A real world OBDM scenario requiring metamodeling

In this section we provide an example of a real-world scenario where metamodeling is necessary in order to capture interesting aspects of the domain of interest. We refer to the Ontology of the Italian Public Debt, that has been designed in a joint project by the Italian Ministry of Economy and Finance [7]. One of the central elements of such an ontology is the class "financial instrument", which is an abstraction for any contract that gives rise to a financial asset of one entity and a financial liability of another entity. Several subclasses of "financial instrument" are of interest in the domain. Two examples of such subclasses are "Buono del Tesoro Poliennale" (or simply "BTP"), which is the Italian bond considered as the benchmark for measuring the Italian spread, and "Commercial paper", which is a money-market security issued to obtain funds to meet short-term debt obligations.

Users are interested in various properties of financial instruments, including the duration and the currency. For example, the financial instrument denoted by the IRI "IT0005069395" is an instance of "BTP", and has 6 months duration, while "ZAG000117292" is an instance of "Commercial paper", and has 18 months duration.

| SubClassOf(:BTP :Financial_instrument) |
| SubClassOf(:Commercial_paper :Financial_instrument) |
| SubClassOf(:Financial_instrument DataSomeValuesFrom(:duration)) |
| DisjointClasses(:BTP :Commercial_paper) |
| ClassAssertion(:BTP :IT0005069395) |
| ClassAssertion(:Commercial_paper :ZAG000117292) |
| DataPropertyAssertion(:duration :IT0005069395 "6"ˆxsd:integer) |
| DataPropertyAssertion(:duration :ZAG000117292 "18"ˆxsd:integer) |

Table 5.1. OWL 2 QL Italian Public Debt ontology with no metamodeling features

The scenario described so far is captured by the OWL 2 QL ontology depicted in Table 5.1.
Note that the ontology is a classical “first-order” ontology, with a strict separation between individuals (e.g., :IT0005069395, and :ZAG000117292), and predicates (e.g., :BTP, and :Commercial_paper). Indeed, the characteristics of the domain described above do not require any metamodeling features, and therefore can be formalized by using classical “first-order” constructs. However, we now show that if we restrict our attention to first-order logic fragments, there are many interesting aspects of the domain that we have to exclude from modeling. As an example, consider the case of the class “BTP”. Such class of financial instruments has been established by the legislative decree “dr_144bis”, that formally specifies the set of rules governing it. How can we model this fact in our ontology? To answer this question, we need to introduce an object property, say “established_by”, and use such relation to connect “BTP” to ”dr_144bis”. The key point here is that what we are connecting to ”dr_144bis” is the class “BTP”, not its specific instances. In other words, we clearly need to treat “BTP” as an individual, whose properties can be described and asserted. So, we are really entering the realm of metamodeling, since “BTP” (and, similarly, all other subclasses of “Financial instrument”) should be treated as a polymorphic entity, playing the twofold role of an individual and a class, simultaneously. Syntactically, this is possible in OWL 2 QL, by virtue of punning, the OWL 2 characteristic allowing the use of the same name in positions of different “type”, e.g., in individual and class position.

Now that “BTP” plays the role of an individual (besides that of a class), we can introduce a new class (actually, a metaclass), called “Type of financial instrument”, and make “BTP” (together with all subclasses of “Financial instrument”) one of its instances. We can further note that the instances of “Type of financial instrument” can be classified based on various criteria, e.g., in individual and class position.

1. whether they are issued on the Italian market or not,
2. the way they pay interests to the holders.

To capture the classification behind criterion[1] we can introduce two new classes, namely “Type of Italian financial instrument” and “Type of foreign financial instrument”, and define them as subclasses of “Type of financial instrument”. Analogously, to capture the classification behind criterion[2] we can introduce two new subclasses of “Type of financial instrument”, namely “Type of financial instrument with coupon” and “Type of financial instrument without coupon”.

It turns out that each of such subclasses has properties that are worth to be considered. For example, each of them is characterized by a set of “Management best practices” specifying the countermeasures (e.g., increase or decrease in the interest rates) to take in order to react to events that may possibly occur on the financial and monetary markets. In order to model this phenomenon, we introduce the class “Category of type of financial instrument”, the class “Management best practice”, and the object property “managed by means of”, and we assert that

• the domain of “managed by means of” is the class “Category of type of financial instrument”;
• the range of “managed by means of” is the class “Management best practice”;
• the classes “Type of financial instrument with coupon” and “Type of financial instrument without coupon” are both instances of “Category of type of financial instrument”,
5.1 A real world OBDM scenario requiring metamodeling

- the individual “bp for types with coupon” is an instance of the class “Management best practice”, and

- “Type of financial instrument with coupon” is linked to “bp for types with coupon” by means of the object property “managed by means of”.

To highlight the chain of the `instanceOf` relation included in the ontology, we provide in Figure 5.1 a pictorial representation of a fragment of the ontology, in the form of a semantic network. The resulting ontology expressed in `OWL 2 QL` is shown in Table 5.2.

**Figure 5.1.** Chain of `instanceOf` relations between entities in the Italian Public Debt domain

Obviously, the benefit of using metamodeling is greatly limited if one cannot express metaqueries to extract data spanning from several levels of the `instanceOf` relation in the ontology. The simplest form of metaquery is an expression where variables appear in class or property positions. More interesting forms of metaqueries allow one to extract complex patterns from the ontology, by allowing variables to appear simultaneously in individual object and class or property positions. For example, within the domain described by the Ontology of the Italian Public Debt, one might be interested in asking for all financial instruments along with the law ruling the types they are instances of. This can be expressed through the following SPARQL query:

```sparql
select $x $y
WHERE{
  ClassAssertion($z $x).
  ObjectPropertyAssertion(:ruled_by $z $y)}
```

As a further example of metaquery, suppose we are interested in retrieving the financial instruments, together with their duration and the law which established the types they are instance of, that belong to a type which is disjoint with the type "Commercial paper" and which can be classified with respect to both the financial market of issuance and the payment mode. This query actually traverses three levels of the `instanceOf` relation:
5. Motivating scenario

```owl
subClassOf(:BTP :Financial_instrument)
subClassOf(:Commercial_paper :Financial_instrument)
subClassOf(:Type_of_f_i :Type_of_f_i)
subClassOf(:Type_of_f_i :Type_of_f_i)
subClassOf(:duration)
subClassOf(:duration)
subClassOf(:duration)
subClassOf(:duration)
subClassOf(:duration)
subClassOf(:managed_by_means_of)
subClassOf(:managed_by_means_of)
subClassOf(:Category_of_type_of_f_i_wrt_pay_mode)
subClassOf(:Category_of_type_of_f_i_wrt_market)
subClassOf(:Established_by :Law)
subClassOf(:managed_by_means_of :Category_of_type_of_f_i)
subClassOf(:managed_by_means_of :Category_of_type_of_f_i)
subClassOf(:managed_by_means_of :Category_of_type_of_f_i)
subClassOf(:managed_by_means_of :Category_of_type_of_f_i)
subClassOf(:managed_by_means_of :Category_of_type_of_f_i)
subClassOf(:Established_by :Law)
subClassOf(:managed_by_means_of :Category_of_type_of_f_i)
```

<table>
<thead>
<tr>
<th>Table 5.2. OWL 2 QL Italian Public Debt ontology with metamodeling features</th>
</tr>
</thead>
<tbody>
<tr>
<td>select $fi $dr $lw where</td>
</tr>
<tr>
<td>ClassAssertion(:Category_of_type_of_f_i_wrt_pay_mode $cat_1) .</td>
</tr>
<tr>
<td>ClassAssertion(:Category_of_type_of_f_i_wrt_market $cat_2) .</td>
</tr>
<tr>
<td>DisjointClasses(:Commercial_paper $tp) .</td>
</tr>
<tr>
<td>ObjectPropertyAssertion(:established_by $tp $lw) .</td>
</tr>
<tr>
<td>ClassAssertion($cat_1 $tp) . ClassAssertion($cat_2 $tp) .</td>
</tr>
<tr>
<td>ClassAssertion($tp $fi) .</td>
</tr>
<tr>
<td>DataPropertyAssertion(:duration $fi $dr)</td>
</tr>
</tbody>
</table>

5.2 Metamodeling and metaquerying in OWL 2 standard semantics

The most powerful OWL 2 language is OWL 2 Full[1] which is an extension of RDF Schema and, like RDF Schema, does not enforce a strict separation of classes, properties, individuals and data values. So, metamodeling is taken for granted in OWL 2 Full. In[61], a landmark paper in OWL 2 metamodeling, it is shown, however, that one of the reasons why reasoning in OWL 2 Full is undecidable is the power of its metamodeling features. With the goal of achieving decidability, both the Description Logic and the Semantic Web community concentrated their attention on OWL 2 DL, that is essentially a first-order language, with no explicit semantic support for metamodeling. Indeed, virtually all systems developed in the context of Description Logics and the Semantic Web support ontologies are constrained by the restrictions required for OWL 2 DL, and the Direct Semantics for OWL 2 advocates the interpretation of ontologies as first-order theories, which in fact are not suitable for formalizing polymorphism of objects as they impose a sharp distinction between the entities

---

[1]Following the same line of [61], we call OWL 2 Full the language obtained by interpreting OWL 2 ontologies by the RDF-based semantics
denoting predicates and those denoting individual objects. Hence, if the ontology makes use of “punning”, under DS, two occurrences of the same name in positions of different types are actually treated as if they were denoting different elements.

For example, let $O$ denote the ontology depicted in Table 5.1. Note that $O$ is a legal OWL 2 DL ontology as it enforces all the syntactic restrictions described in Section 4.2.4. Consider now the following two axioms contained in $O$:

1. ClassAssertion(:BTP :IT0005069395)

2. ClassAssertion(:Type_of_f_i_with_coupon :BTP)

Note that the entity identified by the IRI :BTP occurs in class position in axiom 1 whereas occurs in individual position in axiom 2. Even though the same identifier is used in both the axioms, the DS is not able to recognize that a semantic correlation exists between the two occurrences of :BTP. In fact, the ontology that can be interpreted by the DS does not corresponds to $O$, but to a slightly modified version of it where the above axioms are substituted by the following two:

3. ClassAssertion(:BTP$^{cl}$ :IT0005069395)

4. ClassAssertion(:Type_of_f_i_with_coupon :BTP$^{ind}$)

where :BTP is substituted by a fresh new identifier :BTP$^{cl}$ in axioms where it occurs in class position, whereas it is substituted by and :BTP$^{ind}$ in axioms where it occurs in individual position. Thus at is point it is evident that OWL 2 DL, despite the syntactic support offered by “punning”, is not able to correctly manage metamodeling. This is confirmed by the fact that according to the standard specification of the SPARQL OWL 2 Direct Semantics Entailment regime none of the two queries that have been described in the previous subsection is legal. Indeed, such a regime imposes typing constraints by which no variable can be used both in individual and in class position.

Some approaches for extending OWL 2 DL with multi-level metamodeling features are based on the definition of two or more ontologies (one for each meta-layer), with the idea of keeping the various levels separated. Note, however, that this may result in limited expressive power in both modeling metaclasses, and expressing metaquerying, exactly because of the impossibility of mixing the various levels. Also, the approach requires to put in place synchronization mechanisms between ontologies at different level, which can be costly. An approach based on stratification is presented in [67, 48], where the authors propose a language embedding the notion of multiple meta-layers by allowing the definition of one vocabulary for each stratum and by providing suitable meta-predicates for establishing relations between entities at adjacent strata. Again, such stratification poses challenges for the modeler and can make metaquerying cumbersome. In [38] the authors propose a method to axiomatize the higher-order knowledge on metaclasses into special first-order assertions, but this translation process is not modular, in the sense that it has to be repeated every time a new predicate is added to (or removed from) the ontology. Moreover, the axiomatization
process itself requires the use of complex class expressions that are a source of complexity for reasoning. More recent proposals such as those in [62, 55, 16] are also based on the idea of using complex class expressions so as to reduce reasoning about metamodeling features into reasoning on first-order constructs.
Chapter 6

Adding proper metamodeling capabilities to OWL 2

In this chapter we present our proposal for languages enabling proper metamodeling and metaquerying in the context of the semantic web. In Section 6.1 we present the language Hi(OWL 2), whose syntax is the same as OWL 2 but whose semantics allows for proper metamodeling. Then in Section 6.1 we present the language Hi(OWL 2 QL) as a fragment of Hi(OWL 2). Finally, in Section 6.3 we introduce our query language.

6.1 The language Hi(OWL 2)

In this section we define Hi(OWL 2) as the language which have the same syntax of OWL 2, where the same syntactic restrictions on legal ontologies of OWL 2 DL are enforced, and which interpret ontologies by using a semantics, called MetaModeling Semantics (MMS), that is a Hilog style semantics [27] that has been originally introduced in [58], and allows overcoming the problems with metamodeling of the standard semantics of OWL 2.

6.1.1 Syntax of Hi(OWL 2)

As for the syntax, we simply adopt the standard syntax of OWL 2. This means that syntactic notions of entities (also known as atomic expressions), complex expressions, literals, anonymous individuals, axioms, and ontologies for Hi(OWL 2) coincide with the corresponding notions introduced for OWL 2 in Section 4.2. The notion of type of position of atomic expressions in axioms and complex expressions is introduced by the Definition 16. Moreover, Hi(OWL 2) enforces the same set of restrictions imposed by OWL 2 DL, and introduced in Section 4.2.4. Therefore, every set of OWL 2 axioms forming a legal OWL 2 DL ontology, forms a legal Hi(OWL 2) ontology as well.

6.1.2 Semantics of Hi(OWL 2)

We now describe the MMS semantics for OWL 2 DL ontologies. As for interpretation of datatypes and literals we relies on the notion of OWL 2 datatype map introduced by Definition 17 in Section 4.3.1.

1 Note that in [58] such a semantics is called HOS.
Following the same line of the standard specification, we now introduce the notion of vocabulary of a Hi(OWL 2) ontology. Given a Hi(OWL 2) ontology \( O \) and a OWL 2 datatype map \( DM = (N_{DT}, N_{LS}, N_{FS}, D^T, L^T, F^S) \), the vocabulary \( V_O \) of \( O \) is constituted by the tuple

\[
\]

where \( V^O_N \) is the set containing all the IRIs occurring in \( O \) and:

- \( V^O_C \) is the superset of \{owl:Thing,owl:Nothing \} containing all the IRIs occurring in class position in \( O \);
- \( V^O_{OP} \) is the superset of \{owl:topObjectProperty,owl:bottomObjectProperty \} containing all the IRIs occurring in object property position in \( O \);
- \( V^O_{DP} \) is the superset of \{owl:topDataProperty,owl:bottomDataProperty \} containing all the IRIs occurring in data property position in \( O \);
- \( V^O_I \) is the set containing all the IRIs occurring in along with all the anonymous individual occurring in \( O \);
- \( V^O_{DT} \) is the superset of \{rdfs:Literal \} \( \cup \) \( N_{DT} \) containing all the IRIs occurring in data range position in \( O \);
- \( V^O_{FA} \) is the set of pairs \((F,lt)\) such that the literal \( lt \) satisfies the constraining facet \( F \) according to \( DM \);
- \( L^O \) is the set containing all the literals occurring in \( O \).

Let \( R_{owl} \) and \( D_{owl} \) be the sets of IRIs defined at the beginning of Section 4.2.4 We observe that the syntactic rules of OWL 2 DL imply impose that the following conditions hold on \( V_O \):

- \( V^O_C \cap V^O_{DT} = \emptyset \);
- \( V^O_{OP} \cap V^O_{DP} = \emptyset \);
- \( V^O_C \cap R_{owl} = \{owl:Thing,owl:Nothing\} \);
- \( V^O_{OP} \cap R_{owl} = \{owl:topObjectProperty,owl:bottomObjectProperty\} \);
- \( V^O_{DP} \cap R_{owl} = \{owl:topDataProperty,owl:bottomDataProperty\} \);
- \( V^O_I \cap R_{owl} = \emptyset \);
- \( V^O_{DT} \cap R_{owl} \subseteq D_{owl} \);

Also, given the vocabulary \( V_O = (V^O_N, V^O_C, V^O_{OP}, V^O_{DP}, V^O_I, V^O_{DT}, V^O_{FA}, L^O) \) of an ontology \( O \), we denote with \( Exp^O \) the set of all well-formed expressions that can be built on the basis of \( V_O \), defined as

\[
Exp^O = V^O_N \cup V^O_I \cup Exp^C \cup Exp^OP \cup Exp^DP \cup Exp^DR
\]

where:
6.1 The language Hi (OWL 2)

- \( \text{Exp}_C^O \), the set of well-formed class expressions, is the set obtained as the union of the following sets:
  - \( V_C \);
  - \{ \text{ObjectSomeValuesFrom}(e_1 e_2) \mid e_1 \in \text{Exp}_C^{OP}, e_2 \in \text{Exp}_C^O \};
  - \{ \text{ObjectAllValuesFrom}(e_1 e_2) \mid e_1 \in \text{Exp}_C^{OP}, e_2 \in \text{Exp}_C^O \};
  - \{ \text{DataSomeValuesFrom}(e_1 e_2) \mid e_1 \in V_{DP}, e_2 \in \text{Exp}_D^{DR} \};
  - \{ \text{DataAllValuesFrom}(e_1 e_2) \mid e_1 \in V_{DP}, e_2 \in \text{Exp}_D^{DR} \};
  - \{ \text{ObjectComplementOf}(e) \mid e \in \text{Exp}_C^O \};
  - \{ \text{ObjectIntersectionOf}(e_1 e_2 \ldots e_n) \mid e_i \in \text{Exp}_C^O \};
  - \{ \text{ObjectUnionOf}(e_1 e_2 \ldots e_n) \mid e_i \in \text{Exp}_C^O \};
  - \{ \text{ObjectOneOf}(e_1 e_2 \ldots e_n) \mid e_i \in V_I^O \};
  - \{ \text{ObjectHasValue}(e_1) \mid e_1 \in \text{Exp}_V^{OP}, e_2 \in V_I^C \};
  - \{ \text{DataHasValue}(e) \mid e \in \text{Exp}_V^{OP}, e \in L_O \};
  - \{ \text{ObjectHasSelf}(e) \mid e \in \text{Exp}_V^{OP} \};
  - \{ \text{ObjectMinCardinality}(e_1 e_2) \mid e_1 \in \text{Exp}_V^{OP}, e_2 \in \text{Exp}_V^C \};
  - \{ \text{ObjectMaxCardinality}(e_1 e_2) \mid e_1 \in \text{Exp}_V^{OP}, e_2 \in \text{Exp}_V^C \};
  - \{ \text{ObjectExactCardinality}(e_1 e_2) \mid e_1 \in \text{Exp}_V^{OP}, e_2 \in \text{Exp}_V^C \};
  - \{ \text{DataMinCardinality}(e_1 e_2) \mid e_1 \in V_{DP}, e_2 \in \text{Exp}_V^C \};
  - \{ \text{DataMaxCardinality}(e_1 e_2) \mid e_1 \in V_{DP}, e_2 \in \text{Exp}_V^C \};
  - \{ \text{DataExactCardinality}(e_1 e_2) \mid e_1 \in V_{DP}, e_2 \in \text{Exp}_V^C \}.

- \( \text{Exp}_D^{DR} \), called the set of well-formed data range expressions is the set obtained as the union of the following sets:
  - \( V_{DT}^O \);
  - \{ \text{DataIntersectionOf}(e_1 e_2 \ldots e_n) \mid e_i \in \text{Exp}_D^{DR}, i \in \{1, \ldots, n\} \};
  - \{ \text{DataUnionOf}(e_1 e_2 \ldots e_n) \mid e_i \in \text{Exp}_D^{DR}, i \in \{1, \ldots, n\} \};
  - \{ \text{DataComplementOf}(e) \mid e \in \text{Exp}_D^{DR} \};
  - \{ \text{DataOneOf}(e_1 e_2 \ldots e_n) \mid e_i \in L_O \};
  - \{ \text{DatatypeRestriction}(e) \mid e \in V_D^O, e \in L_O, (F_i, l_i) \in V_F^A \}.

- \( \text{Exp}_D^O = V_O^P \cup \{ \text{ObjectInverseOf}(e_1) \mid e_1 \in V_O^P \} \) is the set of well-formed object property expressions;

- \( \text{Exp}_D^{DP} = V_{DP} \) is the set of well-formed data property expressions.

Note that \( \text{Exp}^O \) is infinite for every legal ontology \( O \). Indeed, for example, as \( V_C^O \) and \( V_O^P \) are nonempty by definition, \( \text{Exp}_C^O \) always contains an infinite sequence of class expressions, involving only the IRIs \text{owl:Thing} and \text{owl:topObjectProperty}, of the form \text{ObjectSomeValuesFrom}, obtained by recursively using as second argument in class position a new \text{ObjectSomeValuesFrom} expression.

The MMS semantics is based on the notion of interpretation structure, which plays the role of “interpretation domain” in classical first-order logic.
Definition 28  An interpretation structure is a tuple $\Sigma = (\Delta_o, \Delta_v, I, E, R, A, T)$ where:

- $\Delta_o$, the object domain, is a non-empty set disjoint with $\Delta_v$;
- $\Delta_v$ is the value domain. All the value spaces of the datatypes in $\mathcal{D}'_{owl}$ are contained in $\Delta_v$;
- $E : \Delta_o \to \mathcal{P}(\Delta_o)$ is a partial function;
- $R : \Delta_o \to \mathcal{P}(\Delta_o \times \Delta_o)$ is a partial function;
- $A : \Delta_o \to \mathcal{P}(\Delta_o \times \Delta_v)$ is a partial function;
- $T : \Delta_o \to \mathcal{P}(\Delta_v)$ is a partial function;
- $I : \Delta_o \to \{\text{true}, \text{false}\}$ is a total function such that for each $d \in \Delta_o$, if $E, R, A, T$ are all undefined for $d$, then $d I = \text{true}$.

Thus, the interpretation structure is not simply a set, but a mathematical representation of a world made up by elements which have complex inter-relationships, represented by the various functions making up $\Sigma$.

- Each element in the world represented by $\Sigma$ is in $\Delta = \Delta_o \cup \Delta_v$, and therefore is either an object or a value.
- $E$ is a function that, given a domain object $d$, either assigns to $d$ its extension as a class, i.e., a set of objects, or is undefined for $d$, reflecting that $d$ is not regarded as a class in the world represented by $\Sigma$.
- $R$ is a function that, given a domain object $d$, either assigns to $d$ its extension as an object property, i.e., a set of object pairs, or is undefined for $d$, reflecting that $d$ is not regarded as an object property in the world represented by $\Sigma$.
- $T$ is a function that, given a domain object $d$, either assigns to $d$ its extension as a datatype, i.e., a set of values, or is undefined for $d$, reflecting that $d$ is not regarded as a datatype in the world represented by $\Sigma$.
- $A$ is a function that, given a domain object $d$, either assigns to $d$ its extension as a data property, i.e., a set of object-value pairs, or is undefined for $d$, reflecting that $d$ is not regarded as a data property in the world represented by $\Sigma$.
- $I$ is the boolean function specifying whether an object $d$ is regarded as an individual object (if $d I$ is true), or not (if $d I$ is false) in the world represented by $\Sigma$. Note that $I$ is crucial for making MMS compatible with DS, based on the idea that the interpretation domain of DS interpretations corresponds to $\Delta_v \cup \Delta_i$, where $\Delta_i$ is the (nonempty) subset of $\Delta_o$ defined as $\{d \in \Delta_o \mid d I = \text{true}\}$.

Hence, the interpretation structure makes it clear that elements in $\Delta_o$ are allowed being polymorphic, in the sense that each $d$ in $\Delta_o$ can simultaneously be seen as an individual object (this is the case where function $d I = \text{true}$), a set of objects (this is the case where $d E$ is defined), a binary relation between objects (this is the case where $d R$ is defined),
Figure 6.1. Interpretation structure

6.1 The language Hi (OWL 2)

binary relation between objects and values (this is the case where \( d^A \) is defined), and a set of values (this is the case where \( d^T \) is defined).

Figure 3 shows a representation of an interpretation structure, where the element denoted by \( \alpha \) is simultaneously an individual object (i.e., is marked as \text{true}), a set of objects and a binary relation between objects, whereas \( \beta \) is a set of objects, but not an individual object (i.e., is marked as \text{false}) or a binary relation.

Given an ontology \( O \), an interpretation \( I \) for \( O \) is a pair, \( \langle \Sigma, I_o \rangle \), where \( \Sigma \) is an interpretation structure, and \( I_o \) is the interpretation function for \( I \), i.e., a function that maps every expression and anonymous individual in \( \text{Exp}^O \) into an object in \( \Delta_o \), every literal in \( \text{LO} \) into a value in \( \Delta_v \), and every pair \( (F,lt) \) \( V_{FA}^O \) into a subset of \( \Delta_v \) according following set of conditions, where \( \Delta_i \) is the (nonempty) subset of \( \Delta_o \) defined as \( \{ d \in \Delta_o | d^I = \text{true} \} \), \( \Delta_i^{\text{named}} \) is the subset of \( \Delta_i \) such that \( e^{I_o} \in \Delta_i^{\text{named}} \) for every \( e \in V_I^O \) and \#B denotes the cardinality of a set B:

- for \( e \in \text{R}_{\text{out}} \): \( (e^{I_o})^I = \text{false}; \)
- for \( e \in \text{Exp}^O_{\text{C}} \): \( (e^{I_o})^E \) is defined, and \( (e^{I_o})^T \) is undefined;
- for \( e \in \text{Exp}^O_{\text{OP}} \): \( (e^{I_o})^R \) is defined, and \( (e^{I_o})^A \) is undefined;
- for \( e \in \text{V}_{\text{DP}} \): \( (e^{I_o})^A \) is defined, and \( (e^{I_o})^R \) is undefined;
- for \( e \in \text{Exp}^O_{\text{DR}} \): \( (e^{I_o})^T \) is defined, \( (e^{I_o})^E \) is undefined;
- for \( e \in V_I^O \): \( (e^{I_o})^I = \text{true}; \)
- for \( e \in \text{D}_{\text{out}}' \): \( (e^{I_o})^T = e^{DT}; \)
- for \( lt = (l,d) \in \text{N}_{LS} \): \( lt^{I_o} = (l,d)^{LS}; \)
- for \( (F,lt) \in V_{FA}^O \): \( (F,lt)^{I_o} = (F,(lt)^{I_o})^{FS}; \)
- \( (\text{owl:Thing}^{I_o})^E = \Delta_i; \)
6. Adding proper metamodeling capabilities to OWL 2

- (owl:Nothing)_{T_o}^E = \emptyset;
- (owl:topObjectProperty)_{T_o}^R = \Delta_i \times \Delta_i;
- (owl:bottomObjectProperty)_{T_o}^R = \emptyset;
- (owl:topDataProperty)_{T_o}^A = \Delta_i \times \Delta_v
- (owl:bottomDataProperty)_{T_o}^A = \emptyset;
- (rdfs:Literal)_{T_o}^T = \Delta_v;
- (objectInverseOf)_{T_o}^R = \{ \langle d_1, d_2 \rangle \mid \langle d_2, d_1 \rangle \in (e_{T_o}^R) \};
- (objectSomeValuesFrom)_{T_o}^E = \{ d \mid \langle d, d_1 \rangle \in (e_{T_o}^R), d_1 \in (e_{T_o}^E) \};
- (objectAllValuesFrom)_{T_o}^E = \{ d \mid \forall d_1. \langle d, d_1 \rangle \in (e_{T_o}^R) \implies d_2 \in (e_{T_o}^E) \};
- (dataSomeValuesFrom)_{T_o}^E = \{ d \mid \langle d, v \rangle \in (e_{T_o}^A), v \in (e_{T_o}^T) \};
- (dataAllValuesFrom)_{T_o}^E = \{ d \mid \forall v. \langle d, v \rangle \in (e_{T_o}^A) \implies v \in (e_{T_o}^T) \};
- (objectIntersectionOf)_{T_o}^E = (e_{T_o}^E) \cap \ldots \cap (e_{T_o}^E);
- (objectUnionOf)_{T_o}^E = (e_{T_o}^E) \cup \ldots \cup (e_{T_o}^E);
- (objectOneOf)_{T_o}^E = \{(e_1)_{T_o}, \ldots, (e_n)_{T_o}\}
- (objectComplementOf)_{T_o}^E = \Delta_i \setminus (e_{T_o}^E);
- (objectHasValue)_{T_o}^E = \{ ind_1 \mid \langle ind_1, e_{T_o}^A \rangle \in (e_{T_o}^R) \};
- (objectHasSelf)_{T_o}^E = \{ d \mid \langle d, d \rangle \in (e_{T_o}^R) \};
- (objectMinCardinality)_{T_o}^E = \{ d \mid \#\{d_i \mid \langle d, d_i \rangle \in (e_{T_o}^R), d_i \in (e_{T_o}^E) \} \geq k \};
- (objectMaxCardinality)_{T_o}^E = \{ d \mid \#\{d_i \mid \langle d, d_i \rangle \in (e_{T_o}^R), d_i \in (e_{T_o}^E) \} \leq k \};
- (objectExactCardinality)_{T_o}^E = \{ d \mid \#\{d_i \mid \langle d, d_i \rangle \in (e_{T_o}^R), d_i \in (e_{T_o}^E) \} = k \};
- (dataHasValue)_{T_o}^E = \{ d \mid \langle ind, \llt \rangle_{T_o} \in (e_{T_o}^A) \};
- (dataMinCardinality)_{T_o}^E = \{ d \mid \#\{v \mid \langle d, v \rangle \in (e_{T_o}^A), v \in (e_{T_o}^T) \} \geq k \};
- (dataMaxCardinality)_{T_o}^E = \{ d \mid \#\{v \mid \langle d, v \rangle \in (e_{T_o}^A), v \in (e_{T_o}^T) \} \leq k \};
- (dataExactCardinality)_{T_o}^E = \{ d \mid \#\{v \mid \langle d, v \rangle \in (e_{T_o}^A), v \in (e_{T_o}^T) \} = k \};
6.1 The language Hi (OWL 2)

To define the semantics of logical axioms, we refer to the usual notion of satisfaction of an axiom with respect to an interpretation \( \mathcal{I} \). The rules defining such notion are specified as follows, where \( e, e_1, e_2, e_3 \) are expressions, \( a \) is an atomic expression and \( lt \) is a literal in \( \mathcal{L}^O \):

\[
\begin{align*}
\mathcal{I} \models \text{ClassAssertion}(e_1 \ e_2) & \text{ if } (e_2^I)^I = \text{true}, \ (e_1^I)^E \text{ is defined, and } e_2^I \in (e_1^I)^E; \\
\mathcal{I} \models \text{ObjectPropertyAssertion}(e_1 \ e_2 \ e_3) & \text{ if } (e_2^I)^I = (e_3^I)^I = \text{true}, \ (e_1^I)^R \text{ is defined, and } \langle e_2^I, e_3^I \rangle \in (e_1^I)^R; \\
\mathcal{I} \models \text{DataPropertyAssertion}(e_1 \ e_2 \ lt) & \text{ if } (e_2^I)^I = \text{true}, \ (e_1^I)^A \text{ is defined, and } \langle e_2^I, lt^I \rangle \in (e_1^I)^A; \\
\mathcal{I} \models \text{NegativeObjectPropertyAssertion}(e_1 \ e_2 \ e_3) & \text{ if } (e_2^I)^I = (e_3^I)^I = \text{true}, \ (e_1^I)^R \text{ is defined, and } \langle e_2^I, e_3^I \rangle \notin (e_1^I)^R; \\
\mathcal{I} \models \text{NegativeDataPropertyAssertion}(e_1 \ e_2 \ lt) & \text{ if } (e_2^I)^I = \text{true}, \ (e_1^I)^A \text{ is defined, and } \langle e_2^I, lt^I \rangle \notin (e_1^I)^A; \\
\mathcal{I} \models \text{SameIndividual}(e_1 \ldots \ e_n) & \text{ if } (e_i^I)^I = \text{true} \text{ for } i \in \{1, \ldots, n\}, \text{ and } e_i^I = e_j^I \text{ for } i, j \in \{1, \ldots, n\}; \\
\mathcal{I} \models \text{DifferentIndividuals}(e_1 \ldots \ e_n) & \text{ if } (e_i^I)^I = \text{true} \text{ for } i \in \{1, \ldots, n\}, \text{ and } e_i^I \neq e_j^I \text{ for } i, j \in \{1, \ldots, n\}; \\
\mathcal{I} \models \text{SubClassOf}(e_1 \ e_2) & \text{ if } (e_1^I)^E \text{ and } (e_2^I)^E \text{ are defined and } (e_1^I)^E \subseteq (e_2^I)^E; \\
\mathcal{I} \models \text{SubObjectPropertyOf}(e_1 \ e_2) & \text{ if } (e_1^I)^R \text{ and } (e_2^I)^R \text{ are defined, and } (e_1^I)^R \subseteq (e_2^I)^R; \\
\mathcal{I} \models \text{SubObjectPropertyOf(ObjectPropertyChain}(e_1 \ldots \ e_n) e) & \text{ if } (e_1^I)^R \text{ is defined, } (e_i^I)^R \text{ is defined for } i \in \{1, \ldots, n\}, \text{ and for all } d_0, \ldots, d_n \text{ such that } (d_0, d_1) \in (e_1^I)^R \text{ and } \ldots \text{ and } (d_{n-1}, d_n) \in (e_n^I)^R \text{ it is implied that } (d_0, d_n) \in (e_1^I)^R; \\
\mathcal{I} \models \text{InverseObjectProperties}(e_1 \ e_2) & \text{ if } (e_1^I)^R \text{ and } (e_2^I)^R \text{ are defined, and } (e_1^I)^R = \{ \langle ind_1, ind_2 \rangle \mid \langle ind_2, ind_1 \rangle \in (e_2^I)^R \}; \\
\mathcal{I} \models \text{SubDataPropertyOf}(e_1 \ e_2) & \text{ if } (e_1^I)^A \text{ and } (e_2^I)^A \text{ are defined, and } (e_1^I)^A \subseteq (e_2^I)^A; \\
\end{align*}
\]
\[ I \models \text{EquivalentClasses}(e_1 \ldots e_n) \text{ if } (e_i^T)^E \text{ is defined for } i \in \{1, \ldots, n\}, \text{ and } (e_i^T)^E = (e_j^T)^E \text{ for } i, j \in \{1, \ldots, n\}; \]

\[ I \models \text{EquivalentObjectProperties}(e_1 \ldots e_n) \text{ if } (e_i^T)^R \text{ is defined for } i \in \{1, \ldots, n\}, \text{ and } (e_i^T)^R = (e_j^T)^R \text{ for } i, j \in \{1, \ldots, n\}; \]

\[ I \models \text{EquivalentDataProperties}(e_1 \ldots e_n) \text{ if } (e_i^T)^A \text{ is defined for } i \in \{1, \ldots, n\}, \text{ and } (e_i^T)^A = (e_j^T)^A \text{ for } i, j \in \{1, \ldots, n\}; \]

\[ I \models \text{DisjointClasses}(e_1 \ldots e_n) \text{ if } (e_i^T)^E \text{ is defined for } i \in \{1, \ldots, n\}, \text{ and } (e_i^T)^E \neq (e_j^T)^E \text{ for } i, j \in \{1, \ldots, n\}; \]

\[ I \models \text{DisjointObjectProperties}(e_1 \ldots e_n) \text{ if } (e_i^T)^R \text{ is defined for } i \in \{1, \ldots, n\}, \text{ and } (e_i^T)^R \neq (e_j^T)^R \text{ for } i, j \in \{1, \ldots, n\}; \]

\[ I \models \text{DisjointDataProperties}(e_1 \ldots e_n) \text{ if } (e_i^T)^A \text{ is defined for } i \in \{1, \ldots, n\}, \text{ and } (e_i^T)^A \neq (e_j^T)^A \text{ for } i, j \in \{1, \ldots, n\}; \]

\[ I \models \text{DisjointUnion}(e_1 \ldots e_n) \text{ if } (e_i^T)^E \text{ is defined, } (e_i^T)^E \text{ is defined for } i \in \{1, \ldots, n\}, \text{ and } E^T = (e_1^T)^E \cup \ldots \cup (e_n^T)^E \text{ and } (e_i^T)^E \cap (e_j^T)^E = \emptyset \text{ for } i, j \in \{1, \ldots, n\} \text{ and } i \neq j; \]

\[ I \models \text{ObjectPropertyDomain}(e_1 e_2) \text{ if } (e_1^T)^R \text{ and } (e_2^T)^E \text{ are defined, and for all } d_1, d_2 \text{ such that } \langle d_1, d_2 \rangle \in (e_1^T)^R \text{ it is implied that } d_1 \in (e_2^T)^E; \]

\[ I \models \text{ObjectPropertyRange}(e_1 e_2) \text{ if } (e_1^T)^R \text{ and } (e_2^T)^E \text{ are defined, and for all } d_1, d_2 \text{ such that } \langle d_1, d_2 \rangle \in (e_1^T)^R \text{ it is implied that } d_2 \in (e_2^T)^E; \]

\[ I \models \text{DataPropertyDomain}(e_1 e_2) \text{ if } (e_1^T)^A \text{ and } (e_2^T)^E \text{ are defined, and for all } d, v \text{ such that } \langle d, v \rangle \in (e_1^T)^A \text{ it is implied that } d \in (e_2^T)^E; \]

\[ I \models \text{DataPropertyRange}(e_1 e_2) \text{ if } (e_1^T)^A \text{ and } (e_2^T)^E \text{ are defined, and for all } d, v \text{ such that } \langle d, v \rangle \in (e_2^T)^E \text{ it is implied that } v \in (e_2^T)^T; \]

\[ I \models \text{ReflexiveObjectProperty}(e) \text{ if } (e^T)^R \text{ is defined and for all } d \text{ in } \Delta_i \text{ it is implied that } \langle d, d \rangle \in (e^T)^R; \]

\[ I \models \text{IrreflexiveObjectProperty}(e) \text{ if } (e^T)^R \text{ is defined and for all } d \text{ in } \Delta_i \text{ it is implied that } \langle d, d \rangle \notin (e^T)^R; \]

\[ I \models \text{SymmetricObjectProperty}(e) \text{ if } (e^T)^R \text{ is defined and for all } d_1, d_2 \text{ such that } \langle d_1, d_2 \rangle \in (e^T)^R \text{ it is implied that } \langle d_2, d_1 \rangle \in (e^T)^R; \]

\[ I \models \text{AsymmetricObjectProperty}(e) \text{ if } (e^T)^R \text{ is defined and for all } d_1, d_2 \text{ such that } \langle d_1, d_2 \rangle \in (e^T)^R \text{ it is implied that } \langle d_2, d_1 \rangle \notin (e^T)^R; \]

\[ I \models \text{TransitiveObjectProperty}(e) \text{ if } (e^T)^R \text{ is defined and } d_1, d_2, d_3 \text{ such that } \langle d_1, d_2 \rangle \in (e^T)^R \text{ and } \langle d_2, d_3 \rangle \in (e^T)^R \text{ it is implied that } \langle d_1, d_3 \rangle \in (e^T)^R; \]

\[ I \models \text{FunctionalObjectProperty}(e) \text{ if } (e^T)^R \text{ is defined and for all } d_1, d_2, d_3 \text{ such that } \langle d_1, d_2 \rangle \in (e^T)^R \text{ and } \langle d_1, d_3 \rangle \in (e^T)^R \text{ it is implied that } d_2 = d_3; \]
\[ I \models \text{InverseFunctionalObjectProperty}(e) \text{ if } (e^{I^O})^R \text{ is defined and for all } d_1, d_2, d_3 \text{ such that } (d_2, d_1) \in (e^{I^O})^R \text{ and } (d_3, d_1) \in (e^{I^O})^R \text{ it is implied that } d_2 = d_3; \]

\[ I \models \text{FunctionalDataProperty}(e) \text{ if } (e^{I^O})^A \text{ is defined and for all } d, v_1, v_2 \text{ such that } (d, v_1) \in (e^{I^O})^A \text{ and } (d, v_2) \in (e^{I^O})^A \text{ it is implied that } v_1 = v_2; \]

\[ I \models \text{DatatypeDefinition}(e_1 \; e_2) \text{ if } (e_1^{I^O})^T \text{ and } (e_2^{I^O})^T \text{ are defined, and } (e_1^{I^O})^T = (e_2^{I^O})^T; \]

\[ I \models \text{HasKey}(e \; e_1 \ldots \; e_n) \text{ ( } e_{n+1} \ldots \; e_m \text{ ) } \text{ if } (e^{I^O})^E \text{ is defined, } (e_i^{I^O})^R \text{ is defined for } i \in \{1, \ldots, n\} \text{ and } (e_j^{I^O})^A \text{ is defined for } j \in \{n + 1, \ldots, m\}, \text{ and for all } d_1, d_2, d_1, \ldots, d_m, v_{n+1}, v_m \text{ such that } (i) d_a, d_b, d_1, \ldots, d_m \in \Delta_{\text{named}}, \text{ (ii) } (d_a, d_i) \in (e_i^{I^O})^R \text{ for } i \in \{1, \ldots, n\}, \text{ and (iii) } (d_a, v_j) \in (e_j^{I^O})^A \text{ for } j \in \{n + 1, \ldots, m\}, \text{ it is implied that } d_a = d_b. \]

We say that an interpretation \( I \) satisfies a \( \text{Hi(OWL 2)} \) ontology \( O \), written \( I \models O \), if it satisfies all the logical axioms of \( O \). In this case we call \( I \) a model of \( O \). A \( \text{Hi(OWL 2)} \) ontology is satisfiable if it admits at least one model, otherwise is unsatisfiable. Finally we say that an axiom \( Ax \) is entailed by an ontology \( O \), written \( I \models Ax \), if \( Ax \) is satisfied by model of \( O \), whereas \( Ax \) is violated by \( O \) if \( O \cup \{Ax\} \) is unsatisfiable.

Note that \( \text{Hi(OWL 2)} \) has the following properties:

- it allows using class expressions imposing cardinality restrictions involving object properties (e.g., expressions of the form \( \text{ObjectMin} \text{Cardinality} \));

- it allows stating transitivity of object properties (e.g., axioms of the form \( \text{Transitive} \text{-ObjectProperty} \));

- given a \( \text{Hi(OWL 2)} \) ontology \( O \), and two IRIs \( e_1 \) and \( e_2 \) in \( V_{OP}^O \), the set of axioms in \( O \) might non trivially entail equality between the extensions of \( e_1 \) and \( e_2 \) when such IRIs are interpreted as object properties, i.e., \( O \) might be such that \( O \models \text{SameIndividual}(e_1 \; e_2) \) and, consequently, imposes that every model \( M \) of \( O \) must be such that \( (e_1^{I^M})^R_M = (e_2^{I^M})^R_M \).

Note also that in [61] has been shown that (i) the problem of checking the satisfiability of ontologies expressed in logics showing the three above characteristics is undecidable, and (ii) satisfiability checking become decidable for ontologies imposing the unique role name assumption (i.e., in the case of a \( \text{Hi(OWL 2)} \) ontology \( O \), satisfiability checking for \( O \) is decidable only if \( O \) contains an axiom of the form \( \text{DifferentIndividuals}(e_1 \ldots e_n) \) involving every \( e_i \in V_{OP}^O \). Thus we can state the following two propositions.

**Proposition 29** Checking satisfiability of a \( \text{Hi(OWL 2)} \) ontology is undecidable.

**Proposition 30** Checking satisfiability of a \( \text{Hi(OWL 2)} \) ontology enforcing the unique role name assumption is decidable in nondeterministic exponential time.
6.2 The language \( \mathcal{H} \text{(OWL 2 QL)} \)

In this section we present the language \( \mathcal{H} \text{(OWL 2 QL)} \) as the language obtained by interpreting \( \text{OWL 2 QL} \) ontologies by the MMS semantics.

The syntax of \( \mathcal{H} \text{(OWL 2 QL)} \) is the same as the syntax of \( \text{OWL 2 QL} \) presented in Section 4.4. However, for the sake of simplicity, we make some assumptions regarding both the considered complex expressions and the considered logical axioms. As for the considered complex expressions:

1. we do not consider expressions of the form \( \text{ObjectComplementOf} \), \( \text{ObjectIntersectionOf} \) as it can be easily shown that they do not add expressive power to the language. As a consequence, we only consider \( \text{OWL 2 QL} \) class expressions from Table 6.1 where \( a \), possibly with subscripts, is an atomic expression, and \( e \) is either an atomic expression or an object property expression. As in \( \text{OWL 2 QL} \), an expression of the form \( \text{ObjectSomeValuesFrom}(op \ a) \) is a subclass expression only if \( a = \text{owl:Thing} \);  

2. we do not consider expressions of the form \( \text{DataIntersectionOf} \), as the the fixed interpretation of \( \text{OWL 2 QL} \) datatypes, which is imposed by the \( \text{OWL 2 QL} \) datatype map \( DM_{\text{ql}} \) defined in Section 4.4.2, combined with the fact that the syntax of \( \text{OWL 2 QL} \) does not allow for using constraining facets on datatypes, allows one to use such an expression only for renaming of an \( \text{OWL 2 QL} \) predefined datatypes.

<table>
<thead>
<tr>
<th>( \text{DataSomeValuesFrom}(a_1 \ a_2) )</th>
<th>( \text{ObjectSomeValuesFrom}(op \ a) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 6.1. ( \mathcal{H} \text{(OWL 2 QL)} ) Class expressions (( a ) is an atomic expression, ( op ) is an object property expression)</td>
<td></td>
</tr>
</tbody>
</table>

As for the logical axioms, we only consider logical axioms having one of the forms listed in Table 6.2 where TBox axioms are partitioned into negative TBox axioms and positive TBox axioms, and (i) \( sub \), possibly with subscripts, is a subclass expression, (ii) \( c \) is a general class expression, (iii) \( op \), possibly with subscripts, is an object property expression, (iv) \( a \), possibly with subscripts, is an atomic expression, and (iii) \( lt \), is a literal.

Note that regarding the allowed axioms we made the following assumptions:

1. We omit to deal with \( \text{OWL 2 QL} \) axioms of the form:
   - \( \text{ObjectPropertyDomain} \),  
   - \( \text{ObjectPropertyRange} \),  
   - \( \text{SymmetricObjectProperty} \),  
   - \( \text{AsymmetricObjectProperty} \),  
   - \( \text{DataPropertyDomain} \),  
   - \( \text{EquivalentClasses} \),  
   - \( \text{EquivalentObjectProperties} \),  
   - \( \text{EquivalentDataProperties} \),  
   - \( \text{InverseObjectProperties} \).
as they can be expressed by appropriate combinations of the axioms taken into account;

2. For the same reason, we only consider axioms of the form `DisjointClasses`, `DisjointObjectProperties`, and `DisjointDataProperties` that predicates on two entities;

3. We do not consider axioms of the form `DatatypeDefinition` and `Declaration (Datatype)` since the restrictions on the use of data types in OWL 2 QL are such that they do not add expressive power to the language.

4. We do not consider axioms of the form `DifferentIndividuals`, because their use in conjunctive queries would lead to insurmountable computational obstacles \[42\], already in OWL 2 QL.

<table>
<thead>
<tr>
<th>Positive TBox axioms</th>
<th>Negative TBox axioms</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SubClassOf(sub, c)</code></td>
<td><code>DisjointClasses(sub_1, sub_2)</code></td>
</tr>
<tr>
<td><code>SubObjectPropertyOf(op_1, op_2)</code></td>
<td><code>DisjointObjectProperties(op_1, op_2)</code></td>
</tr>
<tr>
<td><code>SubDataPropertyOf(a_1, a_2)</code></td>
<td><code>DisjointDataProperties(a_1, a_2)</code></td>
</tr>
<tr>
<td><code>ReflexiveObjectProperty(e)</code></td>
<td><code>IrreflexiveObjectProperty(e)</code></td>
</tr>
<tr>
<td><code>DataPropertyRange(a_1, a_2)</code></td>
<td><code>Declaration</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ABox axioms</th>
<th>Declaration axioms</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ClassAssertion(a_1, a_2)</code></td>
<td><code>Declaration(Class(a))</code></td>
</tr>
<tr>
<td><code>ObjectPropertyAssertion(op, a_1, a_2)</code></td>
<td><code>Declaration(ObjectProperty(a))</code></td>
</tr>
<tr>
<td><code>DataPropertyAssertion(a_1, a_2, l)</code></td>
<td><code>Declaration(DataProperty(a))</code></td>
</tr>
</tbody>
</table>

Table 6.2. Hi(OWL 2 QL) axioms (\(a\) is an atomic expression, \(op\) is an object property expression), \(c\) is a class expression, \(sub\) is a subclass expression, \(lt\) is a literal)

An Hi(OWL 2 QL) ontology (simply ontology in the following) is a finite set of Hi(OWL 2 QL) axioms. Given the vocabulary \(V_O = (V_N^C, V_C, V_{OP}^O, V_{DP}^O, V_I^O, V_{DT}^O, V_E^O, L_O)\) of an ontology \(O\), we denote with \(Exp_O\) the set of all the Hi(OWL 2 QL) well-formed expressions that can be built on the basis of \(V_O\), defined as

\[
Exp_O = V_N^C \cup Exp_C \cup Exp_{OP} \cup Exp_{DP} \cup Exp_{DT}
\]

where

- \(Exp_C = V_C \cup \{ObjectSomeValuesFrom(e_1,e_2) \mid e_1 \in Exp_{OP}^O, e_2 \in V_C\} \cup \{DataSomeValuesFrom(e_1,e_2) \mid e_1 \in V_{DP}, e_2 \in V_{DT}\}\) are the class expressions;

- \(Exp_{OP} = V_{OP} \cup \{ObjectInverseOf(e_1) \mid e_1 \in V_{OP}\}\) are the object property expressions;

- \(Exp_{DP} = V_{DP}\) are the data property expressions;

- \(Exp_{DT} = D_{ql}\) are the datatype expressions.

As in OWL 2, interpretation of datatypes and literals is based on the OWL 2 QL datatype map defined in Section 4.4.2. It is worth noting that \(DM_{ql}\) was designed in such a way to satisfy the following important properties.
• All the set-based relationships holding among the various data types in \( \mathcal{D}_{ql} \), where \( \mathcal{D}_{ql} \) is the set containing the datatypes listed in Table 4.10 in Section 4.4, can be represented as a set of axioms, where each axiom is either a subsetting axiom (stating that the set of values of one datatype is contained in the set of values of another datatype), or a disjointness axiom (stating disjointness between the set of values of two datatypes). This means that, intuitively, we can capture such relationships by treating datatypes as classes, by means of \( \text{OWL 2 QL} \) axioms. Note that such observation will play a crucial role in the next sections.

• The set \( \mathcal{D}_{ql} \) of supported datatypes is such that the intersection of the value spaces of any set of these datatypes is either empty or infinite, which is necessary to obtain the desired computational properties.

Note that in the following sections, we implicitly refer to a \( \text{Hi (OWL 2 QL)} \) ontology \( \mathcal{O} \) with vocabulary \( \mathcal{V}_O = (\mathcal{V}_O^N, \mathcal{V}_O^C, \mathcal{V}_O^{OP}, \mathcal{V}_O^{DP}, \mathcal{V}_O^I, \mathcal{V}_O^{DT}, \mathcal{V}_O^{FA}, \mathcal{L}_O) \) where, without loss of generality the following two conditions hold:

- \( \mathcal{V}_O^{DT} = \mathcal{D}_{ql} \), meaning that the set of entities that occurring in datatype position in \( \mathcal{O} \) are a subset of the \( \text{OWL 2 QL} \) predefined datatypes having a literal space according to \( \text{DM}_{ql} \);

- there are not two distinct literals \( lt, lt' \) in \( \mathcal{L}_O \) such that \( lt^{LS} = lt'^{LS} \).

Clearly, we can always reduce any ontology to an equivalent one that satisfies the above conditions.

### 6.3 Query language

We concentrate on unions of conjunctive queries (simply called queries in the following), expressed using \( \text{SPARQL} \) syntax. Let \( \mathcal{O} \) be an ontology and \( \mathcal{V} \) a set of variables. We start by introducing the notion of query atom. A **query atom over** \( \mathcal{O} \) (simply called **atom** in the following) has the same form of an axiom with the difference that its arguments belong to the set of terms over \( \mathcal{O} \) and \( \mathcal{V} \). The set of terms, denoted \( \text{Exp}_O^\mathcal{V} \), is defined similarly to \( \text{Exp}_O \), with the difference that its base set is \( \mathcal{V}_O^N \cup \mathcal{V}_O^C \), rather than simply \( \mathcal{V}_O^N \). Note that, similarly to axioms, atoms can be classified into TBox atoms and ABox atoms. Let \( a \) be an atom, and \( $x \) be a variable occurring in \( a \). We say that \$x occurs in argument position in \( a \) if either \( a \) is a TBox atom, or \( a \) is an ABox atom and \$x occurs in individual or value position in \( a \), whereas we say that \$x occurs in predicate position in \( a \) if \( a \) is an ABox atom and \$x occurs in class position, in object property position, in data property position, or in datatype position in \( a \).

A conjunctive query \( q \) over an ontology is an expression of the form

\[
\text{SELECT } t_1 \ldots t_m \text{ WHERE } \{B\} \tag{6.1}
\]

where \( m \geq 1 \), \( B \), called **body of the conjunctive query** \( q \) and denoted by \( \text{body}(q) \), is a non-empty conjunction of atoms, \( t_1 \ldots t_m \) is called the head of \( q \), and each \( t_i \) is either an
expression \( q \) in \( V^O \), or a variable in \( V \) occurring in \( body(q) \). A union of conjunctive queries \( Q \) over an ontology is an expression of the form

\[
\text{SELECT } t_1 \ldots t_m \text{ WHERE } \{ B_1 \text{ UNION } \ldots \text{ UNION } B_n \} \tag{6.2}
\]

where \( n \geq 1 \), and each \( B_j \) is a non-empty conjunction of atoms and each \( t_i \) is either an entity name in \( V^O \), or a variable occurring in at least one atom of every \( B_j \).

Note that the terms \( t_1, \ldots, t_m \) are called distinguished terms of \( q \) (or, \( Q \)) and we will sometimes write \( q(\vec{t}) \) (resp., \( Q(\vec{t}) \)) to denote a query of the form (6.1) (resp., (6.2)), where \( \vec{t} = (t_1, \ldots, t_m) \).

A boolean conjunctive query \( q \) over an ontology is an expression of the form

\[
\text{ASK } \{ B \} \tag{6.3}
\]

where \( B \) is a non-empty conjunction of atoms. Finally, a union of boolean conjunctive queries \( Q \) over an ontology is an expression of the form

\[
\text{ASK } \{ B_1 \text{ UNION } \ldots \text{ UNION } B_n \} \tag{6.4}
\]

where each \( B_j \) is a non-empty conjunction of atoms.

Some observations on the terminology we will use in the following are in order.

- A TBox conjunctive query, or simply TBox query, is a conjunctive query whose body contains only TBox atoms.
- An instance conjunctive query is a conjunctive query whose body contains only ABox atoms. An instance query is a union of instance conjunctive queries.
- A metavariable of a query is a variable that occurs in class position, object property position, data property position or datatype position in the query.
- A predicate-variable of a query is a variable that occurs in predicate position in some of the atoms of the query.
- An atom is ground (resp. metaground, predicate-ground)) if no variable (resp. metavariable, predicate-variable) occurs in it.
- A query is ground (resp. metaground, predicate-ground) if all its atoms are ground (resp. metaground, predicate-ground).
- If \( \alpha \) is respectively a tuple, or the body of a conjunctive query, or a set of bodies of conjunctive query, and \( \mu \) is a substitution for the variables \( \vec{x} \) with expressions \( \vec{a} = (a_1, \ldots, a_n) \), written \( \mu : \vec{x} \leftarrow \vec{a} \), then we denote by \( \mu(\alpha) \) respectively the tuple, or the query body, or set of bodies obtained from \( \alpha \) by substituting every \( x_i \) in \( \vec{x} \) with \( a_i \).

We provide the semantics of queries by following the basic steps for defining a new SPARQL entailment regime, thus specifying:

\footnote{We consider here a slight extension of SPARQL, that allows using entity names in the list, besides variables. Note, however, that entity names in the head can be simulated in SPARQL by using the VALUES construct.}
1. which is the semantics used to interpret the queried ontology,
2. which queries are legal, and
3. which is the notion of answer to queries (called solution in SPARQL jargon) that we adopt.

As for 1, we simply adopt MMS, described in the previous sections. As for 2, we base our proposal on the class of queries which are legal for the SPARQL DS entailment regime, but extend it by relaxing the typing constraint. Specifically, a conjunctive query \( q \) is legal with respect to our entailment regime if and only if in \( \text{body}(q) \) there is no variable occurring either both in object property position and in data property position, or both in class position and in datatype position. Moreover, an union of conjunctive queries \( Q \) is legal if all the conjunctive queries in \( Q \) are legal. In all the following chapters we assume to deal with legal queries. As for 3 we rely on the usual definition of certain answers, where we assign the classical logical meaning to both the existentially quantified variables and union. Note that this implies the following:

- We do not require that existentially quantified variables are bound, in every model, to a known ontology element, as in the SPARQL DS entailment regime;
- The certain answers to a union of conjunctive queries are obtained as the intersection of the answers to the union over all models of the ontology, instead of the union of the certain answers to each conjunctive query in the union, as in the SPARQL DS entailment regime.

Thus, as usual in the context of ontology languages, the semantics of conjunctive queries is based on the notion of certain answer: given an ontology \( O \), and a conjunctive query \( q \) with distinguished terms \( \vec{t} = (t_1, \ldots, t_m) \), a tuple \( \vec{a} = (a_1, \ldots, a_m) \) of entity names in \( V^O_N \) is a certain answer of \( q \) over \( O \) under MMS if and only if there exists a substitution \( \mu \) for the variables in \( \vec{t} \) with entity names in \( \vec{a} \) such that

- \( \mu(\vec{t}) = \vec{a} \), and
- the formula \( \exists \vec{y} \mu(\text{body}(q)) \) is true in every model of \( O \) (written \( O \models \exists \vec{y} \mu(\text{body}(q)) \)), where \( \vec{y} \) is the set of all variables in \( \text{body}(q) \) that do not appear in the distinguished terms \( \vec{t} \).

Observe that \( \exists \vec{y} \mu(\text{body}(q)) \) is a boolean conjunctive query, and in order to specify when \( O \models \exists \vec{y} \mu(\text{body}(q)) \), we resort to the notion of query homomorphism, which is an extension of the well-known notion of homomorphism studied in [26].

**Definition 31** Let \( O \) be an ontology, let \( q \) be a boolean conjunctive query over \( O \), and let \( M = (\Sigma, \mathcal{I}_o) \) a model of \( O \), with \( \Sigma = (\Delta_o, \Delta_v, J^M_o, E^M_o, R^M_o, A^M_o, T^M_o) \). A query homomorphism from \( q \) to \( M \) is a function \( f \) from the terms of \( q \) to the domain \( \Delta_o \cup \Delta_v \) of \( \Sigma \) such that for every \( e \in \text{Exp}^O \cup \text{LO} \) occurring in \( q \), \( f(e) = e^M \), and all the semantic properties that the atoms of \( q \) impose on the various terms are preserved under \( f \), namely (here when we write \( \alpha \in \beta, \alpha \notin \beta, \alpha \subseteq \beta, \) or \( \alpha \cup \beta \), we implicitly mean that \( \alpha \) and \( \beta \) are defined; for example, when we write \( (f(e_1))^R_M \subseteq (f(e_2))^R_M \), we assume that \( R_M \) is defined both for \( f(e_1) \) and for \( f(e_2) \):
• if \( e \) occurs in individual position in body\((q)\), then \((f(e))^{f}\) = true;

• if ClassAssertion\((e_{1} e_{2})\) \(\in\) body\((q)\), then \(f(e_{2}) \in (f(e_{1}))^{EM}\);

• if ObjectPropertyAssertion\((e_{1} e_{2} e_{3})\) \(\in\) body\((q)\), then \((f(e_{2}), f(e_{3})) \in (f(e_{1}))^{RM}\);

• if DataPropertyAssertion\((e_{1} e_{2} e_{3})\) \(\in\) body\((q)\), then \((f(e_{2}), f(e_{3})) \in (f(e_{1}))^{AM}\);

• if SubClassOf\((e_{1} e_{2})\) \(\in\) body\((q)\), then \((f(e_{1}))^{EM} \subseteq (f(e_{2}))^{EM}\);

• if DataPropertyRange\((e_{1} e_{2})\) \(\in\) body\((q)\), then \(\{e' | (e, e') \in (f(e_{1}))^{AM}\} \subseteq (f(e_{2}))^{TM}\);

• if SubObjectPropertyOf\((e_{1} e_{2})\) \(\in\) body\((q)\), then \((f(e_{1}))^{RM} \subseteq (f(e_{2}))^{RM}\);

• if SubDataPropertyOf\((e_{1} e_{2})\) \(\in\) body\((q)\), then \((f(e_{1}))^{AM} \subseteq (f(e_{2}))^{AM}\);

• if DisjointClasses\((e_{1} e_{2})\) \(\in\) body\((q)\), then \((f(e_{1}))^{EM} \cap (f(e_{2}))^{EM} = \emptyset\);

• if DisjointObjectProperties\((e_{1} e_{2})\) \(\in\) body\((q)\), then \((f(e_{1}))^{RM} \cap (f(e_{2}))^{RM} = \emptyset\);

• if ReflexiveObjectProperty\((e)\) \(\in\) body\((q)\), then for every \( e_{2} \in \Delta_{1}^{M} \), \((e_{2}, e_{2}) \in (f(e_{1}))^{RM}\);

• if IrreflexiveObjectProperty\((e)\) \(\in\) body\((q)\), then for every \( e_{2} \in \Delta_{0}^{M} \), \((e_{2}, e_{2}) \notin (f(e_{1}))^{RM}\).

Furthermore, if \( Q \) is a union of boolean conjunctive queries, then any query homomorphism from a disjunct of \( Q \) to \( M \) is said to be a query homomorphism from \( Q \) to \( M \).

By exploiting the above definition, we reformulate the notion of certain answer as follows.

**Definition 32** Given an ontology \( O \) and a union of conjunctive queries \( Q \) over \( O \) with distinguished terms \( \vec{t} = t_{1}, \ldots, t_{m} \), a tuple \( \vec{a} = (a_{1}, \ldots, a_{m}) \) of entity names in \( V_{O}^{Q} \) is a certain answer to \( Q \) over \( O \) under MMS if for every model \( M = (\Sigma, I_{o}) \) of \( O \) under MMS, there are a substitution \( \mu \) such that \( \mu(\vec{t}) = \vec{a} \), and a query \( q \) that is a disjunct in \( Q \), such that there is a query homomorphism \( f \) from \( \mu(\text{body}(q)) \) to \( M \).

Furthermore, we can define the semantics of boolean queries as follows.

**Definition 33** A union of boolean conjunctive queries \( Q \) is entailed by an \( OWL 2 QL \) ontology \( O \) under MMS, written \( O \models Q \), if for every model \( M = (\Sigma, I_{o}) \) of \( O \) under MMS, there is a query homomorphism from \( Q \) to \( M \).

Based on the previous definitions, it is easy to see to establish the following relationship between MMS and the classical First-Order Logic (FOL) entailment (denoted \( \models_{FOL} \)).

**Proposition 34** If \( O \) is an \( OWL 2 QL \) ontology and \( q \) is a metaground boolean conjunctive instance query, then \( O \models q \) if and only if \( O \models_{FOL} q \).
We observe that, under MMS, the semantics of the class of queries that are legal under the Direct Semantics, actually coincides with their classical, first-order semantics, where two occurrences of the same entity in positions of different types are treated according to how punning is interpreted by the direct semantics, i.e., as if they were different entities. This is an immediate consequence of the definition of legal queries for DS, which prevents a variable from occurring in positions of different type. Thus, for any such query $q$, it never happens that a substitution $\mu$ leads to a formula $\exists \vec{y} \, \mu(body(q))$ that is logically implied under MMS whereas is not under DS. Note, indeed, that this may happen only because of multiple occurrences in $\mu(body(q))$ of one entity in positions of different type, that are interpreted as the same entity under MMS while they are interpreted as different entities under DS. In our work, based on the observation that metaground instance queries are legal for both MMS and the Direct Semantics, we exploit the fact that their semantics under MMS and under the Direct Semantics coincide, as follows. In order to compute the certain answers to metaground instance queries, we will rely on existing query answering systems available for OWL 2 QL, such as those described in [21,73,75]. Furthermore, based on the observation that any TBox axiom is logically implied by the ontology under MMS if and only if it is logically implied under DS, we will rely on an OWL 2 QL reasoner to compute the set of ground TBox atoms logically implied under MMS.
Part III

Reasoning in $\mathcal{H}i$ (OWL 2 QL)
Chapter 7

Satisfiability and answering instance queries

In this chapter we tackle the problems of checking the satisfiability of a \( Hi(\text{OWL 2 QL}) \) ontology and of answering instance queries. In particular, we present two algorithms to check ontology satisfiability and to answer instance queries, respectively. Such algorithms are both of based on the notion of canonical pseudo interpretation, that is, a structure having the property of representing all the models of a \( Hi(\text{OWL 2 QL}) \) ontology. We start by introducing a the canonical pseudo interpretation in Section 7.1 Then we show how such a structure can be exploited to decide the satisfiability of an ontology in Section 7.2 Finally we present our algorithm for answering instance queries in Section 7.3.

7.1 The \textit{OWL 2 QL} canonical pseudo-interpretation under MMS

In this section we present a structure, called canonical pseudo-interpretation of the ontology, which, similarly to the canonical interpretation of a \textit{DL-Lite} ontology, enjoys the properties of representing all models of the ontology. To this aim, we start by defining the chase procedure in our setting, and then we show how to use it to build the \textit{OWL 2 QL} canonical pseudo-interpretation. Then, we show few properties of such structure that will be crucial for proving the correctness of algorithms discussed in the next sections.

7.1.1 Chase procedure for \textit{OWL 2 QL} under MMS

The basic idea of the chase for our logic is similar to that of the chase for \textit{DL-Lite} [22]: we build a (possibly infinite) structure, starting from an initial structure \( \text{Chase}^0(\mathcal{O}) \), and then repeatedly computing \( \text{Chase}^{j+1}(\mathcal{O}) \) from \( \text{Chase}^j(\mathcal{O}) \) by applying suitable rules, for as long as they are applicable, following a deterministic strategy that is fair, i.e., is such that if at some point a rule is applicable then it will be eventually applied. In doing so, we make use of two infinite ordered alphabets \( S_o \) and \( S_v \) of variables, both disjoint from \( \text{Exp}^O \) and \( \text{L}^O \), for introducing new unknown individuals and new literals, when needed. Also, at each step \( j \geq 0 \), we compute a structure \( \text{Chase}^j(\mathcal{O}) \) over \( V_s^O \cup S_o^j \) and \( L_s^O \cup S_v^j \), where \( S_o^j \subseteq S_o \) and \( S_v^j \subseteq S_v \).

Note that during the construction of the chase we may introduce axioms of the form \textbf{DatatypeAssertion}(\textit{d ll}) that are not in the syntax of \textit{OWL 2 QL}. Intuitively, we need
such axioms to check whether the existence of models of \( \mathcal{O} \) is prevented by the presence in \( \mathcal{O} \) of an axiom \( \text{DataPropertyAssertion}(p \ e \ lt) \) such that the value \((lt)^{LS}\) that \(lt\) is mapped to by the OWL 2 QL datatype map, is not contained in the extension of the datatype \(d\) that is the range of the data property \(p\). The notion of satisfaction of axioms with respect to an interpretation \(\mathcal{I}\) for an ontology \(\mathcal{O}\), is then extended by including the following condition

\[
\mathcal{I} \models \text{DatatypeAssertion}(d \ lt), \text{ if } lt^{LS} \in (d^{LS})^{T}
\]

We are now ready to introduce the procedure by which \(\text{Chase}(\mathcal{O})\) is built. Initially, \(\text{Chase}^0(\mathcal{O})\) is set equal to \(\mathcal{O}\), i.e., \(\text{Chase}^0(\mathcal{O}) = \mathcal{O}\). Then we extend \(\text{Chase}^0(\mathcal{O})\) by executing the following three main steps:

1. For every class expression in \(\mathcal{O}\), every object property expression in \(\mathcal{O}\), and every data property expression in \(\mathcal{O}\), we add a set of axioms expressing the intensional knowledge that is logically implied by their being respectively a class expression, an object property expression, and a data property expression. For example, if \(e \in \mathcal{O}\), then we add the axioms \(\text{SubClassOf}(e \ \text{owl:Thing})\), and \(\text{DisjointClasses}(e \ \text{owl:Nothing})\) to make it explicit that \(e\) is a subclass of \(\text{owl:Thing}\), and is disjoint with \(\text{owl:Nothing}\). Formally, we extend \(\text{Chase}^0\) with the following sets of axioms:

\[
\begin{align*}
\{\text{SubClassOf}(e \ \text{owl:Thing}), \text{DisjointClasses}(e \ \text{owl:Nothing}) | e \in \mathcal{O}\} \\
\{\text{SubObjectPropertyOf}(e \ \text{owl:topObjectProperty}) | e \in \mathcal{O}\} \\
\{\text{DisjointObjectProperties}(e \ \text{owl:bottomObjectProperty}) | e \in \mathcal{O}\} \\
\{\text{SubClassOf}(\text{ObjectSomeValuesFrom}(e \ \text{owl:Thing}) \ \text{owl:Thing}) | e \in \mathcal{O}\} \\
\{\text{DisjointClasses}(\text{ObjectSomeValuesFrom}(e \ \text{owl:Thing}) \ \text{owl:Nothing}) | e \in \mathcal{O}\} \\
\{\text{SubClassOf}(\text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(e) \ \text{owl:Thing}) \ \text{owl:Thing}) | e \in \mathcal{O}\} \\
\{\text{DisjointClasses}(\text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(e) \ \text{owl:Thing}) \ \text{owl:Nothing}) | e \in \mathcal{O}\} \\
\{\text{SubDataPropertyOf}(e \ \text{owl:topDataProperty}) | e \in \mathcal{O}\} \\
\{\text{SubClassOf}(\text{DataSomeValuesFrom}(e \ \text{owl:Thing}) \ \text{owl:Thing}) | e \in \mathcal{O}\} \\
\{\text{DisjointClasses}(\text{DataSomeValuesFrom}(e \ \text{owl:Thing}) \ \text{owl:Nothing}) | e \in \mathcal{O}\}
\end{align*}
\]

2. For every \(\text{DataPropertyAssertion}\) axiom in \(\mathcal{O}\) stating that a pair \((e, lt)\) belongs to a data property, and for every datatype \(d \in D'_q\), we add an axiom \(\text{DatatypeAssertion}(d \ lt)\), if \(d\) represents a set of data values that contains the one denoted by \(lt\) according to \(D_M\), and an axiom \(\text{DatatypeAssertion}(d \ lt)\), otherwise. Formally we add to \(\text{Chase}^0(\mathcal{O})\) the following sets of axioms:

\[
\begin{align*}
\{\text{DatatypeAssertion}(d \ lt) | \text{DataPropertyAssertion}(p \ e \ lt) \in \mathcal{O}, lt^{LS} \in d^{DT}\} \\
\{\text{DatatypeAssertion}(d \ lt) | \text{DataPropertyAssertion}(p \ e \ lt) \in \mathcal{O}, lt^{LS} \notin d^{DT}\}
\end{align*}
\]

Note that axioms added in this step are not in the syntax of OWL 2 QL, but are needed to make explicit in the chase, for each literal occurring in \(\mathcal{O}\), the datatypes the literal
belongs to and the datatypes it does not. The latter is achieved by explicating that the literal belongs to the datatype \(d\), representing the complement of the datatype \(d \in \mathbb{D}_q\). In particular, we need such axioms to be able to correctly check the satisfiability of \(\mathcal{O}\) (cf. later in this section).

3. For every couple of class expressions \(e_1^c\) and \(e_2^c\) (respectively, object properties \(e_1^{op}\) and \(e_2^{op}\), and data properties \(e_1^{dp}\) and \(e_2^{dp}\)) such that \(\mathcal{O} \models \neg \text{DisjointClasses}(e_1^c, e_2^c)\) (resp. \(\mathcal{O} \models \neg \text{DisjointObjectProperties}(e_1^{op}, e_2^{op})\), and \(\mathcal{O} \models \neg \text{DisjointDataProperties}(e_1^{dp}, e_2^{dp})\)) axioms stating that a new unknown individual in \(S^o_0\) (resp. a couple of unknown individuals in \(S^o_0\), and a couple formed by an unknown individual in \(S^o_0\) and an unknown value in \(S^v_0\)), specific for the couple class \(e_1^c\) and \(e_2^c\), belongs to both \(e_1^c\) and \(e_2^c\) (respectively, for object properties \(e_1^{op}\) and \(e_2^{op}\), and data properties \(e_1^{dp}\) and \(e_2^{dp}\)). Formally we add to \(\text{Chase}^0(\mathcal{O})\) the following sets of axioms:

\[
\begin{align*}
(a) & \{ \text{ClassAssertion} \left( e_1^c, \alpha_{e_1,e_2} \right), \text{ClassAssertion} \left( e_2^c, \alpha_{e_1,e_2} \right) \mid e_1^c \in V^c_0, e_2^c \in V^c_0, \\
(b) & \{ \text{ClassAssertion} \left( e_1^c, \alpha_{e_1,e_2} \right), \text{ObjectPropertyAssertion} \left( e_1^{op}, \alpha_{e_1,e_2} \beta_{e_1,e_2} \right) \mid e_1^c \in V^c_0, e_2^c \in \text{Exp}_c, e_2^c = \text{ObjectSomeValuesFrom} \left( e_1^{op}, e' \right), \\
(c) & \{ \text{ClassAssertion} \left( e_1^c, \alpha_{e_1,e_2} \right), \text{ObjectPropertyAssertion} \left( e_1^{op}, \beta_{e_1,e_2} \alpha_{e_1,e_2} \right) \mid e_1^c \in V^c_0, e_2^c \in \text{Exp}_c, e_2^c = \text{ObjectSomeValuesFrom} \left( \text{ObjectInverseOf} \left( e_1^{op} \right), e' \right), \\
(d) & \{ \text{ClassAssertion} \left( e_1^c, \alpha_{e_1,e_2} \right), \text{DataPropertyAssertion} \left( e_1^{dp}, \alpha_{e_1,e_2} s_{e_1,e_2} \right) \mid e_1^c \in V^c_0, e_2^c = \text{DataSomeValuesFrom} \left( e_1^{dp}, e' \right), \\
(e) & \{ \text{ObjectPropertyAssertion} \left( e_1^{op}, \alpha_{e_1,e_2} \beta_{e_1,e_2} \right), \text{ObjectPropertyAssertion} \left( e_2^{op}, \alpha_{e_1,e_2} \beta_{e_1,e_2} \right) \mid e_1^c \in \text{Exp}_c, e_2^c \in \text{Exp}_c, e_1^c = \text{ObjectSomeValuesFrom} \left( e_1^{op}, e' \right), e_2^c = \text{ObjectSomeValuesFrom} \left( e_2^{op}, e'' \right), \\
(f) & \{ \text{ObjectPropertyAssertion} \left( e_1^{op}, \alpha_{e_1,e_2} \beta_{e_1,e_2} \right), \text{ObjectPropertyAssertion} \left( e_2^{op}, \alpha_{e_1,e_2} \beta_{e_1,e_2} \right) \mid e_1^c \in \text{Exp}_c, e_2^c \in \text{Exp}_c, e_1^c = \text{ObjectSomeValuesFrom} \left( e_1^{op}, e' \right), e_2^c = \text{ObjectSomeValuesFrom} \left( \text{ObjectInverseOf} \left( e_2^{op} \right), e'' \right), \\
(g) & \{ \text{ObjectPropertyAssertion} \left( e_1^{op}, \alpha_{e_1,e_2} \beta_{e_1,e_2} \right), \text{ObjectPropertyAssertion} \left( e_2^{op}, \alpha_{e_1,e_2} \beta_{e_1,e_2} \right) \mid e_1^c \in \text{Exp}_c, e_2^c \in \text{Exp}_c, e_1^c = \text{ObjectSomeValuesFrom} \left( \text{ObjectInverseOf} \left( e_1^{op} \right), e' \right), e_2^c = \text{ObjectSomeValuesFrom} \left( \text{ObjectInverseOf} \left( e_2^{op} \right), e'' \right), \\
(h) & \{ \text{ObjectPropertyAssertion} \left( e_1^{op}, \alpha_{e_1,e_2} \beta_{e_1,e_2} \right), \text{DataPropertyAssertion} \left( e_1^{dp}, \alpha_{e_1,e_2} s_{e_1,e_2} \right) \mid e_1^c \in \text{Exp}_c, e_2^c \in \text{Exp}_c, e_1^c = \text{ObjectSomeValuesFrom} \left( e_1^{op}, e' \right), e_2^c = \text{DataSomeValuesFrom} \left( e_2^{dp}, e'' \right), \\
(i) & \{ \text{ObjectPropertyAssertion} \left( e_1^{op}, \alpha_{e_1,e_2} \beta_{e_1,e_2} \right), \text{DataPropertyAssertion} \left( e_1^{dp}, \alpha_{e_1,e_2} s_{e_1,e_2} \right) \mid e_1^c \in \text{Exp}_c, e_2^c \in \text{Exp}_c, e_1^c = \text{ObjectSomeValuesFrom} \left( e_1^{op}, e' \right), e_2^c = \text{DataSomeValuesFrom} \left( e_2^{dp}, e'' \right), \\
\end{align*}
\]
we compute \( \alpha_1, e_2 \in S_o, \beta_1, e_2 \in S_o, s_1, e_2 \in S_e \) \)
\[(j) \{\text{ObjectPropertyAssertion}(e_1^0, \beta_1^1, e_2^0, \beta_2^1, e_2^0), \text{ObjectPropertyAssertion}(e_2^0, \beta_1^1, e_2^0, \beta_2^1, e_2^0) \mid e_1^0, e_2^0 \in V_{OP}^D, e_2^0 \in V_{OP}^D, O \models \neg \text{DisjointObjectProperties}(e_1^0, e_2^0), \beta_1^1, e_2^0 \in S_o \}\]
\[(k) \{\text{ObjectPropertyAssertion}(e_1^0, \eta_1^1, e_2^0, \eta_2^1, e_2^0), \text{ObjectPropertyAssertion}(e_2^0, \eta_1^1, e_2^0, \eta_2^1, e_2^0) \mid e_1^0 \in V_{OP}, e_2^0 \in \text{Exp}_D^P, e_2^0 = \text{ObjectInverseOf}(e_2^0), O \models \neg \text{DisjointObjectProperties}(e_1^0, e_2^0), \eta_1^1, e_2^0 \in S_o \}\]
\[(l) \{\text{DataPropertyAssertion}(s_1^1, e_1^0, \gamma_1^1, e_2^0), \text{DataPropertyAssertion}(s_2^1, e_1^0, \gamma_2^1, e_2^0) \mid e_1^0 \in V_{DP}^O, e_2^0 \in V_{DP}^O, O \models \neg \text{DisjointDataProperties}(e_1^0, e_2^0), \gamma_1^1, e_2^0 \in S_o \}\]

4. For every object property \( e_0 \) such that \( O \models \neg \text{IrreflexiveObjectProperty}(e_0) \) and \( O \models \neg \text{DisjointObjectProperties}(e_0, e_0) \) we add an axiom stating that a couple of unknown individuals in \( S_o^0 \) belongs to \( e_0 \). Formally we add to Chase\(^0\)(\( O \)) the following set of axioms:
\[(a) \{\text{ObjectPropertyAssertion}(e_0, \beta_0^1, e_0, \beta_0^0) \mid e_0 \in V_{OP}^D, \beta_0^0 \in S_o, \beta_0^1 \in S_o \}\]

The importance of defining Chase\(^0\)(\( O \)) by performing steps \( 3 \) and \( 4 \) will be clear in Section \( 8.2 \), when we will deal with queries that are more general than instance queries. Here, we just want to highlight the following properties of Chase\(^0\)(\( O \)):

- For every \( e_c \in \text{Exp}_D^O \) such that \( O \models \neg \text{DisjointClasses}(e_c, e_c) \), there exists \( e_{c,c} \in S_o \) such that \( e_{c,c} \) is asserted to be an instance of only \( e_c \) in Chase\(^0\)(\( O \)). Intuitively, such \( e_{c,c} \), during the construction of the chase, will be an instance of only the class expressions subsuming \( e_c \) in \( O \), and therefore will be used in the chase to falsify all the subclass relationships that are not implied by \( O \). A similar property holds for \( e_0 \in \text{Exp}_D^P \) (resp. \( e_d \in \text{Exp}_D^P \)) such that \( O \models \neg \text{DisjointObjectProperties}(e_0, e_0) \) (resp. \( O \models \neg \text{DisjointDataProperties}(e_d, e_d) \));

- For every negative TBox axiom \( Ax \), that is possible to build starting from expressions in \( \text{Exp}_D^O \), such that \( O \models \neg Ax \), there exists a \( W \subseteq \text{Chase}^0(\( O \)) \) such that \( W \cup \{ Ax \} \) is unsatisfiable.

Once Chase\(^0\)(\( O \)) has been defined, at each step \( j > 0 \), if suitable conditions hold, we compute Chase\(^j+1\)(\( O \)) from Chase\(^j\)(\( O \)), by applying a chase rule. The complete list of chase rules is specified in Table \( 7.3 \), where \( c_i, a_i, d_i \) denote, respectively, classes in \( V_{OP}^O \), object properties in \( V_{OP}^D \), data properties in \( V_{DP}^O \), and datatypes in \( V_{DP}^D \). Moreover, \( e_i^L \) (resp., \( e_i^R \)) denote individuals (resp., values) occurring in Chase\(^j\)(\( O \)), and thus belonging either to \( V_{OP}^O \) or to \( S_j^O \) (resp., \( L^O \) or \( S_j^O \)), while individuals (resp., literals) \( s_{i+1} \) denote individuals (resp., values) in \( S_j^O \) (resp. \( S_j^O +1 \)) that do not occur in Chase\(^j\)(\( O \)). Note in particular that, for each rule, the first column specifies the rule number, the second column specifies the conditions under which it is applicable on Chase\(^j\)(\( O \)), and the third
column specifies how $\text{Chase}^{j+1}(O)$ is computed $\text{Chase}^j(O)$ in the case where the rule is applicable on $\text{Chase}^j(O)$.

For example, the rule cr2 specifies that if (i)\(\text{ClassAssertion}(c_1, e_1^j) \in \text{Chase}^j(O)\) and \(\text{SubClassOf}(c_1, \text{ObjectSomeValuesFrom}(p_1, c_2)) \in \text{Chase}^j(O)\), and (ii) \(\forall e_2^j \in \text{Chase}^j(O)\), \(\text{ObjectPropertyAssertion}(p_1, e_1^j, e_2^j) \notin \text{Chase}^j(O)\) or that \(\text{ClassAssertion}(c_2, e_2^j) \notin \text{Chase}^j(O)\), then we compute \(\text{Chase}^{j+1}(O)\) by adding to \(\text{Chase}^j(O)\) the axioms \(\text{ClassAssertion}(c_2, s^{j+1})\) and \(\text{ObjectPropertyAssertion}(p_1, e_1^j, s^{j+1})\), where \(s^{j+1} \in S_o\) does not appear in \(\text{Chase}^j(O)\).

Finally, we set \(\text{Chase}(O) = \bigcup_{j \in \mathbb{N}} \text{Chase}^j(O)\). Note that \(\text{Chase}(O)\) is a (possibly infinite) set of axioms whose syntax is the usual OWL 2 QL syntax, except for the possible presence of variables in individual and literal positions, and of DatatypeAssertion axioms.

<table>
<thead>
<tr>
<th>cr</th>
<th>Rule applicability conditions</th>
<th>Rule application result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(\text{ClassAssertion}(c_1, e_1^j) \in \text{Chase}^j(O), ) &lt;br&gt; (\text{SubClassOf}(c_1, c_2) \in \text{Chase}^j(O)) &lt;br&gt; (\text{ClassAssertion}(c_2, e_1^j) \notin \text{Chase}^j(O))</td>
<td>(\alpha = {\text{ClassAssertion}(c_2, e_1^j)})</td>
</tr>
<tr>
<td>2</td>
<td>(\text{ClassAssertion}(c_1, e_1^j) \in \text{Chase}^j(O), ) &lt;br&gt; (\text{SubClassOf}(c_1, \text{ObjectSomeValuesFrom}(p_1, c_2)) \in \text{Chase}^j(O),) &lt;br&gt; (\forall e_2^j \in \text{Chase}^j(O)) &lt;br&gt; or (\text{ObjectPropertyAssertion}(p_1, e_1^j, e_2^j) \notin \text{Chase}^j(O))</td>
<td>(\alpha = {\text{ClassAssertion}(c_2, s^{j+1}),) &lt;br&gt; (\text{ObjectPropertyAssertion}(p_1, e_1^j, s^{j+1})})</td>
</tr>
<tr>
<td>3</td>
<td>(\text{ClassAssertion}(c_1, e_1^j) \in \text{Chase}^j(O),) &lt;br&gt; (\text{SubClassOf}(c_1, \text{ObjectSomeValuesFrom}(p_1, c_2)) \in \text{Chase}^j(O),) &lt;br&gt; (\forall e_2^j \in \text{Chase}^j(O)) &lt;br&gt; or (\text{ObjectPropertyAssertion}(p_1, e_1^j, e_2^j) \notin \text{Chase}^j(O))</td>
<td>(\alpha = {\text{ClassAssertion}(c_2, s^{j+1}),) &lt;br&gt; (\text{ObjectPropertyAssertion}(p_1, e_1^j, s^{j+1})})</td>
</tr>
<tr>
<td>4</td>
<td>(\text{ClassAssertion}(c_1, e_1^j) \in \text{Chase}^j(O),) &lt;br&gt; (\text{SubClassOf}(c_1, \text{DataSomeValuesFrom}(a_1, d_1)) \in \text{Chase}^j(O),) &lt;br&gt; (\forall e_2^j \in \text{Chase}^j(O)) &lt;br&gt; or (\text{DataPropertyAssertion}(a_1, e_1^j) \notin \text{Chase}^j(O))</td>
<td>(\alpha = {\text{DatatypeAssertion}(d_1, s^{j+1}),) &lt;br&gt; (\text{DataPropertyAssertion}(a_1, e_1^j, s^{j+1})}) &lt;br&gt; (\bigcup{\text{DatatypeAssertion}(d_2, s^{j+1}),) &lt;br&gt; (\forall d_2^j \in \text{DatatypeAssertion},) &lt;br&gt; (d_2^j \cap d_1^j = \emptyset})</td>
</tr>
<tr>
<td>5</td>
<td>(\text{ObjectPropertyAssertion}(p_1, e_1^j) \in \text{Chase}^j(O),) &lt;br&gt; (\text{SubClassOf}(\text{ObjectSomeValuesFrom}(p_2, \text{owl:Thing}),) &lt;br&gt; (c_1) \in \text{Chase}^j(O)) &lt;br&gt; and (\text{ClassAssertion}(c_1, e_1^j) \notin \text{Chase}^j(O))</td>
<td>(\alpha = {\text{ClassAssertion}(c_1, e_1^j)})</td>
</tr>
<tr>
<td>6</td>
<td>(\text{ObjectPropertyAssertion}(p_1, e_1^j) \in \text{Chase}^j(O),) &lt;br&gt; (\text{SubClassOf}(\text{ObjectSomeValuesFrom}(p_2, \text{owl:Thing}),) &lt;br&gt; (c_1) \in \text{Chase}^j(O)) &lt;br&gt; and (\text{ClassAssertion}(c_1, e_1^j) \notin \text{Chase}^j(O))</td>
<td>(\alpha = {\text{ClassAssertion}(c_2, s^{j+1}),) &lt;br&gt; (\text{ObjectPropertyAssertion}(p_2, e_1^j, s^{j+1})})</td>
</tr>
<tr>
<td>7</td>
<td>(\text{ObjectPropertyAssertion}(p_1, e_1^j) \in \text{Chase}^j(O),) &lt;br&gt; (\text{SubClassOf}(\text{ObjectSomeValuesFrom}(p_2, \text{owl:Thing}),) &lt;br&gt; (c_1) \in \text{Chase}^j(O)) &lt;br&gt; and (\text{ClassAssertion}(c_1, e_1^j) \notin \text{Chase}^j(O))</td>
<td>(\alpha = {\text{ClassAssertion}(c_2, s^{j+1}),) &lt;br&gt; (\text{ObjectPropertyAssertion}(p_2, e_1^j, s^{j+1})})</td>
</tr>
<tr>
<td>cr</td>
<td>Rule applicability conditions</td>
<td>Rule application result</td>
</tr>
<tr>
<td>----</td>
<td>------------------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>8</td>
<td>( \text{ObjectPropertyAssertion}(p_1 e_1^1 e_2^1) \in \text{Chase}^j(O) ),</td>
<td>( \alpha \cap { \text{DatatypeAssertion}(d_1 s_{i_1}^{(1)}), )</td>
</tr>
<tr>
<td></td>
<td>( \text{SubClassOf}(\text{ObjectSomeValuesFrom}(p_1 \text{owl:Thing})) )</td>
<td>( \cup { \text{DatatypeAssertion}(a_1 e_2^1 s_{i_1}^{(1)}), )</td>
</tr>
<tr>
<td></td>
<td>( \text{DataSomeValuesFrom}(a_1 d_1^1) \in \text{Chase}^j(O) ),</td>
<td>( \cup { \text{DatatypeAssertion}(d_2 s_{i_1}^{(1)}), )</td>
</tr>
<tr>
<td></td>
<td>( \forall c_1^j \text{ClassAssertion}(c_1 e_2^1) \in \text{Chase}^j(O) )</td>
<td>( \forall d_2 [d_2^\text{DT} \subset d_2^\text{PT} \lor d_2^\text{PT} \cap d_2^\text{DT} = \emptyset] )</td>
</tr>
<tr>
<td>9</td>
<td>( \text{ObjectPropertyAssertion}(p_1 e_1^1 e_2^1) \in \text{Chase}^j(O) ),</td>
<td>( \alpha \cap { \text{ClassAssertion}(c_1 e_2^1) } )</td>
</tr>
</tbody>
</table>
|    | \( \text{SubClassOf}(\text{ObjectSomeValuesFrom}(p_1 \text{owl:Thing})(c_1) \in \text{Chase}^j(O),
|    | \( \forall c_1^j \text{ClassAssertion}(c_1 e_2^1) \in \text{Chase}^j(O) \)                   | \( \alpha \cap \{ \text{ClassAssertion}(c_2 e_2^2) \} \)                              |
| 10 | \( \text{ObjectPropertyAssertion}(p_1 e_1^1 e_2^1) \in \text{Chase}^j(O) \),                  | \( \alpha \cap \{ \text{ClassAssertion}(c_2 e_2^2) \} \)                              |
|    | \( \text{SubClassOf}(\text{ObjectSomeValuesFrom}(p_1 \text{owl:Thing})(c_1) \in \text{Chase}^j(O),
|    | \( \forall c_1^j \text{ClassAssertion}(c_1 e_2^1) \in \text{Chase}^j(O) \)                   | \( \alpha \cap \{ \text{ObjectPropertyAssertion}(p_2 e_2^2 s_{i_1}^{(1)} + 1) \} \)    |
| 11 | \( \text{ObjectPropertyAssertion}(p_1 e_1^1 e_2^1) \in \text{Chase}^j(O) \),                  | \( \alpha \cap \{ \text{ClassAssertion}(c_2 e_2^2) \} \)                              |
|    | \( \text{SubClassOf}(\text{ObjectSomeValuesFrom}(p_1 \text{owl:Thing})(c_1) \in \text{Chase}^j(O),
|    | \( \forall c_1^j \text{ClassAssertion}(c_1 e_2^1) \in \text{Chase}^j(O) \)                   | \( \alpha \cap \{ \text{ObjectPropertyAssertion}(p_2 e_2^2 s_{i_1}^{(1)} + 1) \} \)    |
| 12 | \( \text{ObjectPropertyAssertion}(p_1 e_1^1 e_2^1) \in \text{Chase}^j(O) \),                  | \( \alpha \cap \{ \text{ClassAssertion}(c_2 e_2^2) \} \)                              |
|    | \( \text{SubClassOf}(\text{ObjectSomeValuesFrom}(p_1 \text{owl:Thing})(c_1) \in \text{Chase}^j(O),
|    | \( \forall c_1^j \text{ClassAssertion}(c_1 e_2^1) \in \text{Chase}^j(O) \)                   | \( \alpha \cap \{ \text{ClassAssertion}(c_1 e_2^1) \} \)                              |
| 13 | \( \text{ObjectPropertyAssertion}(a_1 e_1^1 e_2^1) \in \text{Chase}^j(O) \),                  | \( \alpha \cap \{ \text{ClassAssertion}(c_1 e_2^1) \} \)                              |
|    | \( \text{SubClassOf}(\text{DataSomeValuesFrom}(a_1 d_1^1) \in \text{Chase}^j(O) \),           | \( \alpha \cap \{ \text{ClassAssertion}(c_2 e_2^2) \} \)                              |
|    | \( \forall c_1^j \text{ClassAssertion}(c_1 e_2^1) \in \text{Chase}^j(O) \)                   | \( \alpha \cap \{ \text{ObjectPropertyAssertion}(p_2 e_2^2 s_{i_1}^{(1)} + 1) \} \)    |
| 14 | \( \text{ObjectPropertyAssertion}(a_1 e_1^1 e_2^1) \in \text{Chase}^j(O) \),                  | \( \alpha \cap \{ \text{ClassAssertion}(c_2 e_2^2) \} \)                              |
|    | \( \text{SubClassOf}(\text{DataSomeValuesFrom}(a_1 d_1) \in \text{Chase}^j(O) \),             | \( \alpha \cap \{ \text{ClassAssertion}(c_2 e_2^2) \} \)                              |
|    | \( \forall c_1^j \text{ClassAssertion}(c_1 e_2^1) \in \text{Chase}^j(O) \)                   | \( \alpha \cap \{ \text{ObjectPropertyAssertion}(p_2 e_2^2 s_{i_1}^{(1)} + 1) \} \)    |
| 15 | \( \text{ObjectPropertyAssertion}(a_1 e_1^1 e_2^1) \in \text{Chase}^j(O) \),                  | \( \alpha \cap \{ \text{ClassAssertion}(c_2 e_2^2) \} \)                              |
|    | \( \text{SubClassOf}(\text{DataSomeValuesFrom}(a_1 d_1) \in \text{Chase}^j(O) \),             | \( \alpha \cap \{ \text{ClassAssertion}(c_2 e_2^2) \} \)                              |
|    | \( \forall c_1^j \text{ClassAssertion}(c_1 e_2^1) \in \text{Chase}^j(O) \)                   | \( \alpha \cap \{ \text{ObjectPropertyAssertion}(p_2 e_2^2 s_{i_1}^{(1)} + 1) \} \)    |
7.1 The OWL 2 QL canonical pseudo-interpretation under MMS

Now we show how, based on \( \text{Chase}(\mathcal{O}) \), we build the canonical pseudo-interpretation for the ontology \( \mathcal{O} \). The canonical pseudo-interpretation \( \text{Can}(\mathcal{O}) = (\Sigma_{\text{Can}(\mathcal{O})}, I_{\text{Can}(\mathcal{O})}) \) for \( \mathcal{O} \) can be built as follows:

\[ \Sigma_{\text{Can}(\mathcal{O})} = (\Delta_{\text{Can}(\mathcal{O})}^\mathcal{O}, \Delta_v^{\text{Can}(\mathcal{O})}, I_{\text{Can}(\mathcal{O})}, E_{\text{Can}(\mathcal{O})}, \mathcal{R}_{\text{Can}(\mathcal{O})}, \mathcal{A}_{\text{Can}(\mathcal{O})}, \mathcal{T}_{\text{Can}(\mathcal{O})}) \]

is the interpretation structure for \( \mathcal{O} \) such that:

- \( \Delta_{\text{Can}(\mathcal{O})}^\mathcal{O} = (\text{Exp}^{\mathcal{O}} \cup S_0 \cup \{ \overline{d} \mid d \in \mathcal{D}_q \}) \);
- \( \Delta_v^{\text{Can}(\mathcal{O})} = (\Delta_v \cup S_v) \);
- for every object \( e \in \Delta_{\text{Can}(\mathcal{O})}^\mathcal{O} \), if \( e \) occurs in individual position in \( \text{Chase}(\mathcal{O}) \), then \( e^{I_{\text{Can}(\mathcal{O})}} = \text{true} \); otherwise, \( e^{I_{\text{Can}(\mathcal{O})}} = \text{false} \);
- for every object \( e \in \Delta_{\text{Can}(\mathcal{O})}^\mathcal{O} \), if \( e \notin \text{Exp}^{\mathcal{O}} \), then \( E_{\text{Can}(\mathcal{O})} \) is undefined for \( e \), otherwise \( E_{\text{Can}(\mathcal{O})} \) is defined and such that:
  - if \( e \in \text{V}^{\mathcal{O}} \), then \( e^{E_{\text{Can}(\mathcal{O})}} = \{ e_1 \mid \text{ClassAssertion}(e_1) \in \text{Chase}(\mathcal{O}) \} \);
  - if \( e = \text{ObjectSomeValuesFrom}(e' e'') \), then \( e^{E_{\text{Can}(\mathcal{O})}} = \{ e_1 \mid \text{ObjectPropertyAssertion}(e_1 e_2) \in \text{Chase}(\mathcal{O}) \} \);
  - if \( e = \text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(e' e'')) \), then \( e^{E_{\text{Can}(\mathcal{O})}} = \{ e_2 \mid \text{ObjectPropertyAssertion}(e' e_1 e_2) \in \text{Chase}(\mathcal{O}) \} \);

<table>
<thead>
<tr>
<th>cr</th>
<th>Rule applicability conditions</th>
<th>Rule application result</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>( \text{DataPropertyAssertion}(a_1 e_1' e_2') \in \text{Chase}(\mathcal{O}) ), ( \text{SubClassOf}(\text{DataSomeValuesFrom}(a_1)) ) ( \text{DataSomeValuesFrom}(a_2) \in \text{Chase}(\mathcal{O}) ), and ( \forall d_2 \text{DatatypeAssertion}(d_2) \in \text{Chase}(\mathcal{O}) ) or ( \text{DataPropertyAssertion}(a_2 e_1' e_2') \in \text{Chase}(\mathcal{O}) )</td>
<td>( \alpha = { \text{DatatypeAssertion}(d_2) }, \text{DatatypeAssertion}(a_2 e_1' e_2') } \cup { \text{DatatypeAssertion}(d_3) }, \forall d_3 \in \mathcal{D}_v } \cup \mathcal{D}_v^T ) or ( \mathcal{D}_v^T \cap \mathcal{D}_v^T = \emptyset )</td>
</tr>
<tr>
<td>17</td>
<td>( \text{ObjectPropertyAssertion}(p_1 e_1' e_2') \in \text{Chase}(\mathcal{O}) ), ( \text{SubObjectPropertyOf}(p_1 p_2) \in \text{Chase}(\mathcal{O}) ) ( \text{ObjectPropertyAssertion}(p_2 e_1' e_2') \in \text{Chase}(\mathcal{O}) )</td>
<td>( \alpha = { \text{ObjectPropertyAssertion}(p_2 e_1' e_2') } )</td>
</tr>
<tr>
<td>18</td>
<td>( \text{ObjectPropertyAssertion}(p_1 e_1' e_2') \in \text{Chase}(\mathcal{O}) ), ( \text{SubObjectPropertyOf}(p_1 \text{ObjectInverseOf}(p_2)) \in \text{Chase}(\mathcal{O}) ) ( \text{ObjectPropertyAssertion}(p_2 e_1' e_2') \in \text{Chase}(\mathcal{O}) )</td>
<td>( \alpha = { \text{ObjectPropertyAssertion}(p_2 e_1' e_2') } )</td>
</tr>
<tr>
<td>19</td>
<td>( \text{DataPropertyAssertion}(a_1 e_1' e_2') \in \text{Chase}(\mathcal{O}) ), ( \text{SubDataPropertyOf}(a_1 a_2) \in \text{Chase}(\mathcal{O}) ) ( \text{DataPropertyAssertion}(a_2 e_1' e_2') \in \text{Chase}(\mathcal{O}) )</td>
<td>( \alpha = { \text{DataPropertyAssertion}(a_2 e_1' e_2') } )</td>
</tr>
<tr>
<td>20</td>
<td>( \text{DataPropertyAssertion}(a_1 e_1' e_2') \in \text{Chase}(\mathcal{O}) ), ( \text{DataPropertyRange}(a_1) \in \text{Chase}(\mathcal{O}) ) ( \text{DatatypeAssertion}(d_1 e_1') \in \text{Chase}(\mathcal{O}) )</td>
<td>( \alpha = { \text{DatatypeAssertion}(d_1 e_1') } )</td>
</tr>
<tr>
<td>21</td>
<td>( \text{ReflexiveObjectProperty}(p_1) \in \text{Chase}(\mathcal{O}) ), and there exists ( e_1' \in \mathcal{V}_v^\mathcal{O} \cup \mathcal{S}_v^\mathcal{O} ) such that ( \text{ObjectPropertyAssertion}(p_1 e_1' e_1') \in \text{Chase}(\mathcal{O}) )</td>
<td>( \alpha = { \text{ObjectPropertyAssertion}(p_1 e_1' e_1') } )</td>
</tr>
</tbody>
</table>

Table 7.1. Chase rules for Hi (OWL 2 QL) ontologies
* if \( e = \text{DataSomeValuesFrom} (e', e'') \), then \( e^{E_{\text{Can}}} = \{ e_1 \mid \text{DataPropertyAssertion} (e' e_1), \text{DatatypeAssertion} (e'' e_1) \in \text{Chase}(O) \} \).

- for every object \( e \in \Delta_o^{\text{Can}(O)} \), if \( e \notin \text{Exp}_o^P \), then \( R_{\text{Can}}(O) \) is undefined for \( e \), otherwise \( e^{R_{\text{Can}}(O)} \) is defined and such that:
  * if \( e \in V_O^P \), then \( e^{R_{\text{Can}}(O)} = \{(e_1, e_2) \mid \text{ObjectPropertyAssertion} (e e_1 e_2) \in \text{Chase}(O) \} \);
  * if \( e = \text{ObjectInverseOf} (e') \), then \( e^{R_{\text{Can}}(O)} = \{(e_2, e_1) \mid \text{ObjectPropertyAssertion} (e' e_1 e_2) \in \text{Chase}(O) \} \).

- for every object \( e \in \Delta_o^{\text{Can}(O)} \), if \( e \notin V_D^P \), then \( A_{\text{Can}}(O) \) is undefined for \( e \), otherwise \( e^{A_{\text{Can}}(O)} \) is defined and such that:
  * \( e^{A_{\text{Can}}(O)} = \{(e', v) \mid \text{DataPropertyAssertion} (e' v) \in \text{Chase}(O) \} \).

- for every object \( e \in \Delta_o^{\text{Can}(O)} \), if \( e \notin \{D_{\text{ql}} \cup \{d \mid d \in D_{\text{ql}} \} \} \), then \( T_{\text{Can}}(O) \) is undefined for \( e \), otherwise \( e^{T_{\text{Can}}(O)} \) is defined and such that:
  * \( e^{T_{\text{Can}}(O)} = \{lt^{LS} \mid \text{DatatypeAssertion} (lt) \in \text{Chase}(O), lt \in \text{I}_O \} \cup \{s \mid \text{DatatypeAssertion} (s) \in \text{Chase}(O), s \in S_o \} \).

- \( I_{\text{Can}}(O) \) is a function such that:
  * for every \( e \in \Delta_o^{\text{Can}(O)} \), \( e^{I_{\text{Can}}(O)} = e \), and for every \( s \in S_o \), \( s^{I_{\text{Can}}(O)} = s \);
  * for every \( lt \in \text{I}_O \), \( lt^{I_{\text{Can}}(O)} = lt^{LS} \), and for every \( s \in S_o \), \( s^{I_{\text{Can}}(O)} = s \).

Note that \( \text{Can}(O) \) is called pseudo-interpretation for \( O \) because it does not conform to the definition of interpretation for \( O \). In particular, \( d^{I_{\text{Can}}(O)} \) is defined also for \( d \notin D_{\text{ql}} \). Moreover, for \( d \) such that \( d \in D_{\text{ql}} \), \( d^{I_{\text{Can}}(O)} \) may not coincide with \( d^{DT} \), and may include variables in \( S_o \). Finally, it could be the case that \( \text{owl:Nothing}^{E_{\text{Can}}(O)}, \text{owl:bottomObjectProperty}^{R_{\text{Can}}(O)}, \text{owl:bottomDataProperty}^{A_{\text{Can}}(O)} \) are nonempty. Note also that, when we construct \( \text{Can}(O) \), we allow the functions \( E_{\text{Can}}(O) \), \( R_{\text{Can}}(O) \), \( A_{\text{Can}}(O) \), and \( T_{\text{Can}}(O) \) to be undefined for some elements in \( \Delta_o^{\text{Can}(O)} \), as it is permitted by MMS interpretations; in particular, we let these functions be undefined for all the newly introduced elements from \( S_o \), to make them representing only individuals in \( \text{Can}(O) \). This fact reflects the intuition that there is no need to interpret elements from \( S_o \), that are introduced during the construction of \( \text{Chase}(O) \) and that do not appear in \( O \), as classes or properties to ensure that \( \text{Can}(O) \) represents the minimal knowledge satisfying \( O \).

We highlight that, on the contrary of languages interpreted by MMS, the possibility of interpreting elements of the domain only as individuals, is not offered by languages that are interpreted by a metamodeling semantics, as for example those used in \[\text{[61][33]}\], where the extension functions, that are used to view element of the domain as classes and properties, are total. As a consequence, if one uses semantics equipped with total functions, when interpreting, for example, as a class a newly introduced unknown individual \( e \), there are only two possibilities: (i) either the set of instances associated to \( e \) seen as a class is empty, in which case \( e \) represents a subset of every class in the ontology, or (ii) the extension of \( e \) contains at least one instance. Both cases prevent the existence of a homomorphism from the canonical model to every model of the ontology, which, as we will see, is a fundamental property for exploiting the pseudo-interpretation \( \text{Can}(O) \) to answer general queries under MMS.
7.1.3 Properties of the canonical pseudo interpretation

We next show that $\text{Can}(\mathcal{O})$ plays a crucial role in representing the set of all models of $\mathcal{O}$, with respect to instance checking. To this aim, we define the notion of instance-based homomorphism from $\text{Can}(\mathcal{O})$ to an interpretation $\mathcal{H}$ for $\mathcal{O}$.

**Definition 35** Let $\mathcal{O}$ be an ontology and let $\mathcal{H} = \langle \Sigma_H, I_H \rangle$ be an interpretation for $\mathcal{O}$ such that $\Sigma_H = \langle \Delta_H, \Delta_v, I_H, E_H, R_H, A_H, T_H \rangle$ is an interpretation structure.

Let $\Psi$ be a function from $\Delta_v^\text{Can}(\mathcal{O})$ to $\Delta_v^H$ and from $\Delta_v^\text{Can}(\mathcal{O})$ to $\Delta_v$. We say that $\Psi$ is an instance-based homomorphism from $\text{Can}(\mathcal{O})$ to $\mathcal{H}$ if:

- $\Psi(e_{\text{Can}}^M) = e^H$ for every $e \in \text{Exp}^\mathcal{O}$,
- $\Psi(v_{\text{Can}}^M) = v^H$ for every $v \in L^\mathcal{O}$, and
- $\Psi$ preserves in $\Sigma_H$ the following instance-based properties of $\text{Can}(\mathcal{O})$:

  1. for every $e \in \Delta_v^\text{Can}(\mathcal{O})$, if $(e)_{\text{Can}}^M = \text{true}$ then $(\Psi(e))^H = \text{true}$;
  2. for every $e_1, e_2 \in \Delta_v^\text{Can}(\mathcal{O})$ such that $e_2 \in (e_1)_{\text{Can}}^M$, $\Psi(e_2) \in (\Psi(e_1))^E_H$;
  3. for every $e_1, e_2, e_3 \in \Delta_v^\text{Can}(\mathcal{O})$ such that $(e_2, e_3) \in (e_1)_{\text{Can}}^M$, $(\Psi(e_2), \Psi(e_3)) \in (\Psi(e_1))^A_H$;
  4. for every $e_1, e_2 \in \Delta_v^\text{Can}(\mathcal{O})$ and every $v \in \Delta_v^\text{Can}(\mathcal{O})$ such that $(e_2, v) \in (e_1)_{\text{Can}}^M$, $(\Psi(e_2), \Psi(v)) \in (\Psi(e_1))^A_H$;
  5. for every $d \in D_{\text{ql}}$ and every $v \in \Delta_v^\text{Can}(\mathcal{O})$ such that $v \in (d)_{\text{Can}}^M$, $\Psi(v) \in (\Psi(d))^T_H$;
  6. for every $d$ such that $d \in D'_{\text{ql}}$ and every $v \in \Delta_v^\text{Can}(\mathcal{O})$ such that $v \in (d)_{\text{Can}}^M$, $\Psi(v) \notin (\Psi(d))^T_H$.

We are now able to show that $\text{Can}(\mathcal{O})$ enjoys the following notable property.

**Proposition 36** For every model $M = \langle \Sigma_M, I_M \rangle$ of $\mathcal{O}$, there exists an instance-based homomorphism from $\text{Can}(\mathcal{O})$ to $M$.

**Proof.** Clearly, if $\mathcal{O}$ is unsatisfiable, the proposition is trivially true. Hence, suppose that $\mathcal{O}$ is satisfiable and let $M$ be a model of $\mathcal{O}$.

Let us define $\Psi$ as follows:

- for every $e \in (\text{Exp}^\mathcal{O} \setminus \{ \bar{d} \mid d \in D'_{\text{ql}} \})$, $\Psi(e) = e^{\text{IM}}$;
- for every $v \in \text{rdfs:Literal}^{\text{DT}}$, $\Psi(v) = v$;
- for every $s \in S_\mathcal{O}$ that occurs in Chase$^0(\mathcal{O})$:

  - if $s = \alpha_{e_1, e_2}$, then $\Psi(s) = o$, where $o$ is any object in $(e_1^{\text{IM}})_{E_M} \cap (e_2^{\text{IM}})_{E_M}$.

Note that such an object must exist in $\Delta_v^M$ as, by construction, $\mathcal{O} \models \neg \text{DisjointClasses}(e_i^{\text{IM}})$ for $i, j \in \{1, 2\}$.
if \( s = \beta_{e_1,e_2} \) and \( e_2^2 = \text{ObjectSomeValuesFrom}(e_{op} e') \), then \( \Psi(s) = 0 \), where \( o \) is any object in \((e_{op}^{I_M})^EM\) for which there exists an object \( o' \) in \((e_{op}^{I_M})^EM \cap (e_{op}^{2I_M})^EM\) such that \((o', o)\) is in \((e_{op}^{I_M})^RM\). Note that such objects must exist in \( \Delta_o^M \) as, by construction, \( \mathcal{O} \models \neg \text{DisjointClasses}(e_1^c e_2^c) \) for \( i, j \in \{1, 2\} \). Analogously we can define \( \Psi(s) \) for \( s = \beta_{e_1,e_2}^1 \) (resp. \( s = \beta_{e_1,e_2}^2 \)) and \( e_1^c = \text{ObjectSomeValuesFrom}(e_{op}^1 e') \) (resp. \( e_1^c = \text{ObjectSomeValuesFrom}(e_{op}^2 e') \));

if \( s = \beta_{e_1,e_2}^1 \) (resp. \( s = \beta_{e_1,e_2}^2 \)), then \( \Psi(s) = 0 \), where \( o \) is any object for which there exists an object \( o' \) such that \((o, o')\) is in \((e_{op}^{I_M})^{R_M} \cap (e_{op}^{2I_M})^{R_M}\) (resp. \((o', o)\) is in \((e_{op}^{I_M})^{R_M} \cap (e_{op}^{2I_M})^{R_M}\)). Note that such objects must exist in \( \Delta_o^M \) as, by construction, \( \mathcal{O} \models \neg \text{DisjointObjectProperties}(e_1^i e_{op}^i) \) for \( i, j \in \{1, 2\} \);

if \( s = \eta_{e_1,e_2}^1 \) (resp. \( s = \eta_{e_1,e_2}^2 \)), then \( \Psi(s) = o \), where \( o \) is any object for which there exists an object \( o' \) such that \((o, o')\) is in \((e_{op}^{I_M})^{R_M} \cap ((e_{op}^{2I_M})^{R_M})^{-1}\) (resp. \((o', o)\) is in \((e_{op}^{I_M})^{R_M} \cap ((e_{op}^{2I_M})^{R_M})^{-1}\)). Note that such objects must exist in \( \Delta_o^M \) as, by construction, \( \mathcal{O} \models \neg \text{DisjointObjectProperties}(e_1^i e_{op}^i) \) for \( i, j \in \{1, 2\} \);

if \( s = \nu_{e_1,e_2}^1 \), then \( \Psi(s) = o \), where \( o \) is any object for which there exists a value \( v \) such that \((o, v)\) is in \((e_{op}^{I_M})^{A_M} \cap (e_{op}^{2I_M})^{A_M}\). Note that such objects must exist in \( \Delta_o^M \) as, by construction, \( \mathcal{O} \models \neg \text{DisjointDataProperties}(e_1^i e_{op}^i) \) for \( i, j \in \{1, 2\} \);

if \( s = \nu_{e_1,e_2}^2 \), then \( \Psi(s) = o \), where \( o \) is any object such that \((o, o)\) is in \((e_{op}^{I_M})^{R_M} \cap ((e_{op}^{2I_M})^{R_M})^{-1}\). Note that such objects must exist in \( \Delta_o^M \) as, by construction, \( \mathcal{O} \models \neg \text{DisjointObjectProperties}(e_{op} e_{op}) \) and \( \mathcal{O} \models \neg \text{IrreflexiveObjectProperty}(e_{op}); \)

- for every \( e \in \{d \mid d \in \Delta_o^M\} \), \( \Psi(e) = o' \), where \( o' \) is any object in \( \Delta_o^M \) such that there exists no \( o'' \in \Delta_o^{\text{Can}(O)} \), \( o'' \neq e \), such that \( \Psi(o'') = o' \). Note that such \( o'' \) always exists since \( \Delta_o^M \) is infinite;

- for every \( s \in S_{\text{eq}} \) that occurs in \( \text{Chase}^0(\mathcal{O}) \):

  if \( s = s_{e_1,e_2} \) and \( e_2^2 = \text{DataSomeValuesFrom}(e_{dp} e') \), then \( \Psi(s) = v \), where \( v \) is any value in \((e_{dp}^{I_M})^{I_M} \cap \)
7.1 The OWL 2 QL canonical pseudo-interpretation under MMS

$$(e^M_{ij})_{EM}$$ such that $$(o, v)$$ is in $$(e^M_{dp})_{AM}$$. Note that such object in $${\Delta}_M^O$$ and value in $${\Delta}_M^V$$ must exist as, by construction, $$O \models \lnot \text{DisjointClasses}(e^1_c, e^1_e)$$ for $$i, j \in \{1, 2\}$$.

- If $$s = s^1_{dp}, e^2_{dp}$$, then $$\Psi(s) = v$$, where $$v$$ is any value for which there exists an object $$o$$ such that $$(o, v)$$ is in $$(e^1_{dp})_{IM} \cap (e^2_{dp})_{AM}$$. Note that such object in $${\Delta}_M^O$$ and value in $${\Delta}_M^V$$ must exist as, by construction, $$O \models \lnot \text{DisjointData-Properties}(e^1_{dp}, e^2_{dp})$$ for $$i, j \in \{1, 2\}$$.

Also, let $$\text{Can}^k(O) = \langle \Sigma_k, \mathcal{I}_k \rangle$$ be the portion of $$\text{Can}(O)$$ defined from $$\text{Chase}^k(O)$$, where $$\Sigma_k = \langle \Delta^O_k, \Delta^V_k, l_k, E_k, R_k, A_k, \mathcal{I}_k \rangle$$. By induction, we show that, for every $$k$$, it is possible to define $$\Psi$$ for objects in $${\Delta}^O_k$$ and values in $${\Delta}^V_k$$ in such a way that $$\Psi$$ is an instance-based homomorphism from $$\text{Can}^k(O)$$ to $$M$$.

**Base step.** Let $$k = 0$$, $${\Delta}^O_0 = \text{Exp} \cup \{d \mid d \in \text{V}_d \} \cup S^O_0$$ and $${\Delta}^V_0 = \Delta_s \cup S^V_0$$. Moreover, let $$e_1, e_2$$ be two elements in $${\Delta}^O_0$$ such that $$e_2 \in E^0_0$$. We distinguish between two cases: (a) $$e_2 \in V^O_0$$ and (b) $$e_2 \in S^O_0$$.

(a) in this case $$\Psi(e_2) = e^O_2$$ and, by construction, one of the following is verified:

1. $$e_1 = \text{ObjectSomeValuesFrom}(e'_e, e'')$$ and $$\text{ObjectPropertyAssertion}(e'_e, e_2, e_3)$$, ClassAssertion ($$e''_e, e_3$$) in $$O$$;
2. $$e_1 = \text{ObjectSomeValuesFrom}(e'_e, e'')$$ and $$\text{ObjectPropertyAssertion}(e'_e, e_2, e_3)$$, ClassAssertion ($$e''_e, e_3$$) in $$O$$;
3. $$e_1 = \text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(e'_e), e'')$$, and both $$\text{ObjectPropertyAssertion}(e'_{e3}, e_2, e_3)$$ and ClassAssertion ($$e''_{e3}, e_3$$) in $$O$$;
4. $$e_1 = \text{DataSomeValuesFrom}(e'_e, e'')$$, and both $$\text{DataPropertyAssertion}(e'_e, e_2)$$ and DatatypeAssertion ($$e''_e$$, $$lt$$) are in $$O$$.

In all the above cases, since $$M$$ is a model of $$O$$, we have that $$e^O_{1,2} \in (e^O_{1,2})_{EM}$$, then $$\Psi(e_2) \in (\Psi(e_1))_{EM}$$.

(b) in this case one of the following is verified:

1. $$e_2 = \alpha e_1$$ and $$\Psi(e_2) \in (\Psi(e_1))_{EM}$$;
2. $$e_2 = \beta e_1, e^4$$ and $$e^4 = \text{ObjectSomeValuesFrom}(e_{op} e_1)$$ or $$e^3 = \text{ObjectSomeValuesFrom}(e_{op} e_1)$$, and $$\Psi(e_2) \in (\Psi(e_1))_{EM}$$;
3. $$e_2 = \beta e^4_1, e^4 = \text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(e_{op} e_1))$$ or $$e^3 = \text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(e_{op} e_1))$$, and $$\Psi(e_2) \in (\Psi(e_1))_{EM}$$;
4. $$e_2 = \beta e^4_1, e^4 = \text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(e_{op} e_1))$$ where $$(\Psi(e''_e))_{EM} = (\Psi(e_3))_{EM} \cap (\Psi(e_4))_{EM}$$, and $$\Psi(e_2) \in (\Psi(e_1))_{EM}$$;
5. $$e_2 = \beta e^4_1, e^4 = \text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(e_{op} e_1))$$, and $$e_1 = \text{ObjectSomeValuesFrom}(e_{op} e_1)$$ where $$(\Psi(e''_e))_{EM} = (\Psi(e_3))_{EM} \cap (\Psi(e_4))_{EM}$$, and $$\Psi(e_2) \in (\Psi(e_1))_{EM}$$;
6. $$e_2 = \beta e^4_1$$ (resp. $$e_2 = \beta e^4_1$$), and $$e^3 = \text{ObjectSomeValuesFrom}(e_{op} e_1)$$ (resp. $$e^4 = \text{ObjectSomeValuesFrom}(e_{op} e_1)$$), and $$\Psi(e_2) \in (\Psi(e_1))_{EM}$$;
7. e₂ = β_{e₁,e₄} (resp. e₂ = β_{e₁,e₄}²), and e³ = ObjectSomeValuesFrom/ObjectInverse-Of (e_{op}) e₁ (resp. e³ = ObjectSomeValuesFrom/ObjectInverseOf (e_{op}) e₁), and \( \Psi(e₂) \in (\Psi(e₁))^{EM} \);

8. e₂ = β_{e₁,e₄} (resp. e₂ = β_{e₁,e₄}²), and e³ = ObjectSomeValuesFrom (e_{op}) e' (resp. e³ = ObjectSomeValuesFrom (e_{op}) e'), \( e₁ = \text{ObjectSomeValuesFrom/ObjectInverseOf} (e_{op}) e'' \) where \( (\Psi(e''))^{EM} = (\Psi(e³))^{EM} \cap (\Psi(e₁))^{EM} \), and \( \Psi(e₂) \in (\Psi(e₁))^{EM} \);

9. e₂ = β_{e₁,e₄} (resp. e₂ = β_{e₁,e₄}²), and e³ = ObjectSomeValuesFrom/ObjectInverseOf (e_{op}) e' (resp. e³ = ObjectSomeValuesFrom/ObjectInverseOf (e_{op}) e'), \( e₁ = \text{ObjectSomeValuesFrom} (op₁ \text{owl:Thing}) \) or \( e₁ = \text{ObjectSomeValuesFrom (op₁ \text{owl:Thing}) or} e₁ = \text{ObjectSomeValuesFrom (op₂ \text{owl:Thing})} \), and \( \Psi(e₂) \in (\Psi(e₁))^{EM} \);

10. e₂ = β_{op₁,op₂} (resp. e₂ = β_{op₁,op₂}²), and \( e₁ = \text{ObjectSomeValuesFrom (op₁ \text{owl:Thing}) or} e₁ = \text{ObjectSomeValuesFrom (op₂ \text{owl:Thing})} \), \( e₁ = \text{ObjectSomeValuesFrom (op₁ \text{owl:Thing}) or} e₁ = \text{ObjectSomeValuesFrom (op₂ \text{owl:Thing})} \), and \( \Psi(e₂) \in (\Psi(e₁))^{EM} \);

11. e₂ = \eta_{op₁,op₂} (resp. e₂ = \eta_{op₁,op₂}²), and \( e₁ = \text{ObjectSomeValuesFrom (op₁ \text{owl:Thing}) or} e₁ = \text{ObjectSomeValuesFrom (op₂ \text{owl:Thing})} \), \( e₁ = \text{ObjectSomeValuesFrom (op₁ \text{owl:Thing}) or} e₁ = \text{ObjectSomeValuesFrom (op₂ \text{owl:Thing})} \), and \( \Psi(e₂) \in (\Psi(e₁))^{EM} \);

12. e₂ = γ_{dp₁,dp₂}, and \( e₁ = \text{DataSomeValuesFrom (dp₁ \text{rdfs:Literal}) or} e₁ = \text{ObjectSomeValuesFrom (dp₂ \text{rdfs:Literal})} \), and \( \Psi(e₂) \in (\Psi(e₁))^{EM} \);

Since in all the above cases we have that \( \Psi(e₂) \in (\Psi(e₁))^{EM} \), we proved that \( \Psi \) preserves in \( M \) the instance-based property (2) of Definition 35. Similarly we can prove that \( \Psi \) also preserves in \( M \) instance-based properties (1), (3) (4), (5), and (6).

**Inductive step.** Suppose that \( \Psi \) is an instance-based homomorphism from \( \text{Can}^k(O) \) to \( M \). Depending on the rule that is applied to obtain \( \text{Can}^{k+1}(O) \) starting from \( \text{Can}^k(O) \), we have three possible scenarios:

1. Chase rules with number in (1,5,9,13,17-21): in case one among these rules is applied, we have that \( \Delta^k = \Delta^{k+1} \) and \( \Delta^k = \Delta^{k+1} \), since \( S^k = S^k \) and \( S^{k+1} = S^k \).
   Suppose that \( \text{Can}^{k+1}(O) \) is obtained from \( \text{Can}^k(O) \) by applying rule cr1. Then, ClassAssertion \( (c₁ e) \) is in \( \text{Chase}^k(O) \), SubClassOf \( (c₁ c₂) \) is in \( O \), and ClassAssertion \( (c₂ e) \) is not in \( \text{Chase}^k(O) \). The application of rule cr1 leads to define \( \text{Chase}^{k+1}(O) = \text{Chase}^k(O) \cup \{ \text{ClassAssertion (c₂ e)} \} \).
   Hence, the only difference between \( \text{Can}^k(O) \) and \( \text{Can}^{k+1}(O) \) is that \( e \not\in c₂ E^k \), whereas \( e \in c₂ E^{k+1} \). By inductive hypothesis, we know that \( \Psi(e) \in \Psi(c₁)^{EM} \). Moreover, since \( M \) is a model of \( O \) and \( \text{SubClassOf (c₁ c₂) ∈ O} \), we know that \( M \models \text{SubClassOf (c₁ c₂)} \), then we obtain that \( \Psi(e) \in \Psi(c₂)^{EM} \). This fact proves that \( \Psi \) is an instance-based homomorphism from \( \text{Can}^{k+1}(O) \) to \( M \).

The proof involving rules (5,9,13,17-21) can be given similarly.

2. Chase rules with number in (2,3,6,7,10,11,14,15): in case one among these rules is applied, since \( S^k = S^k \cup \{ s \} \), we have to extend \( \Psi \), i.e., to define \( \Psi(s) \), and to
prove that all the instance-based properties of \( \Psi \) are preserved from \( \text{Can}^{k+1} \) to \( M \), after \( s \) is added to the chase.

Suppose that rule \text{cr2} is applied. By construction, we know that:

- \text{ClassAssertion}(c_1 e) \in \text{Chase}^k(\mathcal{O})
- \text{SubClassOf}(c_1 \text{ObjectSomeValuesFrom}(p_1 c_2)) \in \mathcal{O}
- \text{ObjectPropertyAssertion}(p_1 e s) \in \text{Chase}^{k+1}(\mathcal{O})
- \text{ClassAssertion}(c_2 s) \in \text{Chase}^{k+1}(\mathcal{O})

Since \text{ClassAssertion}(c_1 e) \in \text{Chase}^k(\mathcal{O})\), we know that \( e \in c_1^{E_k} \) and, by inductive hypothesis, \( \Psi(e) \in \Psi(c_1)^{E_M} \). Moreover, since \( M \) is a model of \( \mathcal{O} \), we have that \( M \models \text{SubClassOf}(c_1 \text{ObjectSomeValuesFrom}(p_1 c_2)) \), thus we know that there exists an object \( o \) in \( \Delta^M_o \) such that \( o^{M} = \text{true} \), \( (\Psi(e), o) \in (p_1 I_M)^{R_{M}} \), and \( o \in (c_2^{E_k})^{I_{M}} \).

Let us define \( \Psi(s) = o \). Since by construction \( \Psi(p_1) = p_1^{I_M} \) and \( \Psi(c_2) = c_2^{I_M} \), we can derive that:

Let us define \( \Psi(s) = o \). Then we can derive that:

- \( s^{k+1} = \text{true} \) and \( (\Psi(s))^{I_M} = \text{true} \);
- \( (e, s) \in p_1^{R_{k+1}} \) and \( (\Psi(e)\Psi(s)) \in \Psi(p_1)^{R_{M}} \);
- \( s \in c_2^{E_{k+1}} \) and \( \Psi(s) \in (\Psi(c_2))^{E_M} \)

Therefore we have proved that \( \Psi \) preserves in \( M \) instance-based properties (1), (2), and (3) for \( \text{Can}^{k+1} \). Since after the application of rule \text{cr2} instance-based properties (4), (5), and (6) are trivially preserved in \( M \) for \( \text{Can}^{k+1} \), we proved that \( \Psi \) is an instance-based homomorphism from \( \text{Can}^{k+1} \) to \( M \).

The proof involving rules (3,6,7,10,11,14,15) can be given similarly.

3. Chase rules with number in \((4,8,12,16)\): in this case, we have to consider a scenario similar of the one considered at the previous point, except for the fact that application of rules in \((4,8,12,16)\) causes the invention of a new value variable instead of an object variable, \textit{i.e.}, we have to to take into account that \( S^{k+1}_o = S^k_o \cup \{s\} \). Thus, the proof can be given similarly to the case of rules (2,3,6,7,10,11,14,15).

By a little abuse, we extend the notion of query homomorphism to an analogous function from a query \( Q \) to the pseudo-interpretation \( \text{Can}(\mathcal{O}) \). Thus, it is straightforward to adapt results of [22] to show that \( \text{Can}(\mathcal{O}) \) further enjoys the following property:

- \( \text{Can}(\mathcal{O}) \) is such that the problem of checking the existence of a query homomorphism from a metaground instance query to \( \text{Can}(\mathcal{O}) \) can be reduced to the problem of query answering over an \( \text{OWL 2 QL} \) ontology under the classical first-order semantics.

Let us define a function \( \sigma \) that, given an ontology \( \mathcal{O} \), modifies it by removing from \( \mathcal{O} \) all the negative axioms. Intuitively, \( \sigma \) returns an ontology \( \sigma(\mathcal{O}) \) that can’t be inconsistent, and can be queried to reveal inconsistencies of the original \( \mathcal{O} \).
Proposition 37  Let $\mathcal{O}$ be an ontology and $Q$ be a metaground instance query over $\mathcal{O}$. Then, there exists a query homomorphism from $Q$ to $\text{Can}(\mathcal{O})$ if and only if $\sigma(\mathcal{O}) \models_{\text{FOL}} Q$.

Again from results of [22], the following corollary immediately follows.

Corollary 38  Let $\mathcal{O}$ be a satisfiable ontology and $Q$ be a metaground instance query over $\mathcal{O}$. Then there exists a query homomorphism from $Q$ to $\text{Can}(\mathcal{O})$ if and only if $\mathcal{O} \models_{\text{FOL}} Q$.

Proposition 37 deals with metaground instance queries, as they were defined in Section 6.3. However, as we will see in the next subsection, to provide an algorithm for checking satisfiability, we need to provide an analogous result for a larger class of metaground queries, namely the class of queries comprising, besides OWL 2 QL ABox atoms, also atoms of the form $\text{DatatypeAssertion}$. To this aim, intuitively, we modify the ontology to make it internalize the (positive part of the) OWL 2 QL datatype map as follows:

- we make the new ontology treat both datatypes and their complements as classes, and literals as individuals;
- we assert, via additional axioms, all membership relationships between the newly introduced individuals and classes, and all subset relationships among the newly introduced classes, deriving such set of axioms by the relationships imposed on the corresponding literals and datatypes by the OWL 2 QL datatype map $\text{DM}_{\text{ql}}$.

Thus, let us consider the function $\tau$ that, given an ontology $\mathcal{O}$, modifies it by performing the following steps:

1. it substitutes every datatype $d$ with a class $C_d$, each data property $P_{dp}$ with an object property $P_{dp}$, and each literal $lt$ with an individual $o_v$, where $v = lt^{LS}$ (note that two literals are substituted with the same individual if and only if they represent the same data value);

2. it adds, for every literal $lt$ in $L^\mathcal{O}$ and for every datatype $d$ in $D_{\text{ql}}$, an axiom $\text{ClassAssertion}(C_d o_v)$ if $lt^{LS} \in d^{DT}$, and an axiom $\text{ClassAssertion}(C_d o_v)$ otherwise;

3. it adds for every pair of datatypes $d_1, d_2 \in D_{\text{ql}}$ such that $d_1^{DT} \subseteq d_2^{DT}$, an axiom $\text{SubClassOf}(C_{d,1}, C_{d,2})$.

Similarly, we define the function $\tau_q$ that given a metaground instance query $Q$ over an ontology $\mathcal{O}$, returns a metaground instance query over $\tau(\mathcal{O})$, by modifying the atoms of $Q$ as specified in steps 1 and 2 of the function $\tau$.

We are now able to state the following proposition.

Proposition 39  Let $\mathcal{O}$ be a satisfiable ontology and $Q$ be a metaground instance query over $\mathcal{O}$. Then, there exists a query homomorphism from $Q$ to $\text{Can}(\mathcal{O})$ if and only if $\tau(\sigma(\mathcal{O})) \models_{\text{FOL}} \tau_q(Q)$. 


7.2 Ontology satisfiability

In this section we deal with the problem of checking satisfiability of a Hi(OWL 2 QL) ontology \( \mathcal{O} \). We start by introducing the unsatisfiability query for \( \mathcal{O} \). In a nutshell, such a query checks for the existence of individuals and values contradicting either some negative TBox axiom in \( \mathcal{O} \) or the datatype map \( \mathcal{D}_{\text{mig}} \).

**Definition 40** Let \( \mathcal{O}_{\text{neg}} \) be the set of negative TBox axioms that are contained in \( \mathcal{O} \). The unsatisfiability query for \( \mathcal{O} \) is the metaground instance query having the following form:

\[
Q^\mathcal{O}_{\text{neg}} = \text{DataPropertyAssertion}(\text{owl:bottomDataProperty} \, x \, y) \cup \\
\text{ObjectPropertyAssertion}(\text{owl:bottomObjectProperty} \, x \, y) \cup \\
\text{ClassAssertion}(\text{owl:Nothing} \, x) \cup \bigcup_{a \in O_{\text{neg}}} q_a \cup \bigcup_{d \in \mathcal{D}_{\text{mig}}} q_{\bar{d},d}
\]

where, for every negative axiom \( \alpha \) in \( \mathcal{O}_{\text{neg}} \), the query \( q_a \), called the unsatisfiability query for \( \alpha \), is defined as follows:

- if \( \alpha = \text{DisjointClasses}(e_1, e_2) \), then
  - if \( e_1 \in V^C_{\mathcal{O}} \) and \( e_2 \in V^C_{\mathcal{O}} \), then \( q_a \) is the following query
    \[
    \text{ASK} \{ \text{ClassAssertion}(e_1 \, x) \cup \text{ClassAssertion}(e_2 \, x) \}
    \]
  - if \( e_1 \in V^C_{\mathcal{O}} \) and \( e_2 = \text{ObjectSomeValuesFrom}(e_3 \, \text{owl:Thing}) \), then \( q_a \) is the following query
    \[
    \text{ASK} \{ \text{ClassAssertion}(e_1 \, x) \cup \text{ObjectPropertyAssertion}(e_3 \, x \, y) \}
    \]
  - if \( e_1 \in V^C_{\mathcal{O}} \) and \( e_2 = \text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(e_3 \, \text{owl:Thing})) \), then \( q_a \) is the following query
    \[
    \text{ASK} \{ \text{ClassAssertion}(e_1 \, x) \cup \text{ObjectPropertyAssertion}(e_3 \, y \, x) \}
    \]
- if \( e_1 = \text{ObjectSomeValuesFrom}(e_3 \, \text{owl:Thing}) \) and \( e_2 = \text{ObjectSomeValuesFrom}(e_4 \, \text{owl:Thing}) \), then \( q_a \) is the following query
  - if \( e_1 \in V^C_{\mathcal{O}} \) and \( e_2 = \text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(e_3 \, \text{owl:Thing})) \), then \( q_a \) is the following query
    \[
    \text{ASK} \{ \text{ObjectPropertyAssertion}(e_3 \, x \, y) \cup \text{ObjectPropertyAssertion}(e_4 \, x \, z) \}
    \]
  - if \( e_1 = \text{ObjectSomeValuesFrom}(e_3 \, \text{owl:Thing}) \) and \( e_2 = \text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(e_4 \, \text{owl:Thing})) \), then \( q_a \) is the following query
    \[
    \text{ASK} \{ \text{ObjectPropertyAssertion}(e_3 \, x \, y) \cup \text{ObjectPropertyAssertion}(e_4 \, z \, x) \}
    \]
- if \( e_1 \in V^C_{\mathcal{O}} \) and \( e_2 = \text{DataSomeValuesFrom}(e_3 \, d) \), then \( q_a \) is the following query
  - if \( e_1 \in V^C_{\mathcal{O}} \) and \( e_2 = \text{DataSomeValuesFrom}(e_4 \, d) \), then \( q_a \) is the following query
    \[
    \text{ASK} \{ \text{ClassAssertion}(e_1 \, x) \cup \text{DataPropertyAssertion}(e_3 \, x \, y) \cup \text{DatatypeAssertion}(d \, y) \}
    \]
  - if \( e_1 \in V^C_{\mathcal{O}} \) and \( e_2 = \text{DataSomeValuesFrom}(e_4 \, d) \), then \( q_a \) is the following query
    \[
    \text{ASK} \{ \text{ObjectPropertyAssertion}(e_3 \, x \, y) \cup \text{DataPropertyAssertion}(e_4 \, x \, z) \cup \text{DatatypeAssertion}(d \, z) \}
    \]
Lemma 41 Let $D$ be a set of OWL 2 QL datatypes. If $D$ is intersecting, then:

\[ \text{ASK}\{\text{DatatypeAssertion}(d \, \$x).\text{DatatypeAssertion}(d \, \$x)\} \]

and, for every $d \in D'_{ql}$, $q_{d,d'}$ has the form

\[ \text{ASK}\{\text{DatatypeAssertion}(d \, \$x).\text{DatatypeAssertion}(d \, \$x)\}. \]

Note that the query $Q_{\psi}'$ is not a legal query for MMS because of the presence of $\text{DatatypeAssertion}$ atoms that may involve an entity in datatype position that either belongs to $D'_{ql}$, or has the form $d$ for some $d$ in $D'_{ql}$. Nevertheless, axioms of an analogous form were already introduced in $\text{Chase}(\mathcal{O})$ and semantically captured in $\text{Can}(\mathcal{O})$. Hence, it is straightforward to extend the notion of query homomorphism to $Q_{\psi}'$, and we can state a proposition relating the satisfiability of $\mathcal{O}$ to the existence of a query homomorphism from $Q_{\psi}'$ to $\text{Can}(\mathcal{O})$. To be able to state such a proposition, we need to first show how, given an interpretation $\mathcal{H} = (\Sigma_H, \mathcal{I}_H)$ for $\mathcal{O}$ with $\Sigma_H = (\Delta^H, \Delta_v, I_H', I_H, E_H, R_H, A_H, I_H)$, we can partition the value domain $\Delta_v$ with respect to the set $D_{ql}$ of datatypes that are interpreted by the OWL 2 QL datatype map $\text{DM}_{ql}$. To this aim we introduce the notion of intersecting set of datatypes. If $D \subseteq D_{ql}'$ is a set of datatypes interpreted by the OWL 2 QL datatype map, we say that $D$ is intersecting if $\bigcap_{d \in D} d^{DT} \neq \emptyset$. Note that, by definition, an intersecting set is non-empty. Also, if $D$ is intersecting let the set $S_D \subseteq \Delta_v$ be defined as follows:

\[ S_D = \bigcap_{d \in D} d^{DT} \setminus (\bigcup_{d \in D, d \in D'_{ql} \setminus d^{DT} \subset d^{DT}} d_k^{DT}). \]

and let $S_{\{\text{rdfs:Literal}\}} = \Delta_v \setminus (\bigcup_{d \in D'_{ql} \setminus d^{DT}} d^{DT})$.

Intuitively, given a non-empty set of OWL 2 QL datatypes $D = \{d_1, \ldots, d_m\}$ such that, according to the OWL 2 QL map, all $d_1, \ldots, d_m$ share at least one data value in $\Delta_v$, $S_D$ is the subset of data values in $\Delta_v$ that belong to the intersection of the value spaces of all datatypes in $D$ but do not belong to the value space of any other datatype $d' \in D_{ql}$ such that $d' \notin D$ and $d' \neq \text{rdfs:Literal}$. Thus, $S_{\{\text{rdfs:Literal}\}}$ contains all the data values that are in $\Delta_v$, but cannot be represented by any lexical form according to $\text{DM}_{ql}$.

Based on the definition of $\text{DM}_{ql}$, and on results of [74], we can derive the following lemma.
1. $S_D$ is an infinite set of data values;

2. for every intersecting set of $\text{OWL 2 QL}$ datatypes $D'$, such that $D \neq D'$, $S_D \cap S_{D'} = \emptyset$. 

Now, we can state the proposition showing that $Q'_O$ plays a crucial role in checking satisfiability by stating the following proposition.

**Proposition 42** Let $O$ be an ontology. Then, $O$ is unsatisfiable if and only if there exists a query homomorphism from $Q'_O$ to $\text{Can}(O)$.

**Proof.** $\Leftarrow$: Suppose that there exists a query homomorphism from $Q'_O$ to $\text{Can}(O)$, and suppose, by contradiction that $O$ is satisfiable. Then, there exists at least one disjunct $q$ of $Q'_O$ such that $q$ is true in $\text{Can}(O)$. Suppose that $q$ has the following form:

\[
\text{ASK} \{ \text{DatatypeAssertion}(d \ S x) \cdot \text{DatatypeAssertion}(d \ \bar{S} x) \}
\]

Then there exists a $v$ in $(\Delta_v^{\text{Can}(O)} \cup S_v)$ such that $v \in (d)^{T_{\text{Can}(O)}}$ and $v \in (\bar{d})^{T_{\text{Can}(O)}}$. Let $M = \langle \Sigma_M, I_M \rangle$ be a model of $O$, where $\Sigma_M = \langle \Delta_o^M, \Delta_v^M, I_M, E_M, R_M, \bar{A}_M, T_M \rangle$. By Proposition 36, we know that there exists an instance-based homomorphism $\Psi$ from $\text{Can}(O)$ to $M$. This leads to a contradiction since we would obtain that there exists a $w$ in $\Delta_v^M$ such that $\Psi(v) = w$, $\Psi(v) \in (\Psi(d)^I_M)^{T_M}$, and $\Psi(v) \notin (\Psi(\bar{d})^{I_M})^{T_M}$. Similarly, we would obtain a contradiction by assuming a different form for $q$.

$\Rightarrow$: Suppose that $O$ is unsatisfiable and, by contradiction, that there is no disjunct $q$ of $Q'_O$ such that there exists a query homomorphism from $q$ to $\text{Can}(O)$.

We next show that, in this case, starting from $\text{Can}(O)$, we can define a model $M^c$ of $O$, thus contradicting the hypothesis that $O$ is unsatisfiable. First of all, note that, by hypothesis, there exists no query homomorphism to $\text{Can}(O)$ from the following disjuncts of $Q'_O$:

\[ \text{ASK} \{ \text{ClassAssertion}(\text{owl:Nothing} \ S x) \}; \]
\[ \text{ASK} \{ \text{ObjectPropertyAssertion}(\text{owl:bottomObjectProperty} \ S x \ S y) \}; \]
\[ \text{ASK} \{ \text{DataPropertyAssertion}(\text{owl:bottomDataProperty} \ S x \ S y) \}. \]

Then we know that the sets $(\text{owl:Nothing})^{E_{\text{Can}(O)}}$, $(\text{owl:bottomObjectProperty})^{R_{\text{Can}(O)}}$ and $(\text{owl:bottomDataProperty})^{A_{\text{Can}(O)}}$ are empty. Hence, $\text{Can}(O)$ satisfies all conditions imposed by MMS semantics regarding the interpretation of atomic and complex expressions, except the condition imposing that, if $d$ is in $D_{\text{ql}}$, then $(d^{L_d})^{T_{\text{Can}(O)}} = d^{B_T}$, since, by construction of $\text{Can}(O)$, for every $d \in D_{\text{ql}}$, we have that:

\[
(d^{L_d})^{T_{\text{Can}(O)}} = \{t \in S_{L_S} \mid \text{DatatypeAssertion}(e \ t) \in \text{Chase}(O), t \in L_S \} \\
\cup \{v \mid \text{DatatypeAssertion}(e \ v) \in \text{Chase}(O), v \in S_v \}.
\]

We next show that, if there does not exist any query homomorphism from $Q'_O$ to $\text{Can}(O)$, it is possible to build a model that is identical to $\text{Can}(O)$ except for the definition of the functions $A_{\text{Can}(O)}$ and $T_{\text{Can}(O)}$.

Note that, by construction, for every $s \in S_v$ that is added to the chase, either a single axiom $\text{DatatypeAssertion}(\text{rdfs:Literal} \ s)$ is added or a set of axioms is added,
that includes, for each \( d \in D' \), either an axiom \( \text{DatatypeAssertion}(d \ s) \) or an axiom \( \text{DatatypeAssertion}(\bar{d} \ s) \). Thus, let us consider the set \( D_s = \{ d \in d_{gl}' \mid s \in d_{\text{TCan}(O)} \} \).

Since, by hypothesis, there exists no query homomorphism from \( d_{\text{gl}} \) to \( \text{Can}(O) \), then \( D_s \) is either intersecting or empty. Now, let \( \mu_v : S_v \rightarrow \emptyset \) be an assignment for \( S_v \) defined as follows. For every \( s \in S_v \):

- \( \mu_v(s) \in S_{d_v} \), if \( \{ d \in d_{\text{gl}}' \mid s \in d_{\text{TCan}(O)} \} \) is intersecting, and \( \mu_v(s) \in S_{\{\text{rdfs:Literal}\}} \), otherwise;
- \( \mu_v(s) \notin \{ lt \mid lt \in L^O \} \);
- for every \( s' \) in \( S_v \) such that \( s \neq s' \), \( \mu_v(s') \neq \mu_v(s) \).

It is easy to see that, by Lemma 41, such an assignment always exists. Now, consider the interpretation \( M^c \), that is obtained from \( \text{Can}(O) \) by applying the assignment \( \mu_v \). Formally, let \( M^c = (\Sigma_c, I^c, T^c) \), where \( \Sigma_c = \langle \Delta_v^c, \Delta_o^c, I^c_{\text{Can}(O)}, I^c_{\text{RCan}(O)}, I^c_{\text{Ae}(O)}, I^c_{\text{Tc}} \rangle \), where:

- \( \Delta_v^c = \text{Exp}^O \);
- \( I^c_e = I^c_{\text{Can}(O)} \) for every \( e \in \Delta_o^c \);
- \( \text{lt}^c = \text{lt}^c_{\text{Can}(O)} \) for every \( \text{lt} \in L^O \);
- \( I^c_e, E_e, R_e \) coincide with \( I^c_{\text{Can}(O)}, I^c_{\text{RCan}(O)}, I^c_{\text{Ae}(O)} \), respectively, for every \( o \in \Delta_v^c \);
- for every \( o \in \Delta_v^c \), if \( o^{\text{Ae}(O)} \) is undefined, then \( o^{\text{Ae}(O)} \) is undefined. Otherwise \( o^{\text{Ae}(O)} = \{ (o', v) \mid (o', v) \in o^{\text{Ae}(O)}, v \notin S_v \} \cup \{ (o', \mu_v(s)) \mid (o', s) \in o^{\text{Ae}(O)}, s \in S_v \} \);  
- for every \( o \in \Delta_v^c \), if \( o \notin D_{\text{out}} \), then \( o^{\text{Tc}} \) is undefined. Otherwise \( o^{\text{Tc}} = o^{\text{DT}} \).

Clearly, \( M^c \) is an interpretation for \( O \). We next show that \( M^c \) is in fact a model of \( O \). To this aim, we show that \( M^c \) satisfies every axiom of \( O \). Its is easy to see that, since \( M^c \) coincides with \( \text{Can}(O) \) except for the set \( \Delta_v^c \), strictly included into \( \Delta_o^c \), and for the functions \( \text{Ae}(O) \) and \( \text{Tc}(O) \). Thus \( M^c \) satisfies all positive assertions that do not involve any datatype or literal. Moreover, since \( \text{Ae}(O) \) is obtained from \( \text{Ae}(O) \) by applying the assignment \( \mu_v \) over the value variables introduced during the construction of \( \text{Chase}(O) \), and since \( \mu_v \) simply substitutes each variable in \( S_v \) with a data value not occurring anywhere else within \( \text{Chase}(O) \), it is easy to see that \( M^c \) satisfies every \( \text{DataPropertyAssertion} \) axiom and every \( \text{SubDataPropertyOf} \) axiom of \( O \). On the contrary, the proof showing that every axiom of the form \( \text{DataPropertyRange} \) occurring in \( O \) is satisfied by \( M^c \) is more intricate. Suppose that \( \text{DataPropertyRange} \) \( (P \ d) \in O \), for some \( P \in V^D_{DP} \) and \( d \in d_{\text{gl}}' \). We have to show that every data value within the range of \( (P^{\text{gl}})_{\text{Ae}(O)} \) actually belongs to \( d^{\text{DT}} \), i.e., to the value space of \( d \) according to \( D_{\text{gl}} \). Clearly, this is true if \( d = \text{rdfs:Literal} \). Thus, let us assume that \( d \in d_{\text{gl}}' \) and that \( (o, v) \in (P^{\text{gl}})_{\text{Ae}(O)} \), and that \( v \notin d^{\text{DT}} \). By construction, \( v \) is such that only one of the following is verified:

- \( v = \text{lt}^{\text{LS}} \), where \( \text{lt} \in L^O \). In this case, \( (o, v) \in P^{\text{Ae}(O)} \) and thus the axiom \( \text{DataPropertyAssertion}(P \ o \ \text{lt}) \) belongs to \( \text{Chase}(O) \). This can happen only if \( O \) contains some axiom of the form \( \text{DataPropertyAssertion}(P' \ o \ \text{lt}) \), where \( P' \in V^O_{DP} \); thus, during the construction of \( \text{Chase}(O) \), since \( v \notin d^{\text{DT}} \), \( \text{Chase}^0(O) \) is extended with the axiom \( \text{DatatypeAssertion}(d \ \text{lt}) \). Therefore, \( \text{lt}^{\text{LS}} \in d^{\text{Can}(O)} \).
Thus, given that the construction strategy of \(\text{Chase}(\mathcal{O})\) is fair, at a given moment, rule \(\text{cr20}\) is applied for \(\text{DatatypeAssertion} (d \, \ell t)\), then the axiom \(\text{DatatypeAssertion} (d \, \ell t)\) is added to the chase, thus implying that \(\ell t^{LS} \in d^{\text{Can}(\mathcal{O})}\). Since, by hypothesis, there does not exist any query homomorphism to \(\text{Can}(\mathcal{O})\) from \(q_{d, \bar{d}}\), we obtain a contradiction, and we can derive that \(v \in d^{DT}\):

- \(v = \mu_v(s)\), where \(s \in S_o\): in this case, by construction, \((o, s) \in P^{\text{Can}(\mathcal{O})}\) and thus the axiom \(\text{DataPropertyAssertion}(P \circ o \, s)\) belongs to \(\text{Chase}(\mathcal{O})\). Since \(\mu_v(s) = v\) and \(v \notin d^{DT}\), by the definition of \(\mu_v\) we can derive that \(d \in D_s\). By looking at the chase rules listed in table \([7.1]\) one can easily verify that this can only happen if \(s\) is added to the chase together with an axiom \(\text{DatatypeAssertion} (d \, s)\). Therefore, we know that \(s \in d^{\text{Can}(\mathcal{O})}\). Thus, given that the construction strategy of \(\text{Chase}(\mathcal{O})\) is fair, at a given moment, rule \(\text{cr20}\) is applied for the axiom \(\text{DatatypeAssertion} (d \, s)\), then \(\text{DatatypeAssertion} (d \, s)\) is added to the chase, thus implying that \(s \in d^{\text{Can}(\mathcal{O})}\). Since, by hypothesis, there does not exist any query homomorphism to \(\text{Can}(\mathcal{O})\) from \(q_{d, \bar{d}}\), we obtain a contradiction, and we can derive that \(v \in d^{DT}\).

Then, it remains to show that \(M^c\) satisfies the negative axioms of \(\mathcal{O}\). To this aim, let \(\alpha\) be a negative axiom contained in \(\mathcal{O}\). By hypothesis, there exists no query homomorphism from \(q_o\) to \(\text{Can}(\mathcal{O})\). We show that this fact implies that \(M^c\) satisfies \(\alpha\). Suppose that \(\alpha = \text{DisjointClasses}(e_1 e_2)\), \(e_1\) and \(e_2\) are in \(V^{\mathcal{O}}\), and \(\alpha \in \mathcal{O}\). Then \(q_o\) has the following form

\[
\text{ASK} \{ \text{ClassAssertion}(e_1 s x). \text{ClassAssertion}(e_2 s x) \}
\]

Since no query homomorphism exists from \(q_o\) to \(\text{Can}(\mathcal{O})\), we know that there exists no object \(o\) in \(\Delta^c_o\) such that \(o \in (e_1)^E_o \cap (e_2)^E_o\). Then, by construction, there exists no object \(o\) in \(\Delta^c_o\) such that \(o \in (e_1)^E_o \cap (e_2)^E_o\), which means that \(M^c \models \text{DisjointClasses}(e_1 e_2)\). Similarly, we can prove that \(M^c\) satisfies axioms of the form \(\text{IrreflexiveObjectProperty}, \text{DisjointClasses}\), and \(\text{DisjointObjectProperties}\) that are contained \(\mathcal{O}\). Let us now consider the case of negative axioms of the form \(\text{DisjointDataProperties}\). Suppose that \(\alpha = \text{DisjointDataProperties}(e_1 e_2)\). Since \(\text{Can}(\mathcal{O}) \not\models q_o\), we know that there exist no object \(o \in \Delta^c_o\), and value \(v\) in \(\Delta^c_v\), such that \((o, v) \in e_1^{A_{\text{Can}(\mathcal{O})}} \cap e_2^{A_{\text{Can}(\mathcal{O})}}\). Then, since the assignment \(\mu_v\) is defined so as that (i) \(\mu_v(v) \notin \{\ell t^{LS} \mid \ell t \in L^O\}\), and (ii) for every \(v' \in S_v\) such that \(v' \neq v\), we have \(\mu_v(v') \neq \mu_v(v)\), it follows that does not exist an object \(o\) in \(\Delta^c_o\) and a value \(v\) in \(\Delta^c_v\), such that \((o, v) \in (e_1)^{A_c} \cap (e_2)^{A_c}\), thus proving that \(M^c \models \text{DisjointDataProperties}(e_1 e_2)\).

By combining Propositions \([42, 39, 34]\) we immediately obtain the following theorem.

**Theorem 43** Let \(\mathcal{O}\) be an ontology. Then, \(\mathcal{O}\) is unsatisfiable if and only if \(\tau(\sigma(\mathcal{O})) \models_{\text{FOL}} \tau_q(Q^c_{\mathcal{O}})\).

The above theorem provides the following algorithm that can be used to check the satisfiability of Hi (OWL 2 QL) ontologies:
7. Satisfiability and answering instance queries

Algorithm 1: \textsc{CheckSat}

\begin{algorithmic}[1]
\State \textbf{Input :} $\text{Hi(O\_\text{OWL2 QL})}$ ontology $\mathcal{O}$
\State \textbf{Output :} true if $\mathcal{O}$ is satisfiable, false otherwise
\LineComment{\textbf{1} $Q_\mathcal{O} \leftarrow \text{COMPUTEUNSATISFIABILITYQUERY}(\mathcal{O})$;}
\LineComment{\textbf{2} $\mathcal{O}' \leftarrow \tau(\sigma(\mathcal{O}))$;}
\LineComment{\textbf{3} $Q' \leftarrow \tau_q(Q_\mathcal{O})$;}
\LineComment{\textbf{4} if $\mathcal{O}' \models_{\text{FOL}} Q'$ then}
\LineComment{\hspace{1em} return false;}
\LineComment{\textbf{5} else}
\LineComment{\hspace{1em} return true;}
\LineComment{\textbf{6} end}
\end{algorithmic}

7.3 Answering instance queries

In the previous section we have shown that checking the satisfiability of a Hi(O\_\text{OWL2 QL}) ontology can be reduced to the problem of answering a boolean metaground instance query over the ontology, which in turn can be reduced to the classical problem of answering an instance conjunctive query over an OWL 2 QL ontology under the first-order semantics. In this section, we follow similar line of reasoning to tackle the problem of answering non-metaground instance queries over Hi(O\_\text{OWL2 QL}) ontologies. We start by showing that $\text{Can(}\mathcal{O})$ is representative of all models of $\mathcal{O}$ under HOS with respect to instance queries.

\textbf{Proposition 44} Let $\mathcal{O}$ be a satisfiable ontology and $Q = q_1 \cup \ldots \cup q_n$ be a boolean instance query over $\mathcal{O}$. Then $\mathcal{O} \models Q$ if and only if there exists a query homomorphism from $Q$ to $\text{Can}(\mathcal{O})$.

\textbf{Proof.}

$\Rightarrow$: Suppose that $\mathcal{O} \models Q$. Since, by hypothesis, $\mathcal{O}$ is satisfiable, starting from $\text{Can}(\mathcal{O})$ we can build a model $M^c = (\Sigma_c, \mathcal{I}_c)$ of $\mathcal{O}$, where $\Sigma_c = (\Delta_c^c, \Delta_v^c, \mathcal{I}_c, \mathcal{E}_c, \mathcal{R}_c, \mathcal{A}_c, \mathcal{T}_c)$ as shown in the proof of Proposition [42] Note that $M^c$ is obtained from $\text{Can}(\mathcal{O})$ by exploiting the assignment $\mu_v$ to map variables in $v$ to values in $\Delta_v^c$ so that, for every $s \in S_v$, (i) $\mu_v(s) \notin \{lt^L_s \mid lt \in L^\mathcal{O}\}$, and (ii) if $s' \in S_v$ and $s' \neq s$, then $\mu_v(s) \neq \mu_v(s')$. Hence, the inverse of $\mu_v$ is also a function. Since $M^c$ is a model of $\mathcal{O}$, by definition, there exists a function $f$ and a disjunct $q_i$ of $Q$, for $i \in \{1, \ldots, n\}$, such that $f$ is a query homomorphism from $q_i$ to $M^c$. Thus, the function obtained by composing $f$ and $(\mu_v)^{-1}$ is a query homomorphism from $q_i$ to $\text{Can}(\mathcal{O})$, and, by definition, it is a query homomorphism from $Q$ to $\text{Can}(\mathcal{O})$.

$\Leftarrow$: Suppose that there exists a query homomorphism $f$ from $Q$ to $\text{Can}(\mathcal{O})$. This means that there exists a disjunct $q_i$ of $Q$, for $i \in \{1, \ldots, n\}$, such that $f$ is a query homomorphism from $q_i$ to $\text{Can}(\mathcal{O})$. By Proposition [36] we know that for every model $M$ of $\mathcal{O}$, there exists an instance-based homomorphism $\Psi$ from $\text{Can}(\mathcal{O})$ to $M$. Therefore, being $Q$ an instance query, if we compose the functions $f$ and $\Psi$, we obtain a query homomorphism from $q_i$ to $M$. Then we proved that for every model $M$ of $\mathcal{O}$ there exists a query homomorphism from $Q$ to $M$, and, by definition, that $\mathcal{O} \models Q$. \hfill $\square$

Now we are ready to present our technique for answering instance queries. Such a technique, originally introduced in [57], is called \textit{query answering through metaground-}
ing, and is based on considering that, the canonical pseudo-interpretation $\text{Can}(O) = \langle \Sigma_{\text{Can}(O)}, I_{\text{Can}(O)} \rangle$, where $\Sigma_{\text{Can}(O)} = \langle \Delta_{\text{Can}(O)}, \Sigma_{\text{Can}(O)}, I_{\text{Can}(O)}, E_{\text{Can}(O)}, R_{\text{Can}(O)}, A_{\text{Can}(O)}, T_{\text{Can}(O)} \rangle$, is built, the functions $E_{\text{Can}(O)}, R_{\text{Can}(O)}, A_{\text{Can}(O)}$, and $T_{\text{Can}(O)}$ are left all undefined for variables in $S_{\text{a}}$, so as they can be applied only over known expressions in $\text{Exp}^O$.

The notion of metagrounding of a query $Q$, relies on the notion of instantiation of a conjunctive query that we introduce as follows. Given a conjunctive query $q$, an $n$-tuple $\vec{x} = (x_1, x_2, \ldots, x_n)$ of variables of $q$, an $n$-tuple $\vec{a} = (a_1, a_2, \ldots, a_n)$ of expressions in $\text{Exp}^O$, and a substitution $\mu : \vec{x} \leftarrow \vec{a}$, we denote by $\mu(q)$ the query, called $\mu$-instantiation (or simply instantiation) of $q$, resulting from the following steps:

1. we apply the substitution $\mu$ to $q$, by replacing each $x_i$ in $\vec{x}$ with $a_i$ in $\vec{a}$, for $i \in \{1, \ldots, n\}$, thus obtaining the query $q'$;

2. we replace the atoms in $q'$ that are not legal for OWL 2 QL with equivalent legal atoms involving new variables; for example, an atom of the form $\text{ClassAssertion(Object-SomeValuesFrom}(e_1e_2)e_3)$ in the query is replaced with the conjunction of the atoms $\text{ObjectPropertyAssertion}(e_1e_3w)$ and $\text{ClassAssertion}(e_2w)$, where $w$ is a new variable.

**Definition 45** Let $q$ be a conjunctive query over $O$. A metagrounding of $q$ with respect to $O$ is a $\mu$-instantiation of $q$, for some $\mu : \vec{x} \leftarrow \vec{a}$, such that $\vec{x} = (x_1, x_2, \ldots, x_n)$ are all the metavariables occurring in $q$, and for every $i \in \{1, \ldots, n\}$, if $x_i$ occurs in class (resp. object property, data property, or datatype) position in $q$, then $a_i$ is any expression in $\text{Exp}^O_C$ (resp. $\text{Exp}^O_{OP}, \text{Exp}^O_{DP}$, or $\text{Exp}^O_{DT}$).

Note that, by the above definition, if a variable $x_i$ occurs in more than one type of position in $q$, then substitutions for $x_i$ must be taken by considering the intersection of more than one set of expressions. For example if $x_i$ occurs both in class and object property position in $q$, then substitutions for $x_i$ are taken from the set $\text{Exp}^O_C \cap \text{Exp}^O_{OP}$. Moreover, observe that each atom in a metagrounding of $q$ is either a ground TBox atom, or an ABox atom without metavariables. Thus, in particular, if $q$ is an instance conjunctive query, a metagrounding of $q$ corresponds to a classical first-order conjunctive query.

If $q$ is a conjunctive query, then we denote by $MG(q, O)$ the metaground query that is the union of all metagroundings of $q$ with respect to $O$. If $Q$ is a union of conjunctive queries, then we denote by $MG(Q, O)$ the metaground query that is the union of all metagroundings of every disjunct in $Q$ with respect to $O$.

The following proposition shows that, by computing the metagrounding, it is possible to reduce the problem of checking the existence of a query homomorphism from a instance query to $\text{Can}(O)$ to the problem of checking the existence of a query homomorphism from a metaground instance query to $\text{Can}(O)$, which we can solve as shown in Proposition[37].

**Proposition 46** If $Q$ is an instance query over $O$, then there exists a query homomorphism from $Q$ to $\text{Can}(O)$ if and only if there exists a query homomorphism from $MG(Q, O)$ to $\text{Can}(O)$.

**Proof.** Let $MG(Q, O) = q_1 \cup \ldots \cup q_m$, where every $q_i$, for $i \in \{1, \ldots, m\}$, is a conjunctive metaground instance query.
7. Satisfiability and answering instance queries

⇒: Suppose that there exists a query homomorphism $f$ from $Q$ to $Can(\mathcal{O})$. Then, by definition, $f$ is a query homomorphism from a disjunct $q'$ of $Q$ to $Can(\mathcal{O})$. Let $\{x_1, \ldots, x_n\}$ be the metavariables of $q'$. By definition, if $x_i$, for $i \in \{1, \ldots, n\}$, occurs in class (resp., object property, data property, or datatype) position, then the function $E_{Can(\mathcal{O})}$ (resp., $R_{Can(\mathcal{O})}$, $A_{Can(\mathcal{O})}$, or $I_{Can(\mathcal{O})}$) must be defined for $f(x_i)$. Then, by construction of $Can(\mathcal{O})$, we have that $f(x_i)$ must be in $Exp^O$ (resp., $Exp^O_P$, $Exp^D_P$, $V_D^T$). Let $\mu$ be the substitution $\mu : (x_1, \ldots, x_n) \mapsto (f(x_1), \ldots, f(x_n))$. Clearly $\mu(q')$ is a metagrounding of $q'$ w.r.t. $\mathcal{O}$. Then there exists a disjunct $q_j$ of $MG(Q, \mathcal{O})$, for $j \in \{1, \ldots, m\}$, such that $q_j = \mu(q')$. Therefore, if we define $f'$ as the restriction of $f$ to the variables occurring only in an individual or value position in $q'$, we obtain a query homomorphism from $q_j$ to $Can(\mathcal{O})$, and, by definition, from $MG(Q, \mathcal{O})$ to $Can(\mathcal{O})$.

⇐: Suppose that there exists a query homomorphism $f$ from $MG(Q, \mathcal{O})$ to $Can(\mathcal{O})$. Then, by definition, $f$ is a query homomorphism from a disjunct $q_j$ of $MG(Q, \mathcal{O})$ to $Can(\mathcal{O})$, for some $j \in \{1, \ldots, m\}$. By definition of metagrounding of an union of conjunctive queries, there exists a disjunct $q'$ of $Q$ such that $q_j \in MG(q', \mathcal{O})$. Let $\{x_1, \ldots, x_n\}$ be the metavariables of $q'$, and $\mu$ be the substitution $\mu : (x_1, \ldots, x_n) \mapsto (e_1, \ldots, e_n)$ such that $\mu(q') = q_j$. If we define $f'$ as the extension of $f$ such that $f'(x_i) = e_i$, for $i \in \{1, \ldots, n\}$, we obtain a query homomorphism from $q'$ to $Can(\mathcal{O})$. Then, by definition, $f'$ is a query homomorphism from $Q$ to $Can(\mathcal{O})$.

By combining the above proposition with Proposition 44 and Proposition 38, we obtain the following theorem, which provides an algorithm for answering instance queries over $Hi(\text{OWL 2 QL})$ ontologies.

**Theorem 47** Let $\mathcal{O}$ be a satisfiable ontology and $Q$ a boolean instance query over $\mathcal{O}$. Then, $\mathcal{O} \models Q$ if and only if $\mathcal{O} \models_{\text{FOL}} MG(Q, \mathcal{O})$.

The above theorem provides the following algorithm that can be used to compute the answer to boolean instance queries over satisfiable $Hi(\text{OWL 2 QL})$ ontologies.

**Algorithm 2**: INSTANCEQUERYANS

```
Input : satisfiable $Hi(\text{OWL 2 QL})$ ontology $\mathcal{O}$, boolean instance query $Q$ over $\mathcal{O}$
Output : true if $\mathcal{O} \models Q$, false otherwise

1. $Q' \leftarrow MG(Q, \mathcal{O})$;
2. if $\mathcal{O} \models_{\text{FOL}} Q'$ then
   3. return true;
4. else
5. return false;
6. end
```
Chapter 8

Answering general queries

In this chapter we tackle the problem of answering general queries in \( \text{Hi(OWL 2 QL)} \). We start by showing in Section 8.1 that the problem of answering general queries can be solved by relying to the metagrounding technique introduced in the previous chapter only for a particular class of \( \text{Hi(OWL 2 QL)} \) ontologies, called \textit{TBox-complete ontologies}. Then we present an algorithm to solve the query answering problem over general ontologies in Section 8.2.

8.1 Answering general queries over TBox-complete ontologies

In this section we start to tackle the problem of answering general queries, that are, queries where both ABox and TBox atoms can occur. In the previous chapter we have given an algorithm to answer instance queries that was based on the so-called metagrounding technique, then one might ask whether such a technique can be exploited to answer also queries where also TBox atoms occur. Unfortunately, the answer is negative, as shown by the following example.

Example 48 Let \( O_1 \) be the ontology formed by the following axioms

\[
\begin{align*}
\text{DisjointClasses} ( :A :C) \\
\text{DisjointClasses} ( :A \text{ObjectSomeValuesFrom} (:R \text{owl:Thing})) \\
\text{DisjointClasses} ( :A \text{ObjectSomeValuesFrom} (\text{ObjectInverseOf} (:R \text{owl:Thing}))) \\
\text{ClassAssertion} ( :B :F) \\
\text{ClassAssertion} ( :C :F) \\
\text{ObjectPropertyAssertion} ( :R :F :F) \\
\text{ObjectPropertyAssertion} ( :R :C :A) \\
\text{ObjectPropertyAssertion} ( :R :B :C)
\end{align*}
\]

and let \( q \) be the following query:
ASK \{ClassAssertion(:B $y).
ClassAssertion($z $y).
DisjointClasses(:A $x).
ObjectPropertyAssertion(:R $x $z)\}

Note that $O_1$ is clearly satisfiable, and that the only metavariables occurring in $q$ are $x$ and $z$, as both such variables occur in class position in some atom of $q$. One can easily verify that does not exist a substitution $\mu: (x, z) \leftarrow (e_1, e_2)$ such that $e_1, e_2 \in \text{Exp}_{O_1}$, and $O_1 \models \mu(q)$. Thus, by relying on the metagrounding technique, the query $q$ is evaluated to false. However, suppose to partition the sets of models of $O_1$ into the set $M_1$ of models in which $:A$ and $:B$ are interpreted as non-disjoint classes, and the set $M_2$ of models in which $:A$ and $:B$ are interpreted as disjoint classes. Then, consider the following substitutions:

$\mu_1: (x, z) \leftarrow (:C, :A)$
$\mu_2: (x, z) \leftarrow (:B, :C)$

As entities $:A$, $:C$, and $:B$ are all contained in $\text{Exp}_{O_1}$, both $\mu_1(q)$ and $\mu_2(q)$ are metagroundings of $q$ w.r.t. to $O_1$. Clearly, $\mu_1(q)$ is satisfied by every model of $M_1$, while $\mu_2(Q)$ is not. On the other hand, $\mu_2(Q)$ is satisfied by every model of $M_2$, while $\mu_1(q)$ is not. Still, for every model there exists a substitution of variables that makes $Q$ evaluate to true, which proves that $O_1 \models Q$.

The example shows that, in the presence of TBox atoms in the query, query answering may require to reason by cases, and that such a drawback occurs due to negative axioms, involving expressions in $\text{Exp}_O$, which are neither logically implied by $O$ nor violated by $O$. This consideration leads us to consider a particular class of ontologies, called TBox-complete ontologies, for which we have complete knowledge about negative axioms.

In the following, we denote by $N^O$ the set containing all the negative axioms that is possible to build starting from suitable expressions in $\text{Exp}_O$.

We next introduce the notion of TBox-complete ontology, which in turn is based on the notion of certain negative axiom.

**Definition 49** Let $O$ be an ontology and $\alpha$ be an axiom in $N^O$. We say that $\alpha$ is certain if either $O \models \alpha$, or $\alpha$ is violated by $O$; otherwise, we say that $\alpha$ is uncertain.

**Definition 50** Let $O$ be an ontology. We say that $O$ is TBox-complete if every axiom in $N^O$ is certain.

Note that, by the above definition, we have that every unsatisfiable ontology is trivially TBox-complete.

**Example 51** The ontology $O_1$ presented in Example 48 is not TBox-complete. Indeed, for example, both the axiom DisjointClasses(:A :B) is neither logically implied nor violated by $O_1$. 

\[\square\]
Let \( \alpha \) be a negative axiom in \( \mathcal{N}^O \). The following proposition states that, if \( \mathcal{O} \) is a TBox-complete ontology, \( \alpha \) is satisfied by \( \text{Can}(\mathcal{O}) \) if and only if it is satisfied by \( \text{Can}^0(\mathcal{O}) \), where \( \text{Can}^0(\mathcal{O}) \) is the portion of \( \text{Can}(\mathcal{O}) \) that is defined from \( \text{Chase}^0(\mathcal{O}) \).

**Proposition 52** If \( \mathcal{O} \) is a satisfiable TBox-complete ontology and \( \alpha \) is an axiom in \( \mathcal{N}^O \), then \( \text{Can}^0(\mathcal{O}) \models \alpha \) if and only if \( \text{Can}(\mathcal{O}) \models \alpha \).

**Proof.** Let \( \alpha \) be an axiom in \( \mathcal{N}^O \) and let \( \text{Can}^k(\mathcal{O}) = (\Sigma_k, \mathcal{I}_k) \) be the portion of \( \text{Can}(\mathcal{O}) \) that is defined from \( \text{Chase}^k(\mathcal{O}) \), where \( \Sigma_k = (\Delta_k, \Delta^+_k, \mathcal{I}_k, \mathcal{E}_k, \mathcal{R}_k, \mathcal{A}_k, \mathcal{F}_k) \). We prove the proposition by showing that \( \text{Can}^0(\mathcal{O}) \models \alpha \) if and only if \( \text{Can}^k(\mathcal{O}) \models \alpha \), for every \( k \geq 0 \).

**Base step.** Clearly for \( k = 0 \), both directions of the proposition are trivially verified.

**Inductive step.** Let \( \alpha \) be an axiom in \( \mathcal{N}^O \), and suppose \( \text{Can}^k(\mathcal{O}) \models \alpha \) if and only if \( \text{Can}^0(\mathcal{O}) \models \alpha \).

\[ \iff \quad \text{Let } \alpha \text{ be of the form } \text{DisjointClasses}, \quad \text{and let } e_1, e_2 \in \Delta^+_{k+1} \text{ be such that } e_1^{E_{k+1}} \cap e_2^{E_{k+1}} = \emptyset. \]

Since, by definition, when moving from \( \text{Chase}^j(\mathcal{O}) \) to \( \text{Chase}^{j+1}(\mathcal{O}) \) no axiom is removed from \( \text{Chase}^j(\mathcal{O}) \), we have that \( e_1^{E_k} \cap e_2^{E_k} = \emptyset \) and that \( \text{Can}^0(\mathcal{O}) \models \alpha \).

The proof involving axioms of the form \( \text{DisjointObjectProperties}, \text{DisjointData-Properties}, \text{and IrreflexiveObjectProperty} \) can be given similarly.

\[ \Rightarrow: \text{Let } \alpha \text{ be of the form } \text{DisjointClasses}. \quad \text{Depending on the rule that is applied to obtain } \text{Can}^{k+1}(\mathcal{O}) \text{ starting from } \text{Can}^k(\mathcal{O}) \text{ we have different possible scenarios.} \]

**cr1** In this case we have that \( \text{ClassAssertion } (c_1 e) \) is in \( \text{Chase}^k(\mathcal{O}) \), \( \text{SubClassOf } (c_1 c_2) \) is in \( \mathcal{O} \), \( \text{ClassAssertion } (c_2 e) \) is not in \( \text{Chase}^k(\mathcal{O}) \), and \( \text{Chase}^{k+1}(\mathcal{O}) = \text{Chase}^k(\mathcal{O}) \cup \{ \text{ClassAssertion } (c_2 e) \} \). Then we have that \( c_2^{E_{k+1}} = c_2^{E_k} \cup \{ e \} \), and for every \( e_{cd} \neq e_2 \) we have that \( e_{cd}^{E_{k+1}} = e_{cd}^{E_k} \). Suppose there exists \( e_{cd} \) such that \( e_2^{E_k} \cap e_{cd}^{E_k} = \emptyset \) and \( e_2^{E_{k+1}} \cap e_{cd}^{E_{k+1}} = \emptyset \). Clearly this can happen if and only if \( e \in e_{cd}^{E_k} \). Then we know that \( c_2^{E_k} \cap e_{cd}^{E_k} = \emptyset \) and, by inductive hypothesis, that \( e_{cd}^{E_0} \cap e_{cd}^{E_0} = \emptyset \). Thus, by definition, we know that \( \mathcal{O} \models \neg \text{DisjointClasses } (c_1 e_{cd}) \). Therefore we have that \( \mathcal{O} \models \neg \text{DisjointClasses } (c_2 e_{cd}) \) and, by definition, we have that \( c_2^{E_0} \cap e_{cd}^{E_0} = \emptyset \). Since, by definition, when moving from \( \text{Chase}^j(\mathcal{O}) \) to \( \text{Chase}^{j+1}(\mathcal{O}) \) no axiom is removed from \( \text{Chase}^j(\mathcal{O}) \), we have that \( c_2^{E_k} \cap e_{cd}^{E_k} = \emptyset \), which leads to a contradiction.

**cr2** In this case we have that \( \text{ClassAssertion } (c_1 e) \) is in \( \text{Chase}^k(\mathcal{O}) \), \( \text{SubClassOf } (c_1 c_2) \) is in \( \mathcal{O} \), and does not exists \( e_{in} \) such that both \( \text{ObjectPropertyAssertion } (p e_{in}) \) and \( \text{ClassAssertion } (c_2 e_{in}) \) are in \( \text{Chase}^k(\mathcal{O}) \). Then \( \text{Chase}^{k+1}(\mathcal{O}) = \text{Chase}^k(\mathcal{O}) \cup \{ \text{ClassAssertion } (c_2 s) \} \) where \( s \in S^+_{k+1} \) and \( s \notin S^k \). Let \( e_{p, c_2} = \text{ObjectSomeValuesFrom } (p c_2) \) and \( e_{p, \text{Inv}(p)} = \text{ObjectSomeValuesFrom } (\text{ObjectInverseOf } (p) \text{ owl:Thing}) \). Then we have that \( e_{p, c_2}^{E_{k+1}} = e_{p, c_2}^{E_k} \cup \{ e \} \), \( e_2^{E_{k+1}} = e_2^{E_k} \cup \{ s \} \), \( e_{\text{Inv}(p)}^{E_{k+1}} = e_{\text{Inv}(p)}^{E_k} \cup \{ s \} \), and for every \( e_{cd} \) such that \( e_{cd} \neq e_{p, c_2}, e_{cd} \neq e_2, e_{cd} \neq e_{\text{Inv}(p)} \) we have that \( e_{cd}^{E_{k+1}} = e_{cd}^{E_k} \). Suppose there exists \( e_{cd} \) such that \( e_{cd}^{E_k} \cap e_{cd}^{E_k} = \emptyset \) and \( e_{cd}^{E_{k+1}} \cap e_{cd}^{E_{k+1}} \neq \emptyset \). Clearly this can happen if and only if \( e \in e_{cd}^{E_k} \). Then we know that \( c_1^{E_k} \cap e_{cd}^{E_k} = \emptyset \) and, by
inductive hypothesis, that $c_1^{E_0} \cap c_d^{E_0} \neq \emptyset$. Thus, by definition, we know that $O \models \neg \text{DisjointClasses } (c_1 e_{cd})$. Therefore we have that $O \models \neg \text{DisjointClasses } (e_{p,c_2} e_d)$ and, by definition, we have that $e_{p,c_2} \cap e_d^{E_0} \neq \emptyset$. Since, by definition, when moving from $\text{Chase}^1(O)$ to $\text{Chase}^{j+1}(O)$ no axiom is removed from $\text{Chase}^1(O)$, we have that $e_{p,c_2} \cap e_d^{E_0} \neq \emptyset$, which leads to a contradiction. Since $s \in \Delta_o^{k+1}$ and $s \notin \Delta_o^k$, we know that, for every $e_{cd}$ such that $e_{cd} \neq c_2$ and $e_{cd} \neq e_{Inv}(p)$, $e_{k+1}^{E_{k+1}} \cap e_{cd}^{E_{k+1}} = \emptyset$ if and only if $e_{k}^{E_{k}} \cap e_{cd}^{E_{k}} = \emptyset$ and $e_{k+1}^{E_{k+1}} \cap e_{cd}^{E_{k+1}} = \emptyset$ if and only if $e_{Inv}(p) \cap e_{cd}^{E_{k+1}} = \emptyset$. Then it remains only to show that $c_2 \cap e_{Inv}(p) \neq \emptyset$. Suppose, by contradiction, that $c_2 \cap e_{Inv}(p) = \emptyset$ and, by definition, that $O \models \text{DisjointClasses } (c_2 e_{Inv}(p))$, and therefore, we have also that $O \models \text{DisjointClasses } (e_{p,c_2} e_{p,c_2})$. But we know that $c_1 \neq \emptyset$ so we have that $\text{Can}^k(O) \models \neg \text{DisjointClasses } (c_1 e_{cd})$ and, by inductive hypothesis, that $\text{Can}^k(O) \models \neg \text{DisjointClasses } (c_1 e_{cd})$, and, by definition, that $O \models \neg \text{DisjointClasses } (c_1 e_{cd})$. By the fact that $\text{SubClassOf } (c_1 e_{p,c_2})$ is in $O$, we obtain that $O \models \neg \text{DisjointClasses } (e_{p,c_2} e_{p,c_2})$, which contradicts the hypothesis that $O$ is satisfiable;

cr3 In this case we have that $\text{ClassAssertion } (c_1 e)$ is in $\text{Chase}^k(O)$, $\text{SubClassOf } (c_1 e_{cd})$ $\text{ObjectSomeValuesFrom } (\text{ObjectInverseOf } (p e_{cd}))$ is in $O$, and does not exists $e_{in}$ such that both $\text{ObjectPropertyAssertion } (p e_{in} e)$ and $\text{ClassAssertion } (e_{cd} e_{in})$ are in $\text{Chase}^k(O)$. Then $\text{Chase}^{k+1}(O) = \text{Chase}^k(O) \cup \{ \text{ClassAssertion } (c_2 e), \text{ObjectPropertyAssertion } (p s e) \}$ such that $s \in S^k_{p,c_2}$ and $s \notin S^k_p$. Let $e_{p,c_2} = \text{ObjectInverseOf } (p e_{cd})$ and $e_p = \text{ObjectSomeValuesFrom } (p \text{owl:Thing})$. Then we have that $e_{p,c_2}^{E_{k+1}} = e_{p,c_2}^{E_k} \cup \{ e \}$, $e_{k+1}^{E_k} = e_{k}^{E_k} \cup \{ s \}$, and for every $e_{cd}$ such that $e_{cd} \neq p, e_{cd} \neq c_2$, and $e_{cd} \neq e_p$, we have that $e_{k+1}^{E_{k+1}} = e_{k}^{E_k} \cup \{ e \}$. Suppose there exists $e_{cd}$ such that $e_{k+1}^{E_{k+1}} \neq \emptyset$ and $e_{k+1}^{E_{k+1}} \neq \emptyset$. Clearly this can happen if and only if $e \in e_{k+1}^{E_k}$. Then we know that $c_1 \neq \emptyset$ by, and inductive hypothesis, that $c_1^{E_{0}} \neq \emptyset$. Thus, by definition, we know that $O \models \neg \text{DisjointClasses } (c_1 e_{cd})$. Therefore we have that $O \models \neg \text{DisjointClasses } (e_{p,c_2} e_{cd})$ and, by definition, we have that $e_{p,c_2} \cap e_{cd}^{E_0} \neq \emptyset$. Since, by definition, when moving from $\text{Chase}^1(O)$ to $\text{Chase}^{j+1}(O)$ no axiom is removed from $\text{Chase}^1(O)$, we have that $e_{p,c_2} \cap e_{cd}^{E_0} \neq \emptyset$, which leads to a contradiction. Then, by inductive hypothesis, we have that $e_{p,c_2} \cap e_{cd}^{E_0} = \emptyset$ and, by definition, that $O \models \text{DisjointClasses } (c_2 e_{p,c_2})$, and therefore, we have also that $O \models \text{DisjointClasses } (e_{p,c_2} e_{p,c_2})$. But we know that $c_1 \neq \emptyset$ so we have that $\text{Can}^k(O) \models \neg \text{DisjointClasses } (c_1 e_{cd})$ and, by inductive hypothesis, that $\text{Can}^k(O) \models \neg \text{DisjointClasses } (c_1 e_{cd})$, and, by definition, that $O \models \neg \text{DisjointClasses } (c_1 e_{cd})$. By the fact that $\text{SubClassOf } (c_1 e_{p,c_2})$ is in $O$, we obtain that $O \models \neg \text{DisjointClasses } (e_{p,c_2} e_{p,c_2})$, which contradicts the hypothesis that $O$ is satisfiable;

cr4 In this case we have that $\text{ClassAssertion } (c e)$ is in $\text{Chase}^k(O)$, $\text{SubClassOf } (c e_{cd})$ $\text{DataSomeValuesFrom } (a d)$ is in $O$, and does not exists $v$ such that both $\text{DataPropertyAssertion } (p e v)$ and $\text{DatatypeAssertion } (d v)$ are in $\text{Chase}^k(O)$. Then $\text{Chase}^{k+1}(O) = \text{Chase}^k(O) \cup \{ \text{DataPropertyAssertion } (a e v), \text{DatatypeAssertion } (d s) \}$ where $s \in S^k_{v,c_2}$ and $s \notin S^k_p$. Let $e_a = \text{DataSomeValuesFrom }
In this case we have that \( \text{ObjectPropertyAssertion} \) is in \( \text{Chase}^k(\mathcal{O}) \), which leads to a contradiction. Since \( \forall \mathcal{E} \in \Delta^0 \) and, by definition, that \( \mathcal{E}_0 \) and, by inductive hypothesis, that \( \mathcal{E}_{k+1} \), we have that \( \mathcal{E}_0 \cap \mathcal{E}_{k+1} \neq \emptyset \). Thus, by definition, we know that \( \mathcal{O} \models \neg \text{DisjointClasses}(c_{e_d}) \). Therefore we have that \( \mathcal{O} \models \neg \text{DisjointClasses}(c_{e_d}) \) and, by definition, we have that \( \mathcal{E}_0 \cap \mathcal{E}_{k+1} \neq \emptyset \). Since, by definition, when moving from \( \text{Chase}_j(\mathcal{O}) \) to \( \text{Chase}^{j+1}(\mathcal{O}) \) no axiom is removed from \( \text{Chase}^j(\mathcal{O}) \), we have that \( \mathcal{E}_0 \cap \mathcal{E}_{k+1} \neq \emptyset \) which leads to a contradiction;
also that $\mathcal{O} \models \text{DisjointClasses}(e_{p_2, c}, e_{p_2, c})$. But we know that $e_{p_1}^{E_k} \neq \emptyset$, so we have that $\text{Can}^k(\mathcal{O}) \models \neg \text{DisjointClasses}(e_{p_1}, e_{p_1})$ and, by inductive hypothesis, that $\text{Can}^l(\mathcal{O}) \models \neg \text{DisjointClasses}(e_{p_1}, e_{p_1})$, and, by definition, that $\mathcal{O} \models \neg \text{DisjointClasses}(e_{p_1}, e_{p_1})$. By the fact that $\text{SubClassOf}(e_{p_1}, e_{p_2, c})$ is in $\mathcal{O}$, we obtain that $\mathcal{O} \models \neg \text{DisjointClasses}(e_{p_2, c}, e_{p_2, c})$, which contradicts the hypothesis that $\mathcal{O}$ is satisfiable;

cr7 In this case we have that $\text{ObjectPropertyAssertion}(p_1 \ e_1 \ e_2)$ is in $\text{Chase}^k(\mathcal{O})$, $\text{SubClassOf}(\text{ObjectSomeValuesFrom}(p_1 \ \text{owl:Thing}) \ \text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(p_2) \ c))$ is in $\mathcal{O}$, and does not exists $e_{in}$ such that both $\text{ObjectPropertyAssertion}(p_2 \ \text{e}_{in} \ c)$ and $\text{ClassAssertion}(c \ \text{e}_{in})$ are in $\text{Chase}^k(\mathcal{O})$. Then $\text{Chase}^{k+1}(\mathcal{O}) = \text{Chase}^k(\mathcal{O}) \cup \{\text{ClassAssertion}(c \ s), \text{ObjectPropertyAssertion}(p_2 \ s \ e)\}$ where $s \in S_0^{k+1}$ and $s \notin S_0^k$. Let $e_{p_1} = \text{ObjectSomeValuesFrom}(p_1 \ \text{owl:Thing}) \ e_{\text{Inv}(p_2), c} = \text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(p_2) \ c)$. Then we have that $e_{p_1}^E \ e_{\text{Inv}(p_2), c} \ e = e_{\text{Inv}(p_2), c} \cup \{c\}$, $e_{E_{k+1}} = e_E = e_{E_{k+1}} \cup \{c\}$, and, for every $e_{cl}$ such that $e_{cl} \neq e_{\text{Inv}(p_2), c}$, $e_{cl} \neq e_{p_2}$ and $e_{E_{k+1}} \neq e_{p_2}$, we have that $e_{cl}^{E_{k+1}} = e_{cl}^E$. Suppose there exists $e_{cl}$ such that $e_{E_{k+1}} \cap e_{cl} = \emptyset$ and $e_{E_{k+1}} \cap e_{cl} = \emptyset$. Clearly this can happen if and only if $e_{cl} \notin e_{E_{k+1}}$. Then we know that $e_{cl} \cap e_{E_{k+1}} = \emptyset$, and, by inductive hypothesis, $e_{cl} \cap e_{E_{k+1}} = \emptyset$. Thus, by definition, we know that $\mathcal{O} \models \neg \text{DisjointClasses}(e_{p_1}, e_{cl})$. Therefore we have that $\mathcal{O} \models \neg \text{DisjointClasses}(e_{\text{Inv}(p_2), c}, e_{cl})$ and, by definition, we have that $e_{E_{k+1}} \cap e_{cl} = \emptyset$. Since, by definition, when moving from $\text{Chase}^j(\mathcal{O})$ to $\text{Chase}^{j+1}(\mathcal{O})$ no axiom is removed from $\text{Chase}^j(\mathcal{O})$, we have that $e_{\text{Inv}(p_2), c} \cap e_{cl} = \emptyset$, which leads to a contradiction. Since $s \in S_0^{k+1}$ and $s \notin S_0^k$, we know that, for every $e_{cl}$ such that $e_{cl} \neq c$ and $e_{cl} \neq e_{p_2}$, $e_{E_{k+1}} \cap e_{cl} = \emptyset$ if and only if $c \in e_{E_{k+1}}$. Then it remains only to show that $c \in e_{E_{k+1}} \neq \emptyset$. Suppose, by contradiction, that $c \in e_{E_{k+1}} = \emptyset$. Then, by inductive hypothesis, we know that $c \in e_{E_{k+1}} = \emptyset$, and, by definition, that $\mathcal{O} \models \text{DisjointClasses}(c, e_{p_2})$, and therefore, we have also that $\mathcal{O} \models \text{DisjointClasses}(e_{\text{Inv}(p_2), c}, e_{\text{Inv}(p_2), c})$. But we know that $e_{E_{k+1}} \neq \emptyset$, and, so we have that $\text{Can}^k(\mathcal{O}) \models \neg \text{DisjointClasses}(e_{p_1}, e_{p_1})$, and, by inductive hypothesis, that $\text{Can}^l(\mathcal{O}) \models \neg \text{DisjointClasses}(e_{p_1}, e_{p_1})$, and, by definition, that $\mathcal{O} \models \neg \text{DisjointClasses}(e_{p_1}, e_{p_1})$. By the fact that $\text{SubClassOf}(e_{p_1}, e_{p_2, c})$ is in $\mathcal{O}$, we obtain that $\mathcal{O} \models \neg \text{DisjointClasses}(e_{\text{Inv}(p_2), c}, e_{\text{Inv}(p_2), c})$, which contradicts the hypothesis that $\mathcal{O}$ is satisfiable;

cr8 In this case we have that $\text{ObjectPropertyAssertion}(p \ e_1 \ e_2)$ is in $\text{Chase}^k(\mathcal{O})$, $\text{SubClassOf}(\text{ObjectSomeValuesFrom}(p \ \text{owl:Thing}) \ \text{DataSomeValuesFrom}(a \ d))$ is in $\mathcal{O}$, and does not exists $v$ such that both $\text{DataPropertyAssertion}(a \ e_1 \ v)$ and $\text{DatatypeAssertion}(d \ v)$ are in $\text{Chase}^k(\mathcal{O})$. Then $\text{Chase}^{k+1}(\mathcal{O}) = \text{Chase}^k(\mathcal{O}) \cup \{\text{DataPropertyAssertion}(a \ e_1 \ s), \text{DatatypeAssertion}(d \ s)\}$ where $s \in S_v^{k+1}$ and $s \notin S_v^k$. Let $e_a = \text{DataSomeValuesFrom}(a \ d)$ and $e_p = \text{ObjectSomeValuesFrom}(p \ \text{owl:Thing})$. Then we have that $e_a^{E_{k+1}} = e_a^E \cup \{e_1\}$, and for every $e_{cl} \neq e_a$ we have that $e_{cl}^{E_{k+1}} = e_{cl}^E$. Suppose there exists $e_{cl}$ such that $e_a \cap e_{cl} = \emptyset$ and $e_a \cap e_{cl} = \emptyset$. Clearly this can happen if and only if $e_1 \in e_{cl}$. Then we know
that \( e_p^{E_0} \cap e_c^{E_0} \neq \emptyset \) and, by inductive hypothesis, that \( e_p^{E_0} \cap e_c^{E_0} \neq \emptyset \). Thus, by definition, we know that \( \mathcal{O} \models \neg \text{DisjointClasses} (e_p^{E_0} \cap e_c^{E_0}) \). Therefore we have that \( \mathcal{O} \models \neg \text{DisjointClasses} (e_p, e_c) \) and, by definition, we have that \( e_p^{E} \cap e_c^{E} \neq \emptyset \).

Since, by definition, when moving from \( \text{Chase}^j(\mathcal{O}) \) to \( \text{Chase}^{j+1}(\mathcal{O}) \) no axiom is removed from \( \text{Chase}^j(\mathcal{O}) \), we have that \( e_p^{E} \cap e_c^{E} \neq \emptyset \) which leads to a contradiction;

cr9 In this case we have that \( \text{ObjectPropertyAssertion} (p, e_1, e_2) \) is in \( \text{Chase}^k(\mathcal{O}) \), \( \text{SubClassOf} (\text{ObjectSomeValuesFrom} (\text{ObjectInverseOf} (p) \text{owl:Thing}) c) \) is in \( \mathcal{O} \), \( \text{ClassAssertion} (c, e_2) \) is not in \( \text{Chase}^k(\mathcal{O}) \), and \( \text{Chase}^{k+1}(\mathcal{O}) = \text{Chase}^k(\mathcal{O}) \cup \{ \text{ClassAssertion} (c, e_2) \} \). Then we have that \( c_{p, e_2}^{E_k} = e_p^{E_k} \). Suppose there exists \( e_c \) such that \( e_p^{E_k} \cap e_c^{E_k} = \emptyset \) and \( e_{p, e_2}^{E_k} \cap e_c^{E_k} = e_{p, e_2}^{E_k} \). Clearly this can happen if and only if \( e_2 \in e_c^{E_k} \). Let \( e_{p, e_2}^{E_k} \) and \( e_c \not\in e_c^{E_k} \) and, by inductive hypothesis, that \( e_p^{E_0} \cap e_c^{E_0} \neq \emptyset \). Thus, by definition, we know that \( \mathcal{O} \models \neg \text{DisjointClasses} (e_p, e_c) \). Therefore we have that \( \mathcal{O} \models \neg \text{DisjointClasses} (e_{p, e_2}, e_c) \) and, by definition, we have that \( e_{p, e_2}^{E_0} \cap e_c^{E_0} \neq \emptyset \). Since, by definition, when moving from \( \text{Chase}^j(\mathcal{O}) \) to \( \text{Chase}^{j+1}(\mathcal{O}) \) no axiom is removed from \( \text{Chase}^j(\mathcal{O}) \), we have that \( e_{p, e_2}^{E} \cap e_c^{E} \neq \emptyset \), which leads to a contradiction;

cr10 In this case we have that \( \text{ObjectPropertyAssertion} (p_1, e_1, e_2) \) is in \( \text{Chase}^k(\mathcal{O}) \), \( \text{SubClassOf} (\text{ObjectSomeValuesFrom} (\text{ObjectInverseOf} (p_1) \text{owl:Thing}) \text{ObjectSomeValuesFrom} (p_2, c)) \) is in \( \mathcal{O} \), and does not exists \( e_{p_1} \) such that both \( \text{ObjectPropertyAssertion} (p_2, e_2, e_{p_1}) \) and \( \text{ClassAssertion} (c, e_{p_1}) \) are in \( \text{Chase}^k(\mathcal{O}) \). Then \( \text{Chase}^{k+1}(\mathcal{O}) = \text{Chase}^k(\mathcal{O}) \cup \{ \text{ClassAssertion} (c, e_{p_1}) \} \). Suppose \( e_{p_2, e_2}^{E_k} \cap e_c^{E_k} = \emptyset \) and \( e_{p_2, e_2}^{E_k} \cap e_{p_1}^{E_k} = \emptyset \). Clearly this can happen if and only if \( e_2 \not\in e_c^{E_k} \) and, by inductive hypothesis, that \( e_{p_1}^{E_0} \cap e_c^{E_0} \neq \emptyset \). Thus, by definition, we know that \( \mathcal{O} \models \neg \text{DisjointClasses} (e_{p_1}, e_c) \). Therefore we have that \( \mathcal{O} \models \neg \text{DisjointClasses} (e_{p_2, e_2}, e_c) \) and, by definition, we have that \( e_{p_2, e_2}^{E_0} \cap e_c^{E_0} \neq \emptyset \). Since, by definition, when moving from \( \text{Chase}^j(\mathcal{O}) \) to \( \text{Chase}^{j+1}(\mathcal{O}) \) no axiom is removed from \( \text{Chase}^j(\mathcal{O}) \), we have that \( e_{p_2, e_2}^{E_0} \cap e_c^{E_0} \neq \emptyset \), which leads to a contradiction.
In this case we have that $\text{ObjectPropertyAssertion}(p_2 e_in e_2)$ is in $\text{Chase}^k(\mathcal{O})$, $\text{SubClassOf}(\text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(p_1) \text{owl:Thing}) \text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(p_2) e))$ is in $\mathcal{O}$, and does not exists $e_{in}$ such that both $\text{ObjectPropertyAssertion}(p_2 e_{in} e_2)$ and $\text{ClassAssertion}(c e_{in})$ are in $\text{Chase}^k(\mathcal{O})$. Then $\text{Chase}^{k+1}(\mathcal{O}) = \text{Chase}^k(\mathcal{O}) \cup \{\text{ClassAssertion}(c s), \text{ObjectPropertyAssertion}(p_2 s e_2)\}$ where $s \in \mathcal{S}_d^{k+1}$ and $s \notin \mathcal{S}_d^k$. Let $e_{p_1} = \text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(p_1) \text{owl:Thing}) e_{\text{Inv}(p_2),c} = \text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(p_2) e), e_{p_2} = \text{ObjectSomeValuesFrom}(p_2 \text{owl:Thing})$. Then we have that $e_{\text{Inv}(p_2),c}^k = e_{\text{Inv}(p_2),c} \cup \{e_2\}, e_{\text{Inv}(p_2),c}^{k+1} = e_{\text{Inv}(p_2),c}^k \cup \{s\}$, $e_{p_2} = e_{p_2}^k \cup \{s\}$, and for every $e_{cd}$ such that $e_{cd} \neq e_{\text{Inv}(p_2),c}^k$, $e_{cd} \neq c$, and $e_{cd} \neq e_{p_2}$ we have that $e_{\text{Inv}(p_2),c}^k \cap e_{cd} = \emptyset$ and $e_{\text{Inv}(p_2),c}^{k+1} \cap e_{cd} \neq \emptyset$. Clearly this can happen if and only if $e_2 \in e_{cd}^k$. Then we know that $e_{p_1} \cap e_{cd} \neq \emptyset$ and, by inductive hypothesis, that $e_{p_1} \cap e_{cd} \neq \emptyset$. Thus, by definition, we know that $\mathcal{O} \models \neg \text{DisjointClasses}(e_{p_1} e_{cd})$. Therefore we have that $\mathcal{O} \models \neg \text{DisjointClasses}(e_{p_1} e_{\text{Inv}(p_2),c} e_{\text{cd}})$ and, by definition, we have that $\mathcal{O} \models \neg \text{DisjointClasses}(e_{\text{Inv}(p_2),c} e_{\text{cd}}) \neq \emptyset$. Since, by definition, when moving from $\text{Chase}^j(\mathcal{O})$ to $\text{Chase}^{j+1}(\mathcal{O})$ no axiom is removed from $\text{Chase}^j(\mathcal{O})$, we have that $e_{\text{Inv}(p_2),c}^k \cap e_{cd} \neq \emptyset$, which leads to a contradiction. Since $s \in \Delta_o^{k+1}$ and $s \notin \Delta_o^k$, we know that, for every $e_{cd}$ such that $e_{cd} \neq c$ and $e_{cd} \neq e_{p_2}$, $e_{\text{Inv}(p_2),c}^{k+1} \cap e_{cd} = \emptyset$ if and only if $e_{\text{Inv}(p_2),c} e_{cd} = \emptyset$ and $e_{p_2} e_{\text{Inv}(p_2),c} e_{cd} = \emptyset$ if and only if $e_{p_2} e_{\text{Inv}(p_2),c} e_{cd} = \emptyset$. Then it remains only to show that $e_{\text{Inv}(p_2),c} e_{cd} = \emptyset$. Suppose, by contradiction, that $e_{\text{Inv}(p_2),c} e_{cd} = \emptyset$. Then, by inductive hypothesis, we know that $e_{\text{Inv}(p_2),c} e_{cd} = \emptyset$, and, by definition, we know that $\mathcal{O} \models \text{DisjointClasses}(e_{p_1} e_{p_1})$, and therefore, we have also that $\mathcal{O} \models \text{DisjointClasses}(e_{p_1} e_{\text{Inv}(p_2),c})$. But we know that $e_{\text{Inv}(p_2),c} e_{cd} = \emptyset$, so we have that $\text{Can}^k(\mathcal{O}) \models \neg \text{DisjointClasses}(e_{p_1} e_{p_1})$ and that, by inductive hypothesis, $\text{Can}^j(\mathcal{O}) \models \neg \text{DisjointClasses}(e_{p_1} e_{p_1})$, and that, by definition, $\mathcal{O} \models \neg \text{DisjointClasses}(e_{p_1} e_{p_1})$. By the fact that $\text{ObjectClassOf}(e_{p_1} e_{p_2},c)$ is in $\mathcal{O}$, we obtain that $\mathcal{O} \models \neg \text{DisjointClasses}(e_{p_1} e_{\text{Inv}(p_2),c})$, which contradicts the hypothesis that $\mathcal{O}$ is satisfiable;
and, by definition, we have that \(e_a^E \cap e_d^E \neq \emptyset\). Since, by definition, when moving from \(Chase^j(\mathcal{O})\) to \(Chase^{j+1}(\mathcal{O})\) no axiom is removed from \(Chase^j(\mathcal{O})\), we have that \(e_a^E \cap e_d^E \neq \emptyset\) which leads to a contradiction;

cr13 In this case we have that DataPropertyAssertion \((a \in lt)\) is in \(Chase^k(\mathcal{O})\), SubClassOf (DataSomeValuesFrom \((a \in d)\) \(c\)) is in \(\mathcal{O}\), ClassAssertion \((c \in)\) is not in \(Chase^k(\mathcal{O})\), and \(Chase^{k+1}(\mathcal{O}) = Chase^k(\mathcal{O}) \cup \{\text{ClassAssertion} (c \in)\}\). Then we have that \(e_a^{E_{k+1}} = e_a^E \cup \{c\}\), and for every \(e_d \neq c\) we have that \(e_d^{E_{k+1}} = e_d^E\). Suppose there exists \(e_d\) such that \(e_d^E \cap e_d^E = \emptyset\) and \(e_d^{E_{k+1}} \cap e_d^{E_{k+1}} \neq \emptyset\). Clearly this can happen if and only if \(e \in e_d^E\). Let \(e_a = \text{DataSomeValuesFrom} (a \in d)\). Then we know that \(e_a^{E_d} \cap e_d^E \neq \emptyset\). Since, by definition, when moving from \(Chase^j(\mathcal{O})\) to \(Chase^{j+1}(\mathcal{O})\) no axiom is removed from \(Chase^j(\mathcal{O})\), we have that \(e_d^E \cap e_d^E \neq \emptyset\), which leads to a contradiction;

cr14 In this case we have that DataPropertyAssertion \((a \in lt)\) is in \(Chase^k(\mathcal{O})\), SubClassOf (DataSomeValuesFrom \((a \in d)\) ObjectSomeValuesFrom \((p \in c)\)) is in \(\mathcal{O}\), and does not exists \(e_{in}\) such that both ObjectPropertyAssertion \((p \in e_{in})\) and ClassAssertion \((c \in e_{in})\) are in \(Chase^k(\mathcal{O})\). Then \(Chase^{k+1}(\mathcal{O}) = Chase^k(\mathcal{O}) \cup \{\text{ClassAssertion} (c \in)\, \text{ObjectPropertyAssertion} (p \in s)\}\) where \(s \in S_{o_{k+1}}^0\) and \(s \notin S_{o}^k\). Let \(e_a = \text{DataSomeValuesFrom} (a \in d)\), \(e_p,c = \text{ObjectSomeValuesFrom} (p \in c)\), \(e_{Inv}(p) = \text{ObjectSomeValuesFrom} (\text{ObjectInverseOf} (p) \text{owl:Thing})\). Then we have that \(e_p,c^{E_{k+1}} = e_p,c \cup \{e\}, e^{E_{k+1}} = e^E \cup \{s\}, e_{Inv}(p) = e_{Inv}(p) \cup \{s\}\), and for every \(e_d\) such that \(e_d \neq e_p,c, e_d \neq c\), and \(e_d \neq e_{Inv}(p)\) we have that \(e_d^{E_{k+1}} = e_d^E\). Suppose there exists \(e_d\) such that \(e_d^E \cap e_d^E = \emptyset\) and \(e_d^{E_{k+1}} \cap e_d^{E_{k+1}} \neq \emptyset\). Clearly this can happen if and only if \(e \in e_d^E\). Then we know that \(e_d^E \cap e_d^E \neq \emptyset\). Since, by definition, when moving from \(Chase^j(\mathcal{O})\) to \(Chase^{j+1}(\mathcal{O})\) no axiom is removed from \(Chase^j(\mathcal{O})\), we have that \(e_d^E \cap e_d^E \neq \emptyset\), which leads to a contradiction. Since \(s \in \Delta_{k+1}^{0}\) and \(s \notin \Delta_k\), we know that, for every \(e_d\) such that \(e_d \neq c\) and \(e_d \neq e_{Inv}(p)\), \(e_d^{E_{k+1}} \cap e_d^{E_{k+1}} = \emptyset\) if and only if \(e_d^E \cap e_d^E = \emptyset\) and \(e_{Inv}(p) \cap e_d^E = \emptyset\) if and only if \(e_d^E \cap e_d^E = \emptyset\). Then it remains only to show that \(e^E \cap e_{Inv}(p) \neq \emptyset\). Suppose, by contradiction, that \(e^E \cap e_{Inv}(p) = \emptyset\). Then, by inductive hypothesis, we know that \(e_a^E \cap e_{Inv}(p) = \emptyset\) and, by definition, that \(\mathcal{O} \models \text{DisjointClasses} (c e_{Inv}(p))\), and therefore, we have also that \(\mathcal{O} \models \text{DisjointClasses} (e_p,c e_{Inv}(p))\). But we know that \(e_a^E \neq \emptyset\), so we have that \(\text{Can}^k(\mathcal{O}) \models \neg \text{DisjointClasses} (e_a c_{Inv}(p))\) and, by inductive hypothesis, that \(\text{Can}^{k+1}(\mathcal{O}) \models \neg \text{DisjointClasses} (e_a c_{Inv}(p))\) and, by definition, that \(\mathcal{O} \models \neg \text{DisjointClasses} (e_a c_{Inv}(p))\). By the fact that \(\text{SubClassOf} (e_a e_{Inv}(p)) \in \mathcal{O}\), we obtain that \(\mathcal{O} \models \neg \text{DisjointClasses} (e_p,c e_{Inv}(p))\), which contradicts the hypothesis that \(\mathcal{O}\) is satisfiable;

cr15 In this case we have that DataPropertyAssertion \((a \in lt)\) is in \(Chase^k(\mathcal{O})\), Sub-
ClassOf (DataSomeValuesFrom (a d) ObjectSomeValuesFrom (ObjectInverseOf (p c))) is in \( \mathcal{O} \), and does not exists \( e_{in} \) such that both ObjectPropertyAssertion (p \( e_{in} \) c) and ClassAssertion (c \( e_{in} \)) are in \( \text{Chase}^k(\mathcal{O}) \). Then \( \text{Chase}^{k+1}(\mathcal{O}) = \text{Chase}^k(\mathcal{O}) \cup \{ \text{ClassAssertion (c s), ObjectPropertyAssertion (p s e) \} \) where \( s \in S^k_0 \) and \( s \notin S^k \). Let \( e_a = \text{DataSomeValuesFrom (a d), e_{Inv}(p) = \text{ObjectSomeValuesFrom (ObjectInverseOf (p c))} \) then we have that \( e_{Inv}(p),c = \text{ObjectSomeValuesFrom (ObjectInverseOf (p c))} \). Therefore we have that \( e_{Inv}(p),c = \text{ObjectSomeValuesFrom (ObjectInverseOf (p c))} \). Then we have that \( e_{Inv}(p),c = e_{Inv}(p),e \cup \{ \}, c_{E^{k+1}} = e_{E^k} \cup \{s\}, e_p = e_{E^k} \cup \{s\} \), and for every \( e_{cl} \) such that \( e_{cl} \neq e_{Inv}(p),c, e_{cl} \neq c, \) and \( e_{cl} \neq e_p \) we have that \( e_{cl} \neq e_{E^{k+1}} \). Suppose there exists \( e_{cl} \) such that \( e_{cl} \neq e_{Inv}(p),c, e_{cl} \neq c, \) and \( e_{cl} \neq e_p \) and, by inductive hypothesis, that \( e_{cl} \neq e_{E^{k+1}} \). Thus, by definition, we know that \( \mathcal{O} \models \neg \text{DisjointClasses (e_a e_{cl})} \). Therefore we have that \( \mathcal{O} \models \neg \text{DisjointClasses (e_{Inv}(p),c e_{cl})} \) and, by definition, we have that \( e_{Inv}(p),c \cap e_{E^0} \neq \emptyset \). Since, by definition, when moving from \( \text{Chase}^j(\mathcal{O}) \) to \( \text{Chase}^{j+1}(\mathcal{O}) \) no axiom is removed from \( \text{Chase}^j(\mathcal{O}) \), we have that \( e_{Inv}(p),c \cap e_{E^0} \neq \emptyset \). which leads to a contradiction.

Since \( s \in \Delta^k_0 \) and \( s \notin \Delta^k \), we know that, for every \( e_{cl} \) such that \( e_{cl} \neq c \) and \( e_{cl} \neq e_p \), \( e_{E^{k+1}} \cap e_{E^{k+1}} = \emptyset \) if and only if \( e_{E^k} \cap e_{E^k} = \emptyset \) and \( e_{E^{k+1}} \cap e_{E^{k+1}} = \emptyset \) if and only if \( e_p \cap e_p = \emptyset \). Then it remains only to show that \( e_{E^k} \cap e_p \neq \emptyset \). So, by contradiction, that \( e_{E^k} \cap e_p = \emptyset \). Then, by inductive hypothesis, we know that \( e_{E^k} \cap e_p = \emptyset \) and, by definition, we know that \( \mathcal{O} \models \neg \text{DisjointClasses (e_a e_{cl})} \) and, therefore, we have also that \( \mathcal{O} \models \neg \text{DisjointClasses (e_{Inv}(p),c e_{Inv}(p),c)} \). But we know that \( e_{E^k} \neq \emptyset \), so we know that \( \text{Can}^k(\mathcal{O}) \models \neg \text{DisjointClasses (e_a e_{cl})} \) and, by inductive hypothesis, \( \text{Can}^0(\mathcal{O}) \models \neg \text{DisjointClasses (e_a e_{cl})} \) and, by definition, that \( \mathcal{O} \models \neg \text{DisjointClasses (e_a e_{cl})}. \) By the fact that SubClassOf (e_a e_{Inv}(p),c) is in \( \mathcal{O} \), we obtain that \( \mathcal{O} \models \neg \text{DisjointClasses (e_{Inv}(p),c e_{Inv}(p),c)} \), which contradicts the hypothesis that \( \mathcal{O} \) is satisfiable;

in this case we have that DataPropertyAssertion (a_1 e l t) is in \( \text{Chase}^k(\mathcal{O}) \), SubClassOf (DataSomeValuesFrom (a_1 d_1) DataSomeValuesFrom (a_2 d_2)) is in \( \mathcal{O} \), and does not exists \( v \) such that both DataPropertyAssertion (a_1 e v) and DataPropertyAssertion (d_2 v) are in \( \text{Chase}^k(\mathcal{O}) \). Then \( \text{Chase}^{k+1}(\mathcal{O}) = \text{Chase}^k(\mathcal{O}) \cup \{ \text{DataPropertyAssertion (a_2 e s), DataPropertyAssertion (d_2 s) \} \) where \( s \in S^k_0 \) and \( s \notin S^k \). Let \( e_a = \text{DataSomeValuesFrom (a_1 d_1)} \) and let \( e_{a_2} = \text{DataSomeValuesFrom (a_2 d_2)} \). Then we have that \( e_{a_2} = e_{a_2} \cup \{ e \} \), and for every \( e_{cl} \neq e_{a_2} \) we have that \( e_{cl} = e_{a_2} \). Suppose there exists \( e_{cl} \) such that \( e_{a_2} \cap e_{a_2} = \emptyset \) and \( e_{a_2} \cap e_{a_2} = \emptyset \). Clearly this can happen if and only if \( e \in e_{a_2} \). Then we know that \( e_{a_2} \cap e_{a_2} = \emptyset \) and, by inductive hypothesis, that \( e_{a_2} \cap e_{a_2} = \emptyset \). Thus, by definition, we know that \( \mathcal{O} \models \neg \text{DisjointClasses (e_{a_2} e_{cl})} \). Therefore we have that \( \mathcal{O} \models \neg \text{DisjointClasses (e_{a_2} e_{cl})} \) and, by definition, we have that \( e_{a_2} \cap e_{a_2} = \emptyset \). Since, by definition, when moving from \( \text{Chase}^j(\mathcal{O}) \) to \( \text{Chase}^{j+1}(\mathcal{O}) \) no axiom is removed from \( \text{Chase}^j(\mathcal{O}) \), we have that \( e_{a_2} \cap e_{a_2} = \emptyset \). which leads to a contradiction.

in this case we have that ObjectPropertyAssertion (p_1 e_1 e_2) is in \( \text{Chase}^k(\mathcal{O}) \), SubObjectPropertyOf (p_1 p_2) is in \( \mathcal{O} \), ObjectPropertyAssertion (p_2 e_1 e_2) is not in \( \text{Chase}^k(\mathcal{O}) \), and \( \text{Chase}^{k+1}(\mathcal{O}) = \text{Chase}^k(\mathcal{O}) \cup \{ \text{ObjectPropertyAssertion (p_1 p_2) \} \).
(p₂ e₁ e₂). Let e₁ = ObjectSomeValuesFrom (p₁ owl:Thing), e₁Inv(p₁) = ObjectSomeValuesFrom (ObjectInverseOf (p₁) owl:Thing), e₂ = ObjectSomeValuesFrom (p₂ owl:Thing), and e₂Inv(p₂) = ObjectSomeValuesFrom (ObjectInverseOf (p₂) owl:Thing). Then we have that e₂Ek+1 = e₂Ek ∪ {e₁}, e₂Inv(p₂) = e₂E₀ ∪ {e₁} and for every eₖd such that eₖd ≠ e₂ and eₖd ≠ e₂Inv(p₂) we have that eₖEₖ₊₁ = eₖEₖ. Suppose there exists eₖd such that eₖEₖ₊₁ ∩ eₖEₖ = ∅ and e₂Eₖ₊₁ ∩ eₖEₖ +₁ ≠ ∅. Clearly this can happen if and only if e₁ ∈ e₂Eₖ +₁. Then we know that e₂Eₖ₊₁ ∩ eₖEₖ +₁ ≠ ∅ and, by inductive hypothesis, that e₁Eₖ ∩ e₀Eₖ +₀ ≠ ∅. Thus, by definition, we know that O |= ¬ DisjointClasses(e₂ Eₖ). Therefore we have that O |= ¬ DisjointClasses(e₂ Eₖ) and, by definition, we have that e₂Eₖ₊₁ ∩ eₖEₖ +₁ ≠ ∅. Since, by definition, when moving from Chase₂(O) to Chase₂₊₁(O) no axiom is removed from Chase₂(O), we have that eₖEₖ₊₁ ∩ eₖEₖ +₁ ≠ ∅ which leads to a contradiction. Similarly we can show that does not exist eₖd such that eₖEₖ₊₁ ∩ eₖEₖ +₁ = ∅ and eₖEₖ₊₁ ∩ eₖEₖ +₁ ≠ ∅;

cr18 In this case we have that ObjectPropertyAssertion (p₁ e₁ e₂) is in Chase₅(O), SubObjectPropertyOf (p₁ ObjectInverseOf (p₂)) is in O, ObjectPropertyAssertion (p₂ e₂ e₁) is not in Chase₅(O), and Chase₅₊₁(O) = Chase₅(O) ∪ {ObjectPropertyAssertion (p₂ e₂ e₁)}. Let e₁ = ObjectSomeValuesFrom (p₁ owl:Thing), e₁Inv(p₁) = ObjectSomeValuesFrom (ObjectInverseOf (p₁) owl:Thing), e₂ = ObjectSomeValuesFrom (p₂ owl:Thing), and e₂Inv(p₂) = ObjectSomeValuesFrom (ObjectInverseOf (p₂) owl:Thing). Then we have that e₂Eₖ₊₁ = e₂Eₖ ∪ {e₁}, e₂Inv(p₂) = e₂E₀ ∪ {e₁} and for every eₖd such that eₖd ≠ e₂ and eₖd ≠ e₂Inv(p₂) we have that eₖEₖ₊₁ = eₖEₖ. Suppose there exists eₖd such that eₖEₖ₊₁ ∩ eₖEₖ = ∅ and e₂Eₖ₊₁ ∩ eₖEₖ +₁ ≠ ∅. Clearly this can happen if and only if e₂ ∈ eₖEₖ +₁. Then we know that e₁Eₖ ∩ eₖEₖ +₁ ≠ ∅ and, by inductive hypothesis, that e₁Eₖ ∩ e₀Eₖ +₀ ≠ ∅. Thus, by definition, we know that O |= ¬ DisjointClasses(e₁ eₖd). Therefore we have that O |= ¬ DisjointClasses(e₁ eₖd) and, by definition, we have that e₁Eₖ₊₁ ∩ eₖEₖ +₁ ≠ ∅. Since, by definition, when moving from Chase₅(O) to Chase₅₊₁(O) no axiom is removed from Chase₅(O), we have that eₖEₖ₊₁ ∩ eₖEₖ +₁ ≠ ∅ which leads to a contradiction. Similarly we can show that does not exist eₖd such that eₖEₖ₊₁ ∩ eₖEₖ +₁ = ∅ and eₖEₖ₊₁ ∩ eₖEₖ +₁ ≠ ∅;

cr19 In this case we have that DataPropertyAssertion (a₁ e l) is in Chase₅(O), SubDataPropertyOf (a₁ a₂) is in O, DataPropertyAssertion (a₂ e l) is not in Chase₅(O), and Chase₅₊₁(O) = Chase₅(O) ∪ {DataPropertyAssertion (a₂ e l)}. Let e₁ = DataSomeValuesFrom (a₁ rdfs:Literal) and e₁Inv(a₁) = DataSomeValuesFrom (a₂ rdfs:Literal). Then we have that e₁Eₖ₊₁ = e₁Eₖ ∪ {e} and for every eₖd ≠ e₂ we have that eₖEₖ₊₁ = eₖEₖ. Suppose there exists eₖd such that e₁Eₖ₊₁ ∩ eₖEₖ = ∅ and eₖEₖ₊₁ ∩ eₖEₖ +₁ ≠ ∅. Clearly this can happen if and only if e₁ ∈ eₖEₖ +₁. Then we know that e₁Eₖ ∩ eₖEₖ +₁ ≠ ∅ and, by inductive hypothesis, that e₁Eₖ ∩ e₀Eₖ +₀ ≠ ∅. Thus, by definition, we know that O |= ¬ DisjointClasses(e₁ eₖd). Therefore we have that O |= ¬ DisjointClasses(e₁ eₖd) and, by definition, we have that e₁Eₖ₊₁ ∩ eₖEₖ +₁ ≠ ∅. Since, by definition, when moving from Chase₅(O) to Chase₅₊₁(O) no axiom is
removed from $\text{Chase}^3(\mathcal{O})$, we have that $e^{E_k}_{a_2} \cap e^{E_k}_{c_1} \neq \emptyset$ which leads to a contradiction.

**cr20** In this case it is straightforward to see that for every $e_1$, $e_2$ we have that $e^{E_{k+1}}_1 \cap e^{E_{k+1}}_2 = \emptyset$ if and only if $e^{E_{k+1}}_2 \cap e^{E_{k+1}}_1 = \emptyset$.

**cr21** In this case we have that $\text{ReflexiveObjectProperty}(p)$ is in $\mathcal{O}$ and there exists $e \in V^O_1 \cup S^k_0$ such that $\text{ObjectPropertyAssertion}(p \ e)$ is not in $\text{Chase}^k(\mathcal{O})$. Then $\text{Chase}^{k+1}(\mathcal{O}) = \text{Chase}^k(\mathcal{O}) \cup \{ \text{ObjectPropertyAssertion}(p \ e) \}$. Let $e_p = \text{ObjectSomeValuesFrom}(p \ \text{owl:Thing})$, $e_{\text{Inv}(p)} = \text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(p) \ \text{owl:Thing})$. Then we have that $e_p^{E_{k+1}} = e_p \cup \{ e \}$, $e_{\text{Inv}(p)}^{E_{k+1}} = e_{\text{Inv}(p)} \cup \{ e \}$ and for every $e_d$ such that $e_d \neq e_p$ and $e_d \neq e_{\text{Inv}(p)}$ we have that $e_d^{E_{k+1}} = e_d^{E_{k}}$. Suppose there exists $e_d$ such that $e_p^{E_k} \cap e_d^{E_k} = \emptyset$ and $e_d^{E_{k+1}} \cap e_{\text{Inv}(p)}^{E_{k+1}} \neq \emptyset$.

Clearly this can happen if and only if $e \in e_{\text{Inv}(p)}^{E_k}$. Since $e_d^{E_k} \neq \emptyset$, we know that $\text{Can}^k(\mathcal{O}) \models \neg \text{DisjointClasses}(e_d \ e_d)$. Thus, by inductive hypothesis, we have that $\text{Can}^k(\mathcal{O}) \models \neg \text{DisjointClasses}(e_d \ e_d)$. However, since $\mathcal{O} \models \text{SubClassOf}(\text{owl:Thing} \ e_p)$ and $\mathcal{O} \models \text{SubClassOf}(e_p \ \text{owl:Thing})$, we know, by definition, that $e_p^{E_0} \cap e_d^{E_0} = \emptyset$ if and only if one of the following is verified:

- $\mathcal{O} \models \text{DisjointClasses}(e_d \ e_d)$
- $\mathcal{O} \models \text{DisjointClasses}(\text{owl:Thing} \ e_d \text{owl:Nothing})$

Since both the above cases contradict the hypothesis that $\mathcal{O}$ is satisfiable, we have that such $e_d$ cannot exist. Similarly we can show that does not exist $e_d$ such that $e_{\text{Inv}(p)}^{E_k} \cap e_d^{E_k} = \emptyset$ and $e_{\text{Inv}(p)}^{E_{k+1}} \cap e_d^{E_{k+1}} \neq \emptyset$.

The proof involving axioms of the form $\text{DisjointObjectProperties}$ and $\text{DisjointDataProperties}$ can be given similarly.

Now let $\alpha$ be of the form $\text{IrreflexiveObjectProperty}$. Depending on the rule that is applied to obtain $\text{Can}^{k+1}(\mathcal{O})$ starting from $\text{Can}^k(\mathcal{O})$ we have different possible scenarios. If $\text{Can}^{k+1}(\mathcal{O})$ is obtained by applying a rule identified by a number in the set $\{1, 4, 5, 8, 9, 12, 13, 16, 19, 20\}$, then it is straightforward to see that for every $e$ we have that $e^{R_{k+1}} = e^{R_k}$, then $\text{Can}^{k+1}(\mathcal{O}) \models \text{IrreflexiveObjectProperty}(e)$ if and only if $\text{Can}^k(\mathcal{O}) \models \text{IrreflexiveObjectProperty}(e)$.

**cr2** In this case we have that $\text{ClassAssertion}(c_1 \ e)$ is in $\text{Chase}^k(\mathcal{O})$, $\text{SubClassOf}(c_1 \ \text{ObjectSomeValuesFrom}(p \ c_2))$ is in $\mathcal{O}$, and does not exists $e_{in}$ such that both $\text{ObjectPropertyAssertion}(p \ e \ e_{in})$ and $\text{ClassAssertion}(c_2 \ e_{in})$ are in $\text{Chase}^k(\mathcal{O})$. Then $\text{Chase}^{k+1}(\mathcal{O}) = \text{Chase}^k(\mathcal{O}) \cup \{ \text{ClassAssertion}(c_2 \ s), \text{ObjectPropertyAssertion}(p \ e \ s) \}$ where $s \in S^k_0$ and $s \notin S^k_1$. Let $\text{Inv}(p) = \text{ObjectInverseOf}(p)$. Then we have that $p^{R_{k+1}} = p^{R_k} \cup \{ e \}$, $\text{Inv}(p)^{R_{k+1}} = p^{R_k} \cup \{ e \}$ and for every $e_{op}$ such that $e_{op} \neq p$ and $e_{op} \neq \text{Inv}(p)$ we have that $e_{op}^{R_{k+1}} = e_{op}^{R_k}$. Therefore we have that for every $e_{op}$ such that $e_{op} \neq p$ and $e_{op} \neq \text{Inv}(p)$, $\text{Can}^{k+1}(\mathcal{O}) \models \text{IrreflexiveObjectProperty}(e_{op})$ if and only if $\text{Can}^k(\mathcal{O}) \models \text{IrreflexiveObjectProperty}(e_{op})$. Furthermore, since by definition $s \neq e$, we have
that $\text{Can}^{k+1}(O) \models \text{IrreflexiveObjectProperty} (p)$ and $\text{Can}^{k+1}(O) \models \text{IrreflexiveObjectProperty} (\text{Inv}(p))$ if and only if $\text{Can}^{k}(O) \models \text{IrreflexiveObjectProperty}(\text{Inv}(p))$, and, by inductive hypothesis, if and only if $\text{Can}^{0}(O) \models \text{IrreflexiveObjectProperty} (p)$ and $\text{Can}^{0}(O) \models \text{IrreflexiveObjectProperty} (\text{Inv}(p))$. Similarly we can give the proof in case $\text{Can}^{k+1}(O)$ is obtained by applying a rule with a number in the set \{3, 6, 7, 10, 11, 14, 15\};

cr17 In this case we have that ObjectPropertyAssertion $(p_1 e_1 e_2)$ is in $\text{Chase}^k(O)$, SubObjectPropertyOf $(p_1 p_2)$ is in $O$, ObjectPropertyAssertion $(p_2 e_1 e_2)$ is not in $\text{Chase}^k(O)$, and $\text{Chase}^{k+1}(O) = \text{Chase}^k(O) \cup \{\text{ObjectPropertyAssertion} (p_2 e_1 e_2)\}$. Let $\text{Inv}(p_2) = \text{ObjectInverseOf} (p_2)$. Then we have that $p_2^{R_{k+1}} = p_2^{R_k} \cup (e_1 e_2), \text{Inv}(p_2)^{R_{k+1}} = \text{Inv}(p_2)^{R_k} \cup (e_2 e_1)$ and for every $e_\text{op}$ such that $e_\text{op} \neq p_2$ and $e_\text{op} \neq \text{Inv}(p_2)$ we have that $e^{R_{k+1}}_\text{op} = e^{R_k}_\text{op}$. Therefore we have that for every $e_\text{op}$ such that $e_\text{op} \neq p_2$ and $e_\text{op} \neq \text{Inv}(p_2)$, $\text{Can}^{k+1}(O) \models \text{IrreflexiveObjectProperty} (e_\text{op})$ if and only if $\text{Can}^{k}(O) \models \text{IrreflexiveObjectProperty} (e_\text{op})$. Clearly if $e_1 \neq e_2$ we have that $\text{Can}^{k+1}(O) \models \text{IrreflexiveObjectProperty} (p_2)$ and $\text{Can}^{k+1}(O) \models \text{IrreflexiveObjectProperty} (\text{Inv}(p_2))$ if and only if $\text{Can}^{k}(O) \models \text{IrreflexiveObjectProperty} (p_2)$ and $\text{Can}^{k}(O) \models \text{IrreflexiveObjectProperty} (\text{Inv}(p_2))$, and, by inductive hypothesis, if and only if $\text{Can}^{0}(O) \models \text{IrreflexiveObjectProperty} (p_2)$ and $\text{Can}^{0}(O) \models \text{IrreflexiveObjectProperty} (\text{Inv}(p_2))$. Now suppose that $e_1 = e_2$ and that does not exist $e \in \Delta_k^r$ such that $(e, e) \in p_2^{R_k}$. This is not possible as if $e_1 = e_2$, then $\text{Can}^{k}(O) \models \neg \text{IrreflexiveObjectProperty} (p_1)$, and, by inductive hypothesis, $\text{Can}^{0}(O) \models \neg \text{IrreflexiveObjectProperty} (p_1)$. But in this case we have that $O \models \neg \text{IrreflexiveObjectProperty} (p_2)$ and, by definition, there exists $\beta_{p_2}^{\text{iri}} \in S_0^r$ such that $(\beta_{p_2}^{\text{iri}}, \beta_{p_2}^{\text{iri}}) \in p_2^{R_0}$. Since, by definition, when moving from $\text{Chase}^j(O)$ to $\text{Chase}^{j+1}(O)$ no axiom is removed from $\text{Chase}^j(O)$, we have that $(\beta_{p_2}^{\text{iri}}, \beta_{p_2}^{\text{iri}}) \in p_2^{R_k}$, which leads to a contradiction. Similarly we can give the proof in case $\text{Can}^{k+1}(O)$ is obtained by applying rule cr18;

cr21 In this case we have that ReflexiveObjectProperty $(p)$ is in $O$ and there exists $e \in V^O_j \cup S^r_k$ such that ObjectPropertyAssertion$(p e e)$ is not in $\text{Chase}^k(O)$. Then $\text{Chase}^{k+1}(O) = \text{Chase}^k(O) \cup \{\text{ObjectPropertyAssertion} (p e e)\}$. Then we have that $p^{R_{k+1}} = p^{R_k} \cup (e e)$, and for every $e_\text{op} \neq p$ we have that $e^{R_{k+1}}_\text{op} = e^{R_k}_\text{op}$. Therefore we have that for every $e_\text{op} \neq p$, $\text{Can}^{k+1}(O) \models \text{IrreflexiveObjectProperty} (e_\text{op})$ if and only if $\text{Can}^{k}(O) \models \text{IrreflexiveObjectProperty} (e_\text{op})$. Moreover, since $O \models \neg \text{IrreflexiveObjectProperty} (p)$, we know that there exists $\beta_p^{\text{iri}} \in S_0^r$ such that $(\beta_p^{\text{iri}}, \beta_p^{\text{iri}}) \in p_0^{R_0}$. Since, by definition, when moving from $\text{Chase}^j(O)$ to $\text{Chase}^{j+1}(O)$ no axiom is removed from $\text{Chase}^j(O)$, we have that $(\beta_p^{\text{iri}}, \beta_p^{\text{iri}}) \in p_2^{R_k}$, and therefore $\text{Can}^{k}(O) \models \neg \text{IrreflexiveObjectProperty} (e)$.

$\Box$

By the above proposition, we know that, if $O$ is TBox-complete, to check whether or not a negative axiom in $\Delta^O$ is satisfied by $\text{Can}(O)$, it is sufficient to check whether or not it is satisfied by $\text{Can}^{0}(O)$. Let $\mathcal{P}^O$ denote the set containing all the positive TBox axioms
that is possible to build starting from suitable expressions in $Exp^O$. Let $\alpha$ be an axiom in $P^O$. The following Lemma tell us that, if $\alpha$ is satisfied by $Can(O)$, then $\alpha$ is implied by $O$.

**Lemma 53** Let $O$ be an ontology. For every $k \geq 0$ we have that following statements hold:

1. Let $e_1$ and $e_2$ be two expressions in $Exp^O$. If $Can^k(O) \models \text{SubClassOf}(e_1 e_2)$, then $O \models \text{SubClassOf}(e_1 e_2)$;

2. Let $e_1$ and $e_2$ be two expressions in $Exp^O_{OP}$. If $Can^k(O) \models \text{SubObjectPropertyOf}(e_1 e_2)$, then $O \models \text{SubObjectPropertyOf}(e_1 e_2)$;

3. Let $e_1$ and $e_2$ be two expressions in $Exp^O_{DP}$. If $Can^k(O) \models \text{SubDataPropertyOf}(e_1 e_2)$, then $O \models \text{SubDataPropertyOf}(e_1 e_2)$;

4. Let $e_p$ be an expression in $Exp^O_{OP}$. If $Can^k(O) \models \text{ReflexiveObjectProperty}(e_p)$, then $O \models \text{ReflexiveObjectProperty}(e_p)$.

**Proof.** We prove the Lemma for statement [1]. The proof for statements [2] and [3] can be given similarly.

As for $k = 0$ the proof is trivial since $Can^0(O) \models \text{SubClassOf}(e_1 e_2)$ if and only if $e_{E_1}^0 = \emptyset$. Indeed, if $e_{E_1}^0 \neq \emptyset$, by definition, we know that there exists a $e_{\alpha_{e_1} e_1}$ in $S^0$ such that $\alpha_{e_1} e_1 \in e_{E_1}^0$ and $e_{\alpha_{e_1} e_1} \notin e_{E_1}^0$, and we would find a contradiction. As, by definition, $e_{E_1}^0 = \emptyset$ if and only if $O \models \text{DisjointClasses}(e_1 e_1)$, then $O \models \text{SubClassOf}(e_1 e_2)$ for every $e_2 \in Exp^O$.

Now suppose $Can^k(O) \models \text{SubClassOf}(e_1 e_2)$ for $k > 0$. We distinguish between two cases: (a) $e_{E_1}^{E_k} = \emptyset$, and (b) $e_{E_1}^{E_k} \neq \emptyset$:

(a) In this case we have that $Can^k(O) \models \text{DisjointClasses}(e_1 e_1)$, thus by Proposition [52], we have also that $Can^0(O) \models \text{DisjointClasses}(e_1 e_1)$. Therefore, by definition, we know that $O \models \text{DisjointClasses}(e_1 e_1)$, then $O \models \text{SubClassOf}(e_1 e_2)$ for every $e_2 \in Exp^O$;

(b) In this case we know that $Can^k(O) \models \neg \text{DisjointClasses}(e_1 e_1)$, thus by Proposition [52], we have also that $Can^0(O) \models \neg \text{DisjointClasses}(e_1 e_1)$. Then, by definition, there exists $e_{\alpha_{e_1} e_1}$ such that $\text{ClassAssertion}(e_1 e_{\alpha_{e_1} e_1})$ is in $\text{Chase}^0(O)$, and such that $\alpha_{e_1} e_1$ does not occur in any other axiom in $\text{Chase}^0(O)$. Therefore, by definition, we have that $\alpha_{e_1} e_1 \in e_{E_2}^k$ only if either $\text{SubClassOf}(e_1 e_2)$ is in $O$, or there exists a chain of length $m < k$ of the form $\text{SubClassOf}(e_1 e_{ch 1} 0), SubClassOf(e_{ch 1} e_{ch 2}), \ldots, SubClassOf(e_{ch m-1} e_2)$ such that every axiom occurring in the chain is contained in $O$. In both cases we have that $O \models \text{SubClassOf}(e_1 e_2)$.

Consider now statement [3] of the Lemma. As for $k = 0$ the proof is trivial since there is no $e_p$ in $\Delta^0$ such that $(e_{in}, e_{in}) \in e_{R_0}^p$ for every $e_{in}$ in $\Delta^0 \text{Can}(O)$. Indeed if $e_{R_0}^p \neq \emptyset$, then we know that $O \models \neg \text{DisjointObjectProperties}(e_p e_p)$, and therefore, by definition, there exist $\beta_{e_p, e_p}$ and $\beta_{e_p, e_p}$ in $\Delta^0$ such that $(\beta_{e_p, e_p}, \beta_{e_p, e_p}) \in e_{R_0}^p$, $(\beta_{e_p, e_p}, \beta_{e_p, e_p}) \notin e_{R_0}^p$, and $(\beta_{e_p, e_p}, \beta_{e_p, e_p}) \notin e_{R_0}^p$.

Now suppose $Can^k(O) \models \text{ReflexiveObjectProperty}(e_p)$ for $k > 0$. Then, by definition,
we have that $e^R_p \neq \emptyset$, and therefore we know that $Can^k(O) \models \neg \text{DisjointObjectProperties}(e_p,e_p)$. Thus, by Proposition 52 we also know that $Can^0(O) \models \neg \text{DisjointObjectProperties}(e_p,e_p)$. Therefore, by definition, there exist $\beta^1_{p,e_p}, \beta^2_{p,e_p}$ in $\Delta^0$ such that $(\beta^1_{e_p,p}, \beta^2_{e_p,p}) \in e^R_{p}$, $(\beta^2_{p,e_p}, \beta^1_{p,e_p}) \not\in e^p_{p}$, and $(\beta^2_{p,e_p}, \beta^2_{p,e_p}) \not\in e^R_{p}$. By looking at chase rules in Table 7.1, one can verify that if $(\beta^1_{p,e_p}, \beta^1_{p,e_p}) \in e^R_{p}$ (respectively, $(\beta^2_{p,e_p}, \beta^1_{p,e_p}) \in e^R_{p}$), then there exists $0 < h \leq k$ such that $(\beta^1_{e_p,e_p}) \not\in e^R_{h-1}$, $(\beta^2_{e_p,e_p}) \in e^R_{h}$ (respectively, $(\beta^2_{p,e_p}, \beta^2_{p,e_p}) \not\in e^R_{h-1}$, $(\beta^2_{p,e_p}, \beta^2_{p,e_p}) \in e^R_{h}$), and $Can^h(O)$ is obtained from $Can^{h-1}(O)$ by applying rule cr21 taking into account ReflexiveObjectProperty ($e_p$) and $\beta^1_{p,e_p}$ (respectively, $\beta^2_{p,e_p}$). Therefore we have that $O \models \text{ReflexiveObjectProperty}(e_p)$.\hfill\Box

We now consider the problem of answering arbitrary higher-order queries over a TBox-complete ontology $O$. To this aim, we define the notion of extended homomorphism from the canonical pseudo interpretation $Can(O)$ to an interpretation $H$ for $O$.

**Definition 54** Let $Can(O) = (\Sigma_{Can(O)}, I_{Can(O)})$, where $\Sigma_{Can(O)} = (\Delta^0_{Can(O)}, \Delta^1_{Can(O)}, \Delta^2_{Can(O)}, \Delta^3_{Can(O)}, \Delta^4_{Can(O)}, \Delta^5_{Can(O)}, \Delta^6_{Can(O)}, \Delta^7_{Can(O)})$ be the canonical pseudo interpretation for $O$, and let $H = (\Sigma^H, I^H)$ be an interpretation for $O$, where $\Sigma^H = (\Delta^0_{H}, \Delta^1_{H}, \Delta^2_{H}, \Delta^3_{H}, \Delta^4_{H}, \Delta^5_{H}, \Delta^6_{H}, \Delta^7_{H})$.

Let $\Psi$ be a function from $\Delta^0_{Can(O)}$ to $\Delta^0_{H}$ and from $\Delta^1_{Can(O)}$ to $\Delta^1_{H}$. We say that $\Psi$ is an extended homomorphism from $Can(O)$ to $H$ if:

- $\Psi$ is an instance-based homomorphism from $Can(O)$ to $H$, and
- $\Psi$ further preserves in $\Sigma^H$ the following semantic properties of $\Sigma_{Can(O)}$:
  
  7. for every $e_1, e_2 \in \Delta^0_{Can(O)}$: if $e_1 \in \Delta^0_{Can(O)}$, then $\Psi(e_1)E^H \subseteq \Psi(e_2)E^H$;
  
  8. for every $e_1, e_2 \in \Delta^0_{Can(O)}$: if $e_1 \in \Delta^1_{Can(O)}$, then $\Psi(e_1)E^H \subseteq \Psi(e_2)E^H$;
  
  9. for every $e_1, e_2 \in \Delta^0_{Can(O)}$: if $e_1 \in \Delta^2_{Can(O)}$, then $\Psi(e_1)E^H \subseteq \Psi(e_2)E^H$;
  
  10. for every $e_1, e_2 \in \Delta^0_{Can(O)}$: if $e_1 \in \Delta^3_{Can(O)}$, then $\Psi(e_1)E^H \subseteq \Psi(e_2)E^H$;
  
  11. for every $e_1, e_2 \in \Delta^0_{Can(O)}$: if $e_1 \in \Delta^4_{Can(O)} \cap e_2 \in \Delta^5_{Can(O)} = \emptyset$, then $\Psi(e_1)E^H \cap \Psi(e_2)E^H = \emptyset$;
  
  12. for every $e_1, e_2 \in \Delta^0_{Can(O)}$: if $e_1 \in \Delta^6_{Can(O)} \cap e_2 \in \Delta^7_{Can(O)} = \emptyset$, then $\Psi(e_1)E^H \cap \Psi(e_2)E^H = \emptyset$;
  
  13. for every $e_1, e_2 \in \Delta^0_{Can(O)}$: if $e_1 \in \Delta^8_{Can(O)} \cap e_2 \in \Delta^9_{Can(O)} = \emptyset$, then $\Psi(e_1)A^H \cap \Psi(e_2)A^H = \emptyset$;
  
  14. for every $e_1 \in \Delta^0_{Can(O)}$, if for every $e_2 \in \Delta^1_{Can(O)}$, $e_2 \in e \in \Delta^2_{Can(O)}$, then for every $e_3 \in \Delta^3_{H}, (e_3, e_3) \in \Psi(e_1)R^H$;
  
  15. for every $e_1 \in \Delta^0_{Can(O)}$, if for every $e_2 \in \Delta^1_{Can(O)}$, $(e_2, e_2) \not\in e \in \Delta^2_{Can(O)}$, then for every $e_3 \in \Delta^3_{H}, (e_3, e_3) \not\in \Psi(e_1)R^H$.

The following proposition shows that, besides representing all models of $O$ with respect to instance checking, $Can(O)$ also represents all the semantic properties of all models of $O$. 
Proposition 55 If $\mathcal{O}$ is a satisfiable TBox-complete ontology, then for every model $M$ of $\mathcal{O}$ there exists an extended homomorphism from $\text{Can}(\mathcal{O})$ to $M$.

Proof. Let $M$ be a model of $\mathcal{O}$, and let $\text{Can}^k(\mathcal{O}) = (\Sigma_k, \mathcal{I}_k)$ be the portion of $\text{Can}(\mathcal{O})$ that is defined from $\text{Chase}^k(\mathcal{O})$, where $\Sigma_k = (\Delta^k_0, \Delta^k, \mathcal{I}_k, \mathcal{E}_k, \mathcal{R}_k, \mathcal{A}_k, \mathcal{I}_k)$. We define the function $\Psi$ as specified in the proof of Proposition\[36\] and we show by induction that, for every $k$, it is possible to derive $\Psi$ is an extended homomorphism from $\text{Can}^k(\mathcal{O})$ to $M$.

In Proposition\[36\] we showed that $\Psi$ is an instance-based homomorphism from $\text{Can}(\mathcal{O})$, thus we only need to show that $\Psi$ additionally satisfies properties (7)-(15).

Base step. Let us first consider property (7). Let $e_1, e_2 \in \Delta^0_0$ be such that $e_1^{E_0} \subseteq e_2^{E_0}$. We first show that this can happen only if $e_1^{E_0} = \emptyset$. Indeed, if $e_1^{E_0} \neq \emptyset$, by definition, we know that there exists a $\alpha_{e_1, e_1}$ in $S^0_0$ such that $\alpha_{e_1, e_1} \in e_1^{E_0}$ and $\alpha_{e_1, e_1} \notin e_2^{E_0}$, and we would find a contradiction. So we know that in this case $e_1^{E_0} = \emptyset$. Then, by definition, we know that $\mathcal{O} \models \text{DisjointClasses}(e_1 e_1)$ and, being $M$ a model of $\mathcal{O}$, we have that $M \models \text{DisjointClasses}(e_1 e_1)$. Therefore we have $\Psi(e_1)^{E_M} = \emptyset$ and that $\Psi$ trivially preserves in $M$ property (7) for $\text{Can}^0(\mathcal{O})$. Analogously, we can prove that $\Psi$ preserves in $M$ properties (9), and (10) for $\text{Can}^0(\mathcal{O})$.

Let us now consider property (8). Let $e_1, e_2 \in \Delta^0_0$ be such that $e_2 \in D_M$ and $\{v|(e, v) \in e_1^{A^T}\} \subseteq e_2^{A^T}$. By definition this can happen if and only if for every axiom of the form Data-PropertyAssertion$(e_1 e_1)$ that is contained in $\mathcal{O}$ we have that $h^{LS} \in e_2^{DT}$, where $^{LS}$ and $^{DT}$ are the interpretation functions defined by the OWL 2 QL datatype map for literals and datatypes, respectively. Since every model of $\mathcal{O}$ must agree with the interpretation of literals and datatypes fixed once and for all by the OWL 2 QL datatype map and that, we have that $\Psi(e_2)^{T_M} = e_2^{DT}$, thus we obtain that $\{v'|((e, v') \in \Psi(e_1)^{A_M}) \subseteq \Psi(e_2)^{T_M}$, Thus we have shown that $\Psi$ preserves in $M$ property (8) for $\text{Can}^0(\mathcal{O})$.

Let us now consider property (14). We next show that $\Psi$ trivially preserves in $M$ property (14) for $\text{Can}^0(\mathcal{O})$, as there is no $e_1$ in $\Delta^0_0$ such that $e_2, e_2 \in e_1^{R_0}$ for every $e_2$ in $\Delta^0_0$. Indeed if $e_1^{R_0} \neq \emptyset$, then we know that $\mathcal{O} \models \neg\text{DisjointObjectProperties}(e_1 e_1)$, and therefore, by definition, there exist $\beta_{e_1, e_1}$ in $S^0_0$ such that $\beta_{e_1, e_1} \notin e_1^{R_0}$ and $\beta_{e_1, e_1} \notin e_1^{R_0}$.

Let us now consider property (11). Let $e_1, e_2 \in \Delta^0_0$ be such that $e_1^{E_0} \cap e_2^{E_0} = \emptyset$. We distinguish between two cases: (a) $e_1^{E_0} = \emptyset$, and (b) $e_1^{E_0} \neq \emptyset$:

(a) By construction of $\text{Can}^0(\mathcal{O})$, we know that $\mathcal{O} \models \text{DisjointClasses}(e_1 e_1)$. Being $M$ a model of $\mathcal{O}$, we have that $M \models \text{DisjointClasses}(e_1 e_1)$, thus $\Psi(e_1)^{E_M} = \emptyset$ and, trivially, $\Psi(e_1)^{E_M} \cap \Psi(e_2)^{E_M} = \emptyset$.

(b) By construction of $\text{Can}^0(\mathcal{O})$, we know that $\mathcal{O} \not\models \neg\text{DisjointClasses}(e_1 e_2)$, as otherwise, it would exist a variable $\alpha_{e_1, e_2}$ in $S^0_0$ such that $\alpha_{e_1, e_2} \in e_1^{E_0}$ and $\alpha_{e_1, e_2} \in e_2^{E_0}$. Since $\mathcal{O}$ is TBox-complete, we know that either $\mathcal{O} \models \text{DisjointClasses}(e_1 e_2)$ or $\mathcal{O} \models \neg\text{DisjointClasses}(e_1 e_2)$. Since $\mathcal{O} \not\models \neg\text{DisjointClasses}(e_1 e_2)$, we have that $\mathcal{O} \models \text{DisjointClasses}(e_1 e_2)$ and, being $M$ a model of $\mathcal{O}$, we have that $M \models \text{DisjointClasses}(e_1 e_2)$, and therefore $\Psi(e_1)^{E_M} \cap \Psi(e_2)^{E_M} = \emptyset$.

In both the two above cases $\Psi(e_1)^{E_M} \cap \Psi(e_2)^{E_M} = \emptyset$, thus $\Psi$ preserves in $M$ property (11) for $\text{Can}^0(\mathcal{O})$. Analogously, we can prove that $\Psi$ preserves in $M$ properties (12) and (13) for $\text{Can}^0(\mathcal{O})$. 8. Answering general queries
Finally we consider property (15). Let \( e_1 \in \Delta_o^0 \) be such that for every \( e_2 \in \Delta_o^0 \) we have that \( (e_2, e_2) \notin \Theta^{R_0} \). We distinguish between two cases: (a) \( e_1^{E_0} = \emptyset \), and (b) \( e_1^{E_0} \neq \emptyset \):

(a) By construction of \( \text{Can}^0(\mathcal{O}) \), we know that \( \mathcal{O} \models \text{DisjointObjectProperties} (e_1, e_2) \). Being \( M \) a model of \( \mathcal{O} \), we have that \( M \models \text{DisjointObjectProperties} (e_1, e_2) \), thus \( \Psi(e_1)^{EM} = \emptyset \) and trivially we have that, for every \( e_3 \in \Delta_o^M \), \( (e_3, e_3) \notin \Psi(e_1)^{EM} \).

(b) By construction of \( \text{Can}^0(\mathcal{O}) \), we know that \( \mathcal{O} \not\models \neg \text{IrreflexiveObjectProperty} (e_1) \), as otherwise, it would exist a variable \( \beta_{e_1}^{\text{irr}} \) in \( \mathcal{S}_o^0 \) such that \( (\beta_{e_1}^{\text{irr}}, \beta_{e_1}^{\text{irr}}) \in e_1^{R_0} \).

Since \( \mathcal{O} \) is TBox-complete, we know that either \( \mathcal{O} \models \text{IrreflexiveObjectProperty} (e_1) \) or \( \mathcal{O} \models \neg \text{IrreflexiveObjectProperty} (e_1) \). Since \( \mathcal{O} \not\models \neg \text{IrreflexiveObjectProperty} (e_1) \), and, being \( M \) a model of \( \mathcal{O} \), we have that \( M \models \text{IrreflexiveObjectProperty} (e_1) \), and therefore we have that, for every \( e_3 \in \Delta_o^M \), \( (e_3, e_3) \notin \Psi(e_1)^{EM} \).

In both the two above cases for every \( e_3 \in \Delta_o^M \), we have that \( (e_3, e_3) \notin \Psi(e_1)^{EM} \), thus \( \Psi \) preserves in \( M \) property (15) for \( \text{Can}^0(\mathcal{O}) \).

Then we proved that \( \Psi \) is an extended homomorphism from \( \text{Can}^0(\mathcal{O}) \) to \( M \).

**Inductive step** Suppose that \( \Psi \) is an extended homomorphism from \( \text{Can}^k(\mathcal{O}) \) to \( M \), we next show that \( \Psi \) is an extended homomorphism from \( \text{Can}^{k+1}(\mathcal{O}) \) to \( M \) as well. In the following we say that two sets \( A \) and \( B \) are incomparable if neither \( A \subseteq B \) nor \( B \subseteq A \).

To prove that \( \Psi \) preserves in \( M \) properties (11-13) and (15) we simply rely on Proposition[32] Then it remains to show that \( \Psi \) also preserves in \( M \) properties (7-10) and (14)

Depending on the rule that is applied to obtain \( \text{Can}^{k+1}(\mathcal{O}) \) starting from \( \text{Can}^k(\mathcal{O}) \) we have different possible scenarios.

**cr1** In this case we have that \( \text{ClassAssertion} (c_1 e) \) is in \( \text{Chase}^k(\mathcal{O}) \), \( \text{SubClassOf} (c_1 c_2) \) is in \( \mathcal{O} \), \( \text{ClassAssertion} (c_2 e) \) is not in \( \text{Chase}^k(\mathcal{O}) \), and \( \text{Chase}^{k+1}(\mathcal{O}) = \text{Chase}^k(\mathcal{O}) \cup \{ \text{ClassAssertion} (c_2 e) \} \). As for every \( e_p \) we have that \( e_{p+1}^{R_{k+1}^{E_k}} = e_{p+1}^{R_k} \), \( e_{p+1}^{A_{k+1}} = e_{p+1}^{A_k} \), and \( e_{p+1}^{T_{k+1}} = e_{p+1}^{T_k} \), by inductive hypothesis we know that \( \Psi \) preserves in \( M \) properties (8-10) and (14) for \( \text{Can}^{k+1}(\mathcal{O}) \).

Consider now property (7). We have that \( c_2^{E_k} = c_2 \cup \{ e \} \), and for every \( e_{cd} \neq e_2 \) we have that \( e_{cd}^{E_{k+1}^{E_k}} = e_{cd}^{E_k} \). Moreover only one of the following can be verified: (1) \( e_{cd}^{E_k} \subseteq e_2^{E_k} \), (2) \( e_2^{E_k} \subseteq e_{cd}^{E_k} \) and \( e \in e_{cd}^{E_k} \), (3) \( c_2^{E_k} \subseteq c_2^{E_k} \) and \( e \notin e_{cd}^{E_k} \), (4) \( e_{cd}^{E_k} \) and \( c_2^{E_k} \) are incomparable, and \( e \notin e_{cd}^{E_k} \), (5) \( e_{cd}^{E_k} \) and \( c_2^{E_k} \) are incomparable, \( e \in e_{cd}^{E_k} \), and there exists an object \( e' \neq e \) in \( \Delta_o^k \) such that \( e' \notin e_{cd}^{E_k} \) and \( e' \in c_2^{E_k} \), and (6) \( e_{cd}^{E_k} \) and \( c_2^{E_k} \) are incomparable, \( e \in e_{cd}^{E_k} \), and does not exist any object \( e' \neq e \in \Delta_o^k \) such that \( e' \notin e_{cd}^{E_k} \) and \( e' \in c_2^{E_k} \).

(1) In this case we have that \( e_{cd}^{E_{k+1}^{E_k}} \subseteq e_2^{E_{k+1}^{E_k}} \) and, by inductive hypothesis, we know that \( \Psi(e_{cd})^{EM} \subseteq \Psi(e_2)^{EM} \).

(2) In this case we have that \( c_2^{E_{k+1}^{E_k}} \subseteq e_{cd}^{E_{k+1}^{E_k}} \) and, by inductive hypothesis, we know that \( \Psi(e_2)^{EM} \subseteq \Psi(e_{cd})^{EM} \).

(3)-(5) In these cases we have that \( c_2^{E_{k+1}^{E_k}} \) and \( e_{cd}^{E_{k+1}^{E_k}} \) are incomparable.
8. Answering general queries

In this case we have that \( e_{cd}^{E_{k+1}} \subseteq e_{c2}^{E_k} \). Thus, by Lemma 53 we know that \( \mathcal{O} \models \text{SubClassOf} (e_{cd} c_2) \) and, being \( M \) a model of \( \mathcal{O} \), we have that \( M \models \text{SubClassOf} (e_{cd} c_2) \), and therefore \( \Psi(e_{cd})^E_M \subseteq \Psi(c_2)^E_{k+1} \).

Then, when rule cr1 is applied, we have shown that \( \Psi \) is an extended homomorphism from \( \text{Can}^{k+1}(\mathcal{O}) \) to \( M \).

**cr2** In this case we have that \( \text{ClassAssertion} (c_1 e) \) is in \( \text{Chase}^k(\mathcal{O}) \), \( \text{SubClassOf} (c_1 \text{ObjectSomeValuesFrom} (p c_2)) \) is in \( \mathcal{O} \), and does not exists \( e_{in} \) such that both \( \text{ObjectPropertyAssertion} (p e_{in}) \) and \( \text{ClassAssertion} (c_2 e_{in}) \) are in \( \text{Chase}^k(\mathcal{O}) \). Then \( \text{Chase}^{k+1}(\mathcal{O}) = \text{Chase}^k(\mathcal{O}) \cup \{ \text{ClassAssertion} (c_2 s), \text{ObjectPropertyAssertion} (p e s) \} \) where \( s \in \mathcal{O}^k \) and \( s \notin \mathcal{O}^k \). As for every \( e_s \) we have that \( e_{Ak+1} = e_{Ak}^T \) and \( e_{Tk+1} = e_{Tk}^T \), by inductive hypothesis we know that \( \Psi \) preserves in \( M \) properties (8) and (10) for \( \text{Can}^{k+1}(\mathcal{O}) \). Consider now property (7) and let \( e_{p,c_2} = \text{ObjectSomeValuesFrom} (p c_2) \) and \( e_{inv(p)} = \text{ObjectSomeValuesFrom} (\text{ObjectInverseOf} (p \text{owl:Thing})) \). Then we have that \( e_{p,c_2} = e_{p,c_2} \cup \{ e \} \), \( e_{E_{k+1}} = e_{p,c_2} \cup \{ s \} \), \( e_{inv(p)} = e_{p,c_2} \cup \{ s \} \), and for every \( e_{cd} \) such that \( e_{cd} \neq p c_2 \) and \( e_{cd} \neq c_2 \), and \( e_{cd} \neq e_{inv(p)} \) we have that \( e_{E_{k+1}} = e_{cd}^E \). Then we distinguish between two cases: (a) \( e_{cd} \) is such that \( e_{cd} \neq c_2 \) and \( e_{cd} \neq e_{inv(p)} \), and (b) \( e_{cd} \neq p c_2 \).

(a) In this case we have three possible scenarios: (1) \( e_{E_{cd}}^{E_k} \subseteq e_{c2}^{E_k} \), (2) \( e_{E_{cd}}^{E_k} \subset e_{c2}^{E_k} \), and (3) \( e_{E_{cd}}^{E_k} \) and \( e_{c2}^{E_k} \) are incomparable:

1. \( e_{E_{cd}}^{E_k+1} \subseteq e_{c2}^{E_k+1} \) and, by inductive hypothesis, we know that \( \Psi(e_{cd})^E_M \subseteq \Psi(c_2)^E_{k+1} \);

2. Since \( s \notin e_{E_{cd}}^{E_k+1} \), we have that \( e_{cd}^{E_{k+1}} \) and \( e_{cd}^{E_k} \) are incomparable;

3. \( e_{E_{cd}}^{E_k+1} \) and \( e_{c2}^{E_k+1} \) are incomparable.

Similar considerations can be given if one considers \( e_{inv(p)} \) instead of \( c_2 \).

(b) In this case, we have six possible scenarios: (1) \( e_{E_{cd}}^{E_k} \subseteq e_{p,c_2}^{E_k} \), (2) \( e_{E_{cd}}^{E_k} \subset e_{p,c_2}^{E_k} \), and \( e \notin e_{p,c_2}^{E_k} \), (4) \( e_{E_{cd}}^{E_k} \) and \( e_{p,c_2}^{E_k} \) are incomparable, and \( e \notin e_{p,c_2}^{E_k} \), (5) \( e_{E_{cd}}^{E_k} \) and \( e_{p,c_2}^{E_k} \) are incomparable, \( e \in e_{E_{cd}}^{E_k} \), and there exists an object \( e' \neq e \) in \( \Delta_o^k \) such that \( e' \in e_{E_{cd}}^{E_k} \) and \( e' \notin e_{p,c_2}^{E_k} \), and (6) \( e_{E_{cd}}^{E_k} \) and \( e_{p,c_2}^{E_k} \) are incomparable, \( e \in e_{E_{cd}}^{E_k} \), and does not exists any object \( e' \neq e \) in \( \Delta_o^k \) such that \( e' \in e_{p,c}^{E_k} \) and \( e' \notin e_{E_{cd}}^{E_k} \):

1. In this case we have that \( e_{E_{cd}}^{E_k+1} \subseteq e_{p,c_2}^{E_k+1} \) and, by inductive hypothesis, we know that \( \Psi(e_{cd})^E_M \subseteq \Psi(e_{p,c_2})^E_{k+1} \);

2. In this case we have that \( e_{E_{cd}}^{E_k+1} \subseteq e_{p,c_2}^{E_k+1} \) and, by inductive hypothesis, we know that \( \Psi(e_{cd})^E_M \subseteq \Psi(e_{p,c_2})^E_{k+1} \);

(3)-(5) In these cases we have that \( e_{p,c_2}^{E_k+1} \) and \( e_{cd}^{E_k+1} \) are incomparable.

6. In this case we have that \( e_{cd}^{E_k+1} \subseteq e_{p,c_2}^{E_k+1} \). Thus, by Lemma 53, we know that \( \mathcal{O} \models \text{SubClassOf} (e_{cd} e_{p,c_2}) \) and, being \( M \) a model of \( \mathcal{O} \), we have that \( M \models \text{SubClassOf} (e_{cd} e_{p,c_2}) \), and therefore \( \Psi(e_{cd})^E_M \subseteq \Psi(e_{p,c_2})^E_{k+1} \).
Then, when rule cr2 is applied, we have shown that $\Psi$ preserves in $M$ property (7) for $Can^{k+1}(O)$.

Consider now property (9) and let $Inv(p) = ObjectInverseOf(p)$. Then we have that $p^{R_{k+1}} = p^R_k \cup \{(e, s)\}$, $Inv(p)^{R_{k+1}} = Inv(p)^R_k \cup \{(s, e)\}$, and for every $e_{op}$ such that $e_{op} \neq p$ and $e_{op} \neq Inv(p)$, we have that $e_{op}^{R_{k+1}} = e_{op}^{R_k}$. Let $e_{op} \neq p$, then we have three possible scenarios: (1) $e_{op}^R \subseteq p^R_k$, (2) $p^R_k \subseteq e_{op}^R$, and (3) $e_{op}^R$ and $p^R_k$ are incomparable:

1. $e_{op}^{R_{k+1}} \subseteq p^{R_{k+1}}$ and, by inductive hypothesis, we know that $\Psi(e_{op})^{R_M} \subseteq \Psi(p)^{R_M}$;
2. Since $(e, s) \notin e_{op}^{R_{k+1}}$, we have that $e_{op}^{R_{k+1}}$ and $p^{R_{k+1}}$ are incomparable;
3. $e_{op}^{R_{k+1}}$ and $p^{R_{k+1}}$ are incomparable.

Similar proof can be given if one considers $Inv(p)$ instead of $p$. Then, when rule cr2 is applied, we have shown that $\Psi$ preserves in $M$ property (9) for $Can^{k+1}(O)$.

Consider now property (14). Since we know that $\Delta^{k+1} = \Delta^k \cup \{s\}$ and that does not exists $e_{op} \in \Delta^{k+1}_o$ such that $(s, s) \in e_{op}^{R_{k+1}}$, we have that $\Psi$ trivially preserves in $M$ property (14) for $Can^{k+1}(O)$. Thus when rule cr2 is applied, we have shown that, $\Psi$ is an extended homomorphism from $Can^{k+1}(O)$ to $M$.

### cr4

In this case we have that ClassAssertion ($c \ e$) is in $Chase^k(O)$, SubClassOf ($c \ a$) $\cup$ DatatypeAssertion ($p \ v$) and DataSomeValuesFrom ($a \ d$) is in $O$, and does not exists $v$ such that both DataPropertyAssertion ($p \ e \ v$) and DatatypeAssertion ($d \ v$) are in $Chase^k(O)$. Then $Chase^{k+1}(O) = Chase^k(O) \cup \{DataPropertyAssertion (a \ e \ s), DatatypeAssertion (d \ s)\}$ where $s \in S^{k+1}_o$ and $s \notin S^k_o$. As for every $e_p$ we have that $e_p^{R_{k+1}} = e_p^{R_k}$ we know, by inductive hypothesis, that $\Psi$ preserves in $M$ properties (9) and (14) for $Can^{k+1}(O)$. Consider now property (7) and let $e_a = DataSomeValuesFrom (a \ d)$. Then we have that $e_a^{E_{k+1}} = e_a^E \cup \{e\}$, and for every $e_{cl} \neq e_a$ we have that $e_{cl}^{E_{k+1}} = e_{cl}^E$. Moreover only one of the following can be verified: (1) $e_{cl}^E \subseteq e_a^E$, (2) $e_a^E \subseteq e_{cl}^E$ and $e \in e_{cl}^E$, (3) $e_{cl}^E \subseteq e_a^E$ and $e \notin e_{cl}^E$, (4) $e_{cl}^E$ and $e_a^E$ are incomparable, and $e \notin e_{cl}^E$, (5) $e_{cl}^E$ and $e_a^E$ are incomparable, $e \in e_{cl}^E$, and there exists an object $e' \neq e$ in $\Delta^k_o$ such that $e' \notin e_{cl}^E$ and $e' \in e_a^E$, and (6) $e_{cl}^E$ and $e_a^E$ are incomparable, $e \in e_{cl}^E$, and does not exists any object $e' \neq e$ in $\Delta^k_o$ such that $e' \notin e_{cl}^E$ and $e' \in e_a^E$.

1. In this case we have that $e_{cl}^{E_{k+1}} \subseteq e_a^{E_{k+1}}$ and, by inductive hypothesis, we know that $\Psi(e_{cl})^{E_M} \subseteq \Psi(e_a)^{E_M}$.
2. In this case we have that $e_a^{E_{k+1}} \subseteq e_{cl}^{E_{k+1}}$ and, by inductive hypothesis, we know that $\Psi(e_a)^{E_M} \subseteq \Psi(e_{cl})^{E_M}$.
3)-(5) In these cases we have that $e_a^{E_{k+1}} + e_{cl}^{E_{k+1}}$ are incomparable.

6. In this case we have that $e_{cl}^{E_{k+1}} \subseteq e_a^{E_{k+1}}$. Thus, by Lemma 53, we know that $O \models \text{SubClassOf} (e_{cl} \ e_a)$ and, being $M$ a model of $O$, we have that $M \models \text{SubClassOf} (e_{cl} \ e_a)$, and therefore $\Psi(e_{cl})^{E_M} \subseteq \Psi(e_a)^{E_{k+1}}$. 

---

8.1 Answering general queries over TBox-complete ontologies 119
Then, when rule \texttt{cr4} is applied, we have shown that \( \Psi \) preserves in \( M \) property (7) for \( Can^{k+1}(O) \).

Consider now property (10). We have that \( a^{A_{k+1}} = a^{A_k} \cup \{(e, s)\} \), and for every \( e_{dp} \neq a \), we have that \( e_{dp}^{A_{k+1}} = e_{dp}^{A_k} \). Let \( e_{dp} \neq a \), then we have three possible scenarios: (1) \( e_{dp}^{A_k} \subseteq e_{dp}^{A_{k+1}} \) and, by inductive hypothesis, we know that \( \Psi(e_{dp})^{A_M} \subseteq \Psi(a)^{A_M} \); (2) Since \( (e, s) \notin e_{dp}^{A_{k+1}} \), we have that \( e_{dp}^{A_k} + e_{dp}^{A_{k+1}} \) and \( e_{dp}^{A_{k+1}} \) are incomparable; (3) \( e_{dp}^{A_k} \neq e_{dp}^{A_{k+1}} \) and \( e_{dp}^{A_{k+1}} \) are incomparable.

Then, when rule \texttt{cr4} is applied, we have shown that \( \Psi \) preserves in \( M \) property (10) for \( Can^{k+1}(O) \).

Consider now property (8). We have that \( a^{A_{k+1}} = a^{A_k} \cup \{(e, s)\} \), and for every \( e_{dp} \neq a \), we have that \( e_{dp}^{A_k} = e_{dp}^{A_{k+1}} \). Let \( e_{dt} \in D_{qA} \), then we have three possible scenarios: (1) \( e_{dt}^{T_k} \subseteq \{v|(e, v) \in a^{A_k}\} \), (2) \( \{v|(e, v) \in a^{A_k}\} \subseteq e_{dt}^{T_k} \) and, by inductive hypothesis, we have that \( \{v'|(e, v') \in \Psi(a)^{A_M}\} \subseteq \Psi(e_{dt})^{T_M} \), otherwise we have that \( \{v|(e, v) \in a^{A_k}\} \) and \( e_{dt}^{T_k} \) are incomparable; (3) \( \{v|(e, v) \in a^{A_k}\} \) and \( e_{dt}^{T_k} \) are incomparable or \( e_{dt}^{T_k} \subseteq \{v|(e, v) \in a^{A_k}\} \); (4) Since \( e_{dt}^{T_k} \neq a^{A_k} \), and (5) \( e_{dt}^{T_k} \neq a^{A_{k+1}} \), we have that \( e_{dt}^{T_k} \) and \( e_{dt}^{T_k} \) are incomparable.

Then, when rule \texttt{cr4} is applied, we have shown that \( \Psi \) is an extended homomorphism from \( Can^{k+1}(O) \) to \( M \).

\texttt{cr17} In this case we have that \texttt{ObjectPropertyAssertion} \( (p_1, e_1, e_2) \) is in \( \textit{Chase}^k(O) \), \texttt{SubObjectPropertyOf} \( (p_1, p_2) \) is in \( O \), \texttt{ObjectPropertyAssertion} \( (p_2, e_1, e_2) \) is not in \( \textit{Chase}^k(O) \), and \( \textit{Chase}^{k+1}(O) = \textit{Chase}^k(O) \cup \{\texttt{ObjectPropertyAssertion} \ (p_2, e_1, e_2)\} \). As for every \( e_p \) we have that \( e_{p}^{A_{k+1}} = e_{p}^{A_k} \) and \( e_{p}^{T_{k+1}} = e_{p}^{T_k} \), by inductive hypothesis we know that \( \Psi \) preserves in \( M \) properties (8) and (10) for \( Can^{k+1}(O) \).

Consider now property (7), and let \( e_{p_2} = \texttt{ObjectSomeValuesFrom}(p_2 \textit{owl:Thing}) \) and \( e_{Inv(p_2)} = \texttt{ObjectSomeValuesFrom}(\texttt{ObjectInverseOf}(p_2 \textit{owl:Thing})) \). Then we have that \( e_{p_2} = \texttt{ObjectSomeValuesFrom}(p_2 \textit{owl:Thing}) \) and \( e_{Inv(p_2)} = \texttt{ObjectSomeValuesFrom}(\texttt{ObjectInverseOf}(p_2 \textit{owl:Thing})) \). Then we have six possible scenarios: (1) \( e_{p_2} \subseteq e_{p_2}^{E_k} \cup \{e_1\} \), (2) \( e_{p_2} \cup e_{p_2}^{E_k} \cup \{e_2\} \) and for every \( e_{cl} \) such that \( e_{cl} \neq e_{p_2} \) and \( e_{cl} \neq e_{Inv(p_2)} \) we have that \( e_{cl}^{E_{k+1}} = e_{cl}^{E_k} \). Let us first consider \( e_{p_2} \). Then we have six possible scenarios: (1) \( e_{p_2} \subseteq e_{p_2}^{E_k} \cup \{e_1\} \), (2) \( e_{p_2} \cup e_{p_2}^{E_k} \cup \{e_2\} \), (3) \( e_{p_2} \\cup e_{p_2}^{E_k} \) and \( e_{p_2} \neq e_{p_2}^{E_k} \), (4) \( e_{p_2} \) and \( e_{p_2}^{E_k} \) are incomparable, and \( e \notin e_{p_2}^{E_k} \), (5) \( e_{p_2} \) and \( e_{p_2}^{E_k} \) are incomparable, and \( e \notin e_{p_2}^{E_k} \), and there exists an object \( e' \neq e \) in \( \Delta_k^o \) such that \( e' \notin e_{p_2}^{E_k} \) and \( e' \in e_{p_2}^{E_k} \), and (6) \( e_{p_2}^{E_k} \) and \( e_{p_2}^{E_k} \) are incomparable, \( e \notin e_{p_2}^{E_k} \), and does not exists any object \( e' \neq e \) in \( \Delta_k^o \) such that \( e' \notin e_{p_2}^{E_k} \) and \( e' \in e_{p_2}^{E_k} \).
(1) In this case we have that $e_{cl}^{E_{k+1}} \subseteq e_{cl}^{E_k}$ and, by inductive hypothesis, we know that $\Psi(e_{cl})^{E_M} \subseteq \Psi(e_{cl})^{E_M}$.

(2) In this case we have that $e_{cl}^{E_{k+1}} \subseteq e_{cl}^{E_k}$ and, by inductive hypothesis, we know that $\Psi(e_{cl})^{E_M} \subseteq \Psi(e_{cl})^{E_M}$.

(3)-(5) In these cases we have that $e_{cl}^{E_{k+1}}$ and $e_{cl}^{E_k}$ are incomparable.

(6) In this case we have that $e_{cl}^{E_{k+1}} \subseteq e_{cl}^{E_k}$. Thus, by Lemma 53, we know that $O \models \text{SubClassOf}(e_{cl} e_{op})$ and, being $M$ a model of $O$, we have that $M \models \text{SubClassOf}(e_{cl} e_{op})$, and therefore $\Psi(e_{cl})^{E_M} \subseteq \Psi(e_{cl})^{E_M}$.

As similar proof can be given when one considers $e_{Inv(p_2)}$ instead of $e_{op}$, when rule cr17 is applied, we have shown that $\Psi$ preserves in $M$ property (7) for $\text{Can}^{k+1}(O)$.

Consider now property (9) and let $\text{Inv}(p_2) = \text{ObjectInverseOf}(p_2)$. Then we have that $p_2^{R_{k+1}} = p_2^{R_k} \cup \{(e_1, e_2)\}$, $\text{Inv}(p_2)^{R_k+1} = \text{Inv}(p_2)^{R_k} \cup \{(e_2, e_1)\}$, and for every $e_{op}$ such that $e_{op} \neq p_2$ and $e_{op} \neq \text{Inv}(p_2)$, we have that $e_{op}^{R_{k+1}} = e_{op}^{R_k}$. Let $e_{op} \neq p_2$, then we have six possible scenarios: (1) $e_{op}^R \subseteq p_2^R$, (2) $p_2^R \subseteq e_{op}^R$, and (3) $e_{op}^R \subseteq p_2^R$ and $e_{op} \neq e_{op}^R$, (4) $e_{op}^R \subseteq p_2^R$ and $e_{op} \neq p_2^R$ are incomparable, and $(e_1, e_2) \notin p_2^R$ and $p_2^R$ are incomparable, (5) $e_{op}^R$ and $p_2^R$ are incomparable, and there exists a couple of objects $(e', e'') \neq (e_1, e_2)$ in $\Delta^{k+1}_i \times \Delta^{k+1}_i$ such that $(e', e'') \in e_{op}^{R_k}$ and $(e', e'') \notin p_2^{R_k}$, and (6) $e_{op}^R$ and $p_2^R$ are incomparable, and $(e_1, e_2) \in e_{op}^{R_k}$, and does not exist any couple of objects $(e', e'') \neq (e_1, e_2)$ in $\Delta^{k+1}_i \times \Delta^{k+1}_i$ such that $(e', e'') \in e_{op}^{R_k}$ and $(e', e'') \notin p_2^{R_k}$.

(1) In this case we have that $e_{op}^{R_{k+1}} \subseteq p_2^{R_{k+1}}$ and, by inductive hypothesis, we know that $\Psi(e_{op})^{E_M} \subseteq \Psi(p_2)^{E_M}$.

(2) In this case we have that $p_2^{R_{k+1}} \subseteq e_{op}^{R_{k+1}}$ and, by inductive hypothesis, we know that $\Psi(p_2)^{E_M} \subseteq \Psi(e_{op})^{E_M}$.

(3)-(5) In these cases we have that $p_2^{R_{k+1}}$ and $e_{op}^{R_{k+1}}$ are incomparable.

(6) In this case we have that $e_{op}^{R_{k+1}} \subseteq p_2^{R_{k+1}}$. Thus, by Lemma 53, we know that $O \models \text{SubObjectPropertyOf}(e_{op} p_2)$, and, being $M$ a model of $O$, we have that $M \models \text{SubObjectPropertyOf}(e_{op} p_2)$, and therefore $\Psi(e_{op})^{E_M} \subseteq \Psi(p_2)^{E_M}$.

Similar proof can be given if one considers $\text{Inv}(p_2)$ instead of $p_2$. Then, when rule cr17 is applied, we have shown that $\Psi$ preserves in $M$ property (9) for $\text{Can}^{k+1}(O)$.

Consider now property (14). Since we know that $\Delta^{k+1}_i = \Delta^{k}_i$, we distinguish between two cases: (a) $e_1 \neq e_2$, and (b) $e_1 = e_2$.

(a) In this case for every $e_{in} \in \Delta^{k+1}_i$, we know that $(e_{in}, e_{in}) \in p_2^{R_{k+1}}$ if and only $e_{in} \in \Delta^{k}_i$, $(e_{in}, e_{in}) \in p_2^{R_{k}}$, and, by inductive hypothesis we know that for every $e_{in} \in \Delta^{k}_i$, $(e_{in}, e_{in}) \in \Psi(p_2)^{E_M}$.

(b) Let $e = e_1 = e_2$. In this case, if for every $e_{in} \in \Delta^{k}_i$, $e_{in} \neq e$, we have that $(e_{in}, e_{in}) \in p_2^{R_{k+1}}$, then for every $e \in \Delta^{k+1}_i$, we have that $(e, e) \in p_2^{R_{k+1}}$ and therefore $\text{Can}^{k+1}(O) \models \text{ReflexiveObjectProperty}(p_2)$. Thus, by
Lemma 53, we know that $\mathcal{O} \models \text{ReflexiveObjectProperty}(p_2)$ and, being $M$ a model of $\mathcal{O}$, $M \models \text{ReflexiveObjectProperty}(p_2)$. Then for every $e_3 \in \Delta_1^M \cdot (e_3, e_3) \in \Psi(p_2)^{R_M}$.

Similar proof can be given if one considers $\text{Inv}(p_2)$ instead of $p_2$. Then, when rule cr17 is applied, we have shown that $\Psi$ preserves in $M$ property (14) for $\text{Can}^{k+1}(\mathcal{O})$, and therefore $\Psi$ is an extended homomorphism from $\text{Can}^{k+1}(\mathcal{O})$ to $M$.

**cr19** In this case we have that DataPropertyAssertion $(a_1 \ e \ lt)$ is in $\text{Chase}^k(\mathcal{O})$, SubpropertyOf $(a_1 \ a_2)$ is in $\mathcal{O}$, DataPropertyAssertion $(a_2 \ e \ lt)$ is not in $\text{Chase}^k(\mathcal{O})$, and $\text{Chase}^{k+1}(\mathcal{O}) = \text{Chase}^k(\mathcal{O}) \cup \{\text{DataPropertyAssertion} \ a_2 \ e \ lt\}$. As for every $e_p$ we have that $e_p \in \text{Chase}(\mathcal{O})$, by inductive hypothesis, we know that $\Psi$ preserves in $M$ properties (9) and (14) for $\text{Can}^{k+1}(\mathcal{O})$.

Consider now property (7), and let $e_{o_2} = \text{DataSomeValuesFrom}(a_2 \ \text{rdfs:Literal})$. Then we have that $e_{o_2}^{E_{k+1}} = e_{o_2} \cup \{e\}$ and for every $e_{cl} \neq e_{o_2}$ we have that $e_{cl}^{E_{k+1}} = e_{cl}^{E_k}$. Moreover only one of the following can be verified: (1) $e_{cl}^{E_k} \subseteq e_{o_2}^{E_k}$, (2) $e_{cl}^{E_k} \subseteq e_{o_2}^{E_k}$ and $e \in e_{o_2}^{E_k}$, (3) $e_{cl}^{E_k} \subseteq e_{o_2}^{E_k}$ and $e \notin e_{o_2}^{E_k}$, (4) $e_{cl}^{E_k} \subseteq e_{o_2}^{E_k}$, (5) $e_{cl}^{E_k}$ and $e_{o_2}^{E_k}$ are incomparable, and $e \in e_{o_2}^{E_k}$, there exists an object $e' \neq e$ in $\Delta_k^\mathcal{O}$ such that $e' \notin e_{cl}^{E_k}$ and $e' \in e_{o_2}^{E_k}$, and (6) $e_{cl}^{E_k}$ and $e_{o_2}^{E_k}$ are incomparable, $e \in e_{o_2}^{E_k}$, and does not exists any object $e' \neq e$ in $\Delta_k^\mathcal{O}$ such that $e' \notin e_{cl}^{E_k}$ and $e' \in e_{o_2}^{E_k}$.

1. In this case we have that $e_{cl}^{E_{k+1}} \subseteq e_{o_2}^{E_{k+1}}$ and, by inductive hypothesis, we know that $\Psi(e_{cl}^{E_M}) \subseteq \Psi(e_{o_2}^{E_M})$.
2. In this case we have that $e_{o_2}^{E_{k+1}} \subseteq e_{cl}^{E_{k+1}}$ and, by inductive hypothesis, we know that $\Psi(e_{o_2}^{E_M}) \subseteq \Psi(e_{cl}^{E_M})$.
3. In these cases we have that $e_{cl}^{E_{k+1}}$ and $e_{o_2}^{E_{k+1}}$ are incomparable.
4. In these case we have that $e_{o_2}^{E_{k+1}} \subseteq e_{cl}^{E_{k+1}}$. Thus, by Lemma 53, we know that $\mathcal{O} \models \text{SubClassOf} \ (e_{cl} \ e_{o_2})$, and being $M$ a model of $\mathcal{O}$, we have that $M \models \text{SubClassOf} \ (e_{cl} \ e_{o_2})$, and therefore $\Psi(e_{cl}^{E_M}) \subseteq \Psi(e_{o_2}^{E_M})$.

Then, when rule cr19 is applied, we have shown that preserves in $M$ property (7) for $\text{Can}^{k+1}(\mathcal{O})$.

Consider now property (10). We have that $a_{2}^{A_{k+1}} = p^{A_{k}} \cup \{(e_1, e_2)\}$, and for every $e_{dp} \neq a_2$ we have that $e_{dp}^{A_{k+1}} = e_{dp}^{A_{k}}$. Then we have six possible scenarios:

1. $e_{dp}^{A_{k}} \subseteq a_2^{A_{k}}$, (2) $a_2^{A_{k}} \subseteq e_{dp}^{A_{k}}$ and $(e, lt) \in e_{dp}^{A_{k}}$, (3) $a_2^{A_{k}} \subseteq e_{dp}^{A_{k}}$ and $(e, lt) \notin e_{dp}^{A_{k}}$, (4) $e_{dp}^{A_{k}}$ and $a_2^{A_{k}}$ are incomparable, and $(e, lt) \notin e_{dp}^{A_{k}}$, (5) $e_{dp}^{A_{k}}$ and $a_2^{A_{k}}$ are incomparable, $(e, lt) \in e_{dp}^{A_{k}}$, and there exists a couple $(e', lt') \neq (e, lt)$ in $\Delta_k^\mathcal{O} \times \Delta_k^\mathcal{V}$ such that $(e', lt') \in e_{dp}^{A_{k}}$ and $(e', lt') \notin a_2^{A_{k}}$, and (6) $e_{dp}^{A_{k}}$ and $a_2^{A_{k}}$ are incomparable, $(e, lt) \in e_{dp}^{A_{k}}$, and does not exists any couple $(e', lt') \neq (e_1, e_2)$ in $\Delta_k^\mathcal{O} \times \Delta_k^\mathcal{V}$ such that $(e', lt') \in e_{dp}^{A_{k}}$ and $(e', lt') \notin a_2^{A_{k}}$.

1. In this case we have that $e_{dp}^{A_{k+1}} \subseteq a_2^{A_{k+1}}$ and, by inductive hypothesis, we know that $\Psi(e_{dp}^{A_{M}}) \subseteq \Psi(a_2^{A_{M}})$.
In this case we have that \( a_2^{A_k+1} \subseteq e_{dp}^{A_{k+1}} \) and, by inductive hypothesis, we know that \( \Psi(a_2)^{A_M} \subseteq \Psi(e_{dp})^{A_M} \).

(3)-(5) In these cases we have that \( a_2^{A_k+1} \) and \( e_{dp}^{A_{k+1}} \) are incomparable.

(6) In this case we have that \( e_{dp}^{A_{k+1}} \subseteq a_2^{A_{k+1}} \). Thus, by Lemma 53, we know that \( O \models \text{SubDataPropertyOf}(e_{dp}, a_2) \) and, being \( M \) a model of \( O \), we have that \( M \models \text{SubDataPropertyOf}(e_{dp}, a_2) \), and therefore \( \Psi(e_{dp})^{A_M} \subseteq \Psi(e_{dp})^{A_M} \).

Then, when rule cr19 is applied, we have shown that \( \Psi \) preserves in \( M \) property (10) for \( \text{Can}^{k+1}(O) \).

Consider now property (8). We have that \( a_2^{A_k+1} = a^{A_k} \cup \{(e, lt)\} \), and for every \( e_{dp} \neq a \), we have that \( e_{dp}^{A_k} = e_{dp}^{A_k} \). Let \( e_{dt} \in D_{qI} \). We have three possible scenarios:

1. \( e_{dt} \subseteq \{v|(e, v) \in a_2^{A_{k+1}}\} \),
2. \( \{v|(e, v) \in a_2^{A_k}\} \subseteq e_{dt} \), and
3. \( \{v|(e, v) \in a_2^{A_k}\} \) and \( e_{dt} \) are incomparable:

   1. In this case we have that \( e_{dt} \subseteq \{v|(e, v) \in a_2^{A_{k+1}}\} \);
   2. In this case if \( lt \subseteq e_{dt} \), then we have that \( \{v|(e, v) \in a_2^{A_k}\} \subseteq e_{dt}^{T_k} \) and, by inductive hypothesis, we have that \( \{v'(e, v') \in \Psi(a)^{A_M}\} \subseteq \Psi(e_{dt})^{T_M} \), otherwise we have that either \( \{v|(e, v) \in a_2^{A_k}\} \) and \( e_{dt}^{T_k} \) are incomparable or \( e_{dt}^{T_k} \subseteq \{v|(e, v) \in a_2^{A_k}\} \);
   3. In this case we have that \( \{v|(e, v) \in a_2^{A_k}\} \) and \( e_{dt}^{T_k} \) are incomparable.

Then, when rule cr19 is applied, we have shown that \( \Psi \) preserves in \( M \) property (8) for \( \text{Can}^{k+1}(O) \), and therefore \( \Psi \) is an extended homomorphism from \( \text{Can}^{k+1}(O) \) to \( M \).

\[ \text{cr20} \quad \text{In this case we have that} \quad \text{DataPropertyAssertion} \quad \text{(a e lt)} \quad \text{is in} \quad \text{Chase}^k(O), \quad \text{DataPropertyRange (a d) is in} \quad O, \quad \text{DatatypeAssertion (d lt) is not in} \quad \text{Chase}^k(O), \quad \text{and} \quad \text{Chase}^{k+1}(O) = \text{Chase}^k(O) \cup \text{DatatypeAssertion (d lt)} \].

As for every \( e_p \) we have that \( e_p^{A_{k+1}} = e_p^{A_k}, e_p^{R_{k+1}} = e_p^{R_k}, e_p^{A_{k+1}} = e_p^{A_k}, \) and \( e_p^{T_{k+1}} = e_p^{T_k} \), by inductive hypothesis we know that \( \Psi \) preserves in \( M \) properties (7), (9), (10) and (14) for \( \text{Can}^{k+1}(O) \). Consider now property (8). We have that \( d \in D_{qI}, d^{T_k+1} = d^{T_k} \cup \{lt\} \), and for every \( e_{dt} \neq d, e_{dt} \in D_{dt} \) we have that \( e_{dt}^{T_k+1} = e_{dt}^{T_k} \). For every \( e_{dp} \) we have six possible scenarios: (1) \( \{v|(e, v) \in e_{dp}^{A_k}\} \subseteq d^{T_k} \), (2) \( d^{T_k} \subseteq Vals^{k}(e_{dp}) \), and \( lt \subseteq \{v|(e, v) \in e_{dp}^{A_k}\} \), (3) \( d^{T_k} \subseteq Vals^{k}(e_{dp}) \), and \( lt \notin \{v|(e, v) \in e_{dp}^{A_k}\} \), (4) \( \{v|(e, v) \in e_{dp}^{A_k}\} \) and \( d^{T_k} \) are incomparable and \( lt \notin \{v|(e, v) \in e_{dp}^{A_k}\} \), (5) \( \{v|(e, v) \in e_{dp}^{A_k}\} \) and \( d^{T_k} \) are incomparable, \( lt \subseteq \{v|(e, v) \in e_{dp}^{A_k}\} \), and there exists \( lt' \neq lt \) such that \( lt' \in \{v|(e, v) \in e_{dp}^{A_k}\} \) and \( lt' \notin d^{T_k} \), and (6) \( \{v|(e, v) \in e_{dp}^{A_k}\} \) and \( d^{T_k} \) are incomparable, \( lt \subseteq \{v|(e, v) \in e_{dp}^{A_k}\} \), and does not exist \( lt' \neq lt \) such that \( lt' \in \{v|(e, v) \in e_{dp}^{A_k}\} \) and \( lt' \notin d^{T_k} \):

1. In this case we have that \( \{v|(e, v) \in e_{dp}^{A_k+1}\} \subseteq d^{T_k+1} \) and, by inductive hypothesis, we have that \( \{v'(e, v') \in \Psi(e_{dp})^{A_M}\} \subseteq \Psi(d)^{T_M} \);
2. In this case we have that \( d^{T_k+1} \subseteq \{v|(e, v) \in e_{dp}^{A_k+1}\} \);
(3-5) In this cases we have that $d^{T_{k+1}}$ and $\{v|(e,v) \in e_{dp}^{A_{k+1}}\}$ are incomparable;

(6) In this case we have that $\{v|(e,v) \in e_{dp}^{A_{k+1}}\} \subset d^{T_{k+1}}$. By definition we have that if an axiom of the form $DataPropertyAssertion(\epsilon_{dp} \ v_{in})$ is contained in $Chase^{3}$, then $Chase^{3}$ contains also an axiom of the form $DatatypeAssertion(\rho_{in})$ for every $d_{in} \in D_{ql}$ such that $\rho_{in}^{LS} \in d^{DT}_{in}$, where $LS$ and $DT$ are the functions defined by the $OWL2QL$ datatye map to interpret literals and datatypes, respectively. Thus for every $v_{in} \in \{v|(e,v) \in e_{dp}^{A_{k+1}}\}$, we have that $v_{in} \in d^{T_{k+1}}$ if and only if $v_{in}^{LS} \in d^{DT}$. Then we know that $\{v|(e,v) \in e_{dp}^{A_{k+1}}\} \subset d^{DT}$, and therefore, since every model of $\mathcal{O}$ must agree with the interpretation of literals and datatypes fixed once and for all by the $OWL2QL$ datatye map, we have that $\{v'|(e,v') \in \Psi(e_{1})^{AM}\} \subseteq \Psi(e_{2})^{TM}$.

Then, when rule $cr20$ is applied, we have shown that $\Psi$ preserves in $M$ property (8) for $Can^{k+1}(\mathcal{O})$, and therefore $\Psi$ is an extended homomorphism from $Can^{k+1}(\mathcal{O})$ to $M$.

$cr21$ In this case we have that $ReflexiveObjectProperty(\rho)$ is in $\mathcal{O}$ and there exists $e \in V_{I}^{O} \cup S_{k}$ such that $ObjectPropertyAssertion(\rho \ v)$ is not in $Chase^{k}(\mathcal{O})$. Then $Chase^{k+1}(\mathcal{O}) = Chase^{k}(\mathcal{O}) \cup \{ObjectPropertyAssertion(\rho \ v)\}$. As for every $\epsilon_{a}$ we have that $e_{a}^{A_{k+1}} = e_{a}$ and $e_{T_{k+1}}^{A_{k+1}} = e_{T_{k}}$, by inductive hypothesis we know that $\Psi$ preserves in $M$ properties (8) and (10) for $Can^{k+1}(\mathcal{O})$.

Consider now property (7) and let $e_{p} = ObjectSomeValuesFrom(\rho \ owl:Thing)$, $e_{Inv(p)} = ObjectSomeValuesFrom(ObjectInverseOf(\rho \ owl:Thing))$. Then we have that $e_{p}^{E_{k+1}} = e_{p}^{E_{k}} \cup \{e\}, e_{Inv(p)}^{E_{k+1}} = e_{Inv(p)}^{E_{k}} \cup \{e\}$, and for every $e_{cl}$ such that $e_{cl} \neq e_{p}$ and $e_{cl} \neq e_{Inv(p)}$ we have that $e_{cl}^{E_{k+1}} = e_{cl}^{E_{k}}$. Let us first consider $e_{p}$ and $e_{cl}$ such that $e_{cl} \neq e_{p}$. We have that only one of the following can be verified: (1) $e_{cl}^{E_{k}} \subseteq e_{p}^{E_{k}}$, (2) $e_{p}^{E_{k}} \subseteq e_{cl}^{E_{k}}$ and $e \in e_{cl}^{E_{k}}$, (3) $e_{p}^{E_{k}} \subseteq e_{cl}^{E_{k}}$ and $e \notin e_{cl}^{E_{k}}$, (4) $e_{cl}^{E_{k}}$ and $e_{p}^{E_{k}}$ are incomparable, and $e \notin e_{cl}^{E_{k}}$, (5) $e_{cl}^{E_{k}}$ and $e_{p}^{E_{k}}$ are incomparable, $e \in e_{cl}^{E_{k}}$, and there exists an object $e' \neq e$ in $\Delta^{k}$ such that $e' \notin e_{cl}^{E_{k}}$ and $e' \in e_{p}^{E_{k}}$, and (6) $e_{cl}^{E_{k}}$ and $e_{p}^{E_{k}}$ are incomparable, $e \in e_{cl}^{E_{k}}$, and does not exists any object $e' \neq e$ in $\Delta^{k}$ such that $e' \notin e_{cl}^{E_{k}}$ and $e' \in e_{p}^{E_{k}}$.

(1) In this case we have that $e_{cl}^{E_{k+1}} \subseteq e_{p}^{E_{k+1}}$ and, by inductive hypothesis, we know that $\Psi(e_{cl})^{EM} \subseteq \Psi(e_{p})^{EM}$.

(2) In this case we have that $e_{p}^{E_{k+1}} \subseteq e_{cl}^{E_{k+1}}$ and, by inductive hypothesis, we know that $\Psi(e_{p})^{EM} \subseteq \Psi(e_{cl})^{EM}$.

(3-5) In these cases we have that $e_{cl}^{E_{k+1}}$ and $e_{cl}^{E_{k+1}}$ are incomparable.

(6) In this case we have that $e_{cl}^{E_{k+1}} \subseteq e_{p}^{E_{k+1}}$. Thus, by Lemma 53 we know that $\mathcal{O} \models SubClassOf (e_{cl} \ e_{p})$ and, being $M$ a model of $\mathcal{O}$, we have that $M = SubClassOf (e_{cl} \ e_{p})$, and therefore $\Psi(e_{cl})^{EM} \subseteq \Psi(e_{p})^{E_{k+1}}$.

Similar proof can be given if one considers $Inv(p)$ instead of $p$. Then, when rule $cr21$ is applied, we have shown that $\Psi$ is an extended homomorphism from $Can^{k+1}(\mathcal{O})$ to $M$. 

Consider now property (9) and let $Inv(p) = ObjectInverseOf(p)$. Then we have that $p^{R_{k+1}} = p^{R_k} \cup \{(e,e)\}$, $Inv(p)^{R_{k+1}} = Inv(p)^{R_k} \cup \{(e,e)\}$, and for every $e_{op}$ such that $e_{op} \neq p$ and $e_{op} \neq Inv(p)$, we have that $e_{op}^{R_{k+1}} = e_{op}^{R_k}$. Let us first consider $p$ and $e_{op}$ such that $e_{op} \neq e$. We have six possible scenarios: (1) $e_{op}^{R_k} \subseteq p^{R_k}$, (2) $p^{R_k} \subseteq e_{op}^{R_k}$ and $(e_1, e_2) \in e_{op}^{R_k}$, (3) $p^{R_k} \subseteq e_{op}^{R_k}$ and $(e_1, e_2) \notin e_{op}^{R_k}$, (4) $e_{op}^{R_k}$ and $p^{R_k}$ are incomparable, and $(e_1, e_2) \notin e_{op}^{R_k}$, (5) $e_{op}^{R_k}$ and $p^{R_k}$ are incomparable, $(e_1, e_2) \notin e_{op}^{R_k}$, and there exists a couple of objects $(e', e'') \neq (e_1, e_2)$ in $\Delta^{k}_{o} \times \Delta^{k}_{o}$ such that $(e', e'') \in e_{op}^{R_k}$ and $(e', e'') \notin p^{R_k}$, and (6) $e_{op}^{R_k}$ and $p^{R_k}$ are incomparable, $(e_1, e_2) \in e_{op}^{R_k}$, and does not exists any couple of objects $(e', e'') \neq (e_1, e_2)$ in $\Delta^{k}_{o} \times \Delta^{k}_{o}$ such that $(e', e'') \in e_{op}^{R_k}$ and $(e', e'') \notin p^{R_k}$:

(1) In this case we have that $e_{op}^{R_{k+1}} \subseteq p^{R_{k+1}}$ and, by inductive hypothesis, we know that $\Psi(e_{op})^{R_M} \subseteq \Psi(p)^{R_M}$.

(2) In this case we have that $p^{R_{k+1}} \subseteq e_{op}^{R_{k+1}}$ and, by inductive hypothesis, we know that $\Psi(p)^{R_M} \subseteq \Psi(e_{op})^{R_M}$.

(3)-(5) In these cases we have that $p^{R_{k+1}}$ and $e_{op}^{R_{k+1}}$ are incomparable.

(6) In this case we have that $e_{op}^{R_{k+1}} \subseteq p^{R_{k+1}}$. Thus, by Lemma 52, we know that $\mathcal{O} \models \text{SubObjectPropertyOf}(e_{op}, p)$ and, being $M$ a model of $\mathcal{O}$, we have that $M \models \text{SubObjectPropertyOf}(e_{op}, p)$, and therefore $\Psi(e_{op})^{R_M} \subseteq \Psi(p)^{R_M}$.

Similar proof can be given if one considers $Inv(p)$ instead of $p$. Then, when rule cr21 is applied, we have shown that $\Psi$ preserves in $M$ property (9) for $Can^{k+1}(\mathcal{O})$.

Consider now property (14). We know that $\mathcal{O} \models \text{ReflexiveObjectProperty}(p)$ and, being $M$ a model of $\mathcal{O}$, $M \models \text{ReflexiveObjectProperty}(p)$. Thus if for every $e_{in} \in \Delta^{k+1}_{i}$ we have that $(e_{in}, e_{in}) \in p^{R_{k+1}}$, we know that for every $e_3 \in \Delta^{M}_{i}$, $(e_3, e_3) \in \Psi(p)^{R_M}$. Similar proof can be given if one considers $Inv(p)$ instead of $p$. Then, when rule cr17 is applied, we have shown that $\Psi$ preserves in $M$ property (14) for $Can^{k+1}(\mathcal{O})$, and therefore $\Psi$ is an extended homomorphism from $Can^{k+1}(\mathcal{O})$ to $M$.

If $Can^{k+1}$ is obtained starting from $Can^{k}(\mathcal{O})$ by applying a rule identified by a number in the set \{3,5,6,7,8,9,10,11,12,13,14,15,16,18\}, the proof can be given by suitably combining the proofs given for rules in \{1,2,4,7,17,19,20,21\}.

Now that we have shown that, for every model $M$ of $\mathcal{O}$, there exists an extended homomorphism from $Can(\mathcal{O})$ to $M$, we can extend Theorem 44 to metaground queries, i.e., we show that, for every satisfiable TBox-complete ontology $\mathcal{O}$, $Can(\mathcal{O})$ is universal with respect to metaground queries.

**Theorem 56** Let $\mathcal{O}$ be a satisfiable TBox-complete ontology and let $Q$ be a metaground boolean query over $\mathcal{O}$. We have that $\mathcal{O} \models Q$ if and only if $Can(\mathcal{O}) \models Q$.

**Proof.** The proof is based on Proposition 55 and follows the same line of reasoning of Proposition 44. \qed
Let $\mathcal{O}^{all}$ denote the set of all axioms that is possible to build starting from suitable expressions in $\text{Exp}^\mathcal{O}$, i.e., $\mathcal{O}^{all} = \mathcal{P}^\mathcal{O} \cup \mathcal{N}^\mathcal{O}$. It is easy to see that the following corollary holds.

**Proposition 57** Let $\mathcal{O}$ be a satisfiable TBox-complete ontology and let $\alpha$ be an axiom in $\mathcal{O}^{all}$. We have that $\mathcal{O} \models \alpha$ if and only if $\text{Can}(\mathcal{O}) \models \alpha$.

Proposition 57 and Theorem 56 provide the basis to answer metaground queries over TBox-complete ontologies, as shown by the following Theorem.

**Theorem 58** Let $\mathcal{O}$ be a satisfiable ontology and let $Q = q_1 \cup \ldots \cup q_n$ be a metaground boolean query over $\mathcal{O}$. $\mathcal{O} \models Q$ if and only if $\text{Can}(\mathcal{O}) \models \text{int}(q_i)$ and $\text{Can}(\mathcal{O}) \models \text{ext}(q_i)$, for some $i \in \{1, \ldots, n\}$, where $\text{ext}(q_i)$ and $\text{int}(q_i)$ denote the conjunctions of extensional and intensional atoms of $q_i$, respectively.

Theorem 58 provides the following algorithm solving the problem of query answering over TBox-complete ontologies:

**Algorithm 3: TBoxCompleteAns**

Input : satisfiable TBox-complete Hi(OWL 2 QL) ontology $\mathcal{O}$, boolean query $Q$ over $\mathcal{O}$

Output : true if $\mathcal{O} \models Q$, false otherwise

1. $Q' \leftarrow \text{MG}(Q, \mathcal{O})$;
2. if $\mathcal{O} \models_{\text{FOL}} Q'$ then
   3. return true;
4. else
   5. return false;
6. end

The following theorem highlights the importance of metagrounding for TBox-complete ontologies.

**Theorem 59** Let $\mathcal{O}$ be a satisfiable TBox-complete ontology and let $Q = q_1 \cup \ldots \cup q_n$ be a boolean query over $\mathcal{O}$. Then, $\mathcal{O} \models Q$ if and only if $\mathcal{O} \models q'$ for some $q' \in \text{MG}(Q, \mathcal{O})$.

Proof. $\Leftarrow$: By Theorem 58, if $\mathcal{O} \models q'$ for some $q'$ in $\text{MG}(Q, \mathcal{O})$, then $\text{Can}(\mathcal{O}) \models \text{int}(q')$ and $\text{Can}(\mathcal{O}) \models \text{ext}(q')$. Let $q_i$ be the disjunct of $Q$ such that $q' \in \text{MG}(q_i, \mathcal{O})$, $\mu'$ be the instantiation of $q_i$ that led to $q'$, and let $\mu$ be obtained by extending $\mu'$ with the assignment of the variables of $\text{ext}(q')$ that makes $\text{ext}(q')$ evaluate to true in $\text{Can}(\mathcal{O})$. It follows that $\mathcal{O} \models \mu(q)$ and, hence, $\mathcal{O} \models Q$.

$\Rightarrow$: Suppose that $\mathcal{O} \models Q$. If $\mathcal{O} \models Q$, there exists an assignment $\mu$ of all variables of $Q$ such that $\mathcal{O} \models \mu(q_1) \cup \ldots \cup \mu(q_n)$. Thus, by Theorem 58, there exists $i \in \{1, \ldots, n\}$ such that $\text{Can}(\mathcal{O}) \models \mu(q_i)$. Consider the restriction $\mu'$ of $\mu$ to the metavariables of $q_i$ and let $q' = \mu'(q_i)$. Clearly, $q' \in \text{MG}(Q, \mathcal{O})$ and $\mathcal{O} \models q'$, which concludes the proof.

Theorem 59 combined with Theorem 58 provides the following algorithm solving the problem of query answering over TBox-complete ontologies:
8.2 Answering general queries

Let us now come back to the case of non TBox-Complete ontologies, i.e. ontologies for which there exist a set of negative axioms in \(N^O\) that are uncertain. In order to solve the issue highlighted by Example 48 for such ontologies, we now introduce the notion of completion of an ontology, that is crucial to characterize the sets of models of a non-TBox-complete ontology under MMS and relies on the notion of violation set of a negative axiom. Intuitively, given an axiom \(\alpha\) in \(N^O\), the violation set of \(\alpha\), denoted by \(F^\alpha\), is the set containing the ABox axioms that are necessary and sufficient to make \(\alpha\) be violated by the ontology \(O \cup F^\alpha\). The violation set \(F^\alpha\) for every possible axiom in \(N^O\) are depicted in Table 8.1 where \(e\), possibly with subscripts, denotes a complex expression in \(Exp^O\), \(c\), possibly with subscripts, denotes an expression in \(V^O_C\), \(p\), possibly with subscripts, denotes an expression in \(V^O_{OP}\), \(dp\), possibly with subscripts, denotes an expression in \(V^O_{DP}\), \(s_\alpha\), \(s'_\alpha\), \(s''_\alpha\) denote distinct IRIs specific for \(\alpha\) that are not contained in \(V^O_N\), and \(w_\alpha\) denotes any literal that do not belong to \(L^O\). Let \(U^O\subseteq N^O\) be the set of uncertain negative axioms in \(O\), \(\sigma\) be a set of axioms such that \(\sigma\subseteq U^O\). The completion of \(O\) with \(\sigma\) is the ontology, denoted as \(O^\sigma\), defined as follows:

\[
O^\sigma = O \cup \sigma \cup C^O \setminus \sigma
\]

where \(C^O \setminus \sigma\) is the union of the violation sets of every intensional assertion in \(U^O\) that is not in \(\sigma\). Intuitively, the completion of \(O\) with \(\sigma\) is the ontology that is obtained by asserting that all axioms in \(\sigma\) are satisfied by \(O\) and that all axioms in \(U^O\) that are not in \(\sigma\) are violated by \(O\). Note that \(O^\sigma\) may be satisfiable or not.

It is straightforward to see that, by definition, given an ontology \(O\), for every subset \(\sigma\) of \(U^O\), we have that \(O^\sigma\) is TBox-complete.

The following example shows how, given a non-TBox-ontology \(O\), we can compute the completion \(O^\sigma\) of \(O\).

**Example 60** Let \(O_1\) be the ontology considered in Example 48, that is formed by the following axioms:

\[
\begin{align*}
\text{DisjointClasses} &: (A : C) \\
\text{DisjointClasses} &: (A \text{ObjectSomeValuesFrom} (R \text{owl:Thing})) \\
\text{DisjointClasses} &: (A \text{ObjectSomeValuesFrom} (\text{ObjectInverseOf} (R \text{owl:Thing}))) \\
\text{ClassAssertion} &: (B : F) \\
\text{ClassAssertion} &: (C : F) \\
\text{ObjectPropertyAssertion} &: (R : F : F) \\
\text{ObjectPropertyAssertion} &: (R : C : A) \\
\text{ObjectPropertyAssertion} &: (R : B : C)
\end{align*}
\]

One can easily verify that \(U^O_1 = \{\text{DisjointClasses} (:A : B)\}\), as all other axioms in \(N^O_1\) are either implied or violated by \(O_1\). Let \(\sigma_0 = \emptyset\), \(\sigma_1 = \{\text{DisjointClasses} (:A : B)\}\), \(\sigma_2 = \{\text{DisjointClasses} (:B : A)\}\), and \(\sigma_3 = \{\text{DisjointClasses} (:A : B), \text{DisjointClasses} (:B : A)\}\). Then we can build the following the completions of \(O_1\):
8. Answering general queries

<table>
<thead>
<tr>
<th>Negative axiom $\alpha$</th>
<th>Violation set of $\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DisjointClasses($c_1,c_2$)</td>
<td>{ClassAssertion($c_1, s_1$), ClassAssertion($c_2, s_2$)}</td>
</tr>
<tr>
<td>DisjointClasses($c$) st. $e$:ObjectSomeValuesFrom($p$,owl:Thing)</td>
<td>{ClassAssertion($c, s_1$), ObjectPropertyAssertion($p, s_1, s'_1$)}</td>
</tr>
<tr>
<td>DisjointClasses($c$) st. $e$:ObjectSomeValuesFrom(ObjectInverseOf($p$,owl:Thing))</td>
<td>{ClassAssertion($c, s_1$), ObjectPropertyAssertion($p, s_1, s'_1$)}</td>
</tr>
<tr>
<td>DisjointClasses($c$) st. $e$:DataSomeValuesFrom($d p$,dt)</td>
<td>{ClassAssertion($c, s_1$), DataPropertyAssertion($d p, s_1, s'_1$)}</td>
</tr>
<tr>
<td>DisjointClasses($e_1,e_2$) st. $e_1$:ObjectSomeValuesFrom($p_1$,owl:Thing) and $e_2$:ObjectSomeValuesFrom($p_2$,owl:Thing)</td>
<td>{ObjectPropertyAssertion($p_1, s_1, s'_1$), ObjectPropertyAssertion($p_2, s_1, s'_1$)}</td>
</tr>
<tr>
<td>DisjointObjectProperties($p_1$, $p_2$)</td>
<td>{ObjectPropertyAssertion($p_1, s_1, s'_1$), ObjectPropertyAssertion($p_2, s_1, s'_1$)}</td>
</tr>
<tr>
<td>DisjointDataProperties($d_{p_1}$,$d_{p_2}$)</td>
<td>{DataPropertyAssertion($d_{p_1}, s_1, s'<em>1$), DataPropertyAssertion($d</em>{p_2}, s_1, s'_1$)}</td>
</tr>
<tr>
<td>IrreflexiveObjectProperty($p$)</td>
<td>{ObjectPropertyAssertion($p, s_1, s'_1$)}</td>
</tr>
</tbody>
</table>

Table 8.1. Violation sets of negative axioms

$O^\alpha_1 = O_1 \cup \{\text{ClassAssertion}(:A \ s), \text{ClassAssertion}(:B \ s'), \text{ClassAssertion}(:b \ s')\}$
$O^\alpha_1 = O_1 \cup \{\text{ClassAssertion}(:A \ s), \text{ClassAssertion}(:s)\}$
$O^\beta_2 = O_1 \cup \{\text{DisjointClasses}(:A :B), \text{ClassAssertion}(:B \ s), \text{ClassAssertion}(:A \ s)\}$
$O^\beta_1 = O_1 \cup \{\text{DisjointClasses}(:B :A), \text{DisjointClasses}(:A :B)\}$

where $s, s'$ denote IRIs not in $V^O_N$. Note, in particular, that all completions are TBox-complete and that $O^\alpha_1$ and $O^\beta_2$ are unsatisfiable.

In the following, we denote by $\text{Mod}(O)$ the set containing all the models of $O$. Building upon the previous sections, we have the following theorems.

**Proposition 61 (Completeness of the completion)** Let $O$ be a satisfiable ontology. We have that $\text{Mod}(O) \subseteq \bigcup_{\alpha \in \alpha(O)} \text{Mod}(O^\alpha)$.
8.2 Answering general queries

Proof. Let $M$ be a model of $\mathcal{O}$. For every $Ax \in \mathcal{U}^\mathcal{O}$, either $M \models Ax$ or $M \not\models Ax$. Thus, let $\sigma$ be the subset of axioms of $\mathcal{U}^\mathcal{O}$ that are satisfied in $M$. By construction, $M \models \mathcal{O}^\sigma$, which concludes the proof. \qed

**Proposition 62** (Correctness of the completion) Let $\mathcal{O}$ be a satisfiable ontology. We have that $\bigcup_{\sigma \subseteq \mathcal{U}^{\mathcal{O}}} \operatorname{Mod}(\mathcal{O}^\sigma) \subseteq \operatorname{Mod}(\mathcal{O})$.

Proof. Follows straightforwardly from the fact that $\mathcal{O} \subseteq \mathcal{O}^\sigma$. \qed

It is easy to see that from Propositions 61 and 62 we can derive the following theorem.

**Theorem 63** Let $\mathcal{O}$ be a satisfiable ontology and $Q$ be a boolean query. Then $\mathcal{O} \models Q$ if and only if $\mathcal{O}^\sigma \models Q$, for every subset $\sigma$ of $\mathcal{U}^{\mathcal{O}}$.

The following example shows how $\mathcal{O}^\sigma$ can be exploited to solve the query answering problem over general ontologies.

**Example 64** Consider the ontology $\mathcal{O}_1$ and the query $Q$ discussed in Example 48. It is easy to see that $\mathcal{O}_1^{\sigma_0} \models Q$ and $\mathcal{O}_1^{\sigma_3} \models Q$, as expected. \qed

Combined with Theorem 59 Theorem 63 provides the following algorithm for answering queries over general Hi(OWL 2 QL) ontologies.

**Algorithm 4: QueryAns**

Input : satisfiable Hi(OWL 2 QL) ontology $\mathcal{O}$, boolean query $Q$

Output : true if $\mathcal{O} \models Q$, false otherwise

1. $Q' \leftarrow \operatorname{MG}(Q, \mathcal{O})$;
2. $\mathcal{U}^{\mathcal{O}} \leftarrow \operatorname{COMPUTEUNCERTAINAXIOMS}(\mathcal{O})$;
3. $\mathcal{O}^\sigma \leftarrow \operatorname{COMPUTECOMPLETION}(\mathcal{O}, \mathcal{U}^{\mathcal{O}})$;
4. foreach TBox-complete ontology $\mathcal{O}' \in \mathcal{O}^\sigma$ do
5.  if $\mathcal{O}' \models_{FOL} Q'$ then
6.      continue;
7.  else
8.      return false;
9. end
10 end
11 return true;
Chapter 9

Complexity analysis

In this chapter we analyze the computational complexity for the problems of ontology satisfiability and of query answering that have been described in the previous chapters. We start by giving upper bounds in Section 9.1 then in Section 9.2 we illustrate two lower bounds for query answering in the most general settings showing that query answering in $\mathbf{Hi(OWL 2 QL)}$ is coNP-complete with reference to ontology complexity and $\Pi_2^P$-complete with reference to combined complexity.

9.1 Upper bounds

In this section we analyze the computational complexity of the algorithms that have been described in Sections 7.2, 7.3, 8.1, and 8.2 to solve the problems of ontology satisfiability, answering instance queries, answering general queries over TBox-complete ontologies, and answering general queries over general ontologies.

9.1.1 Ontology satisfiability

As for ontology satisfiability we consider the Algorithm presented at the end of Section 7.2. Given an ontology $O$, such an algorithm performs four main steps:

1. It computes the unsatisfiability query $Q_O$ for $O$;
2. It applies functions $\sigma$ and $\tau$, both described at the end of Section 7.1 over $O$ as to obtain an ontology $O'$ that does not contain negative axioms and where data properties, datatypes and literals are suitably replaced by object properties, classes and individuals, respectively;
3. It applies function $\tau_o$ over $Q'_O$ as to obtain a query $Q'$ where, coherently with the substitutions performed by $\tau$ over $O$, data properties, datatypes and literals occurring in $Q'_O$ are suitably replaced by object properties, classes and individuals, respectively;
4. It evaluates $Q'$ over $O'$ under the standard FOL semantics.

Note that functions $\sigma$ and $\tau$ are both linear with respect to the size of $O$, and that the evaluation of $Q'$ over $O'$ can be delegated to any standard OWL 2 QL reasoner. Therefore we can state the following theorem providing an upper bound for the ontology satisfiability problem.
Theorem 65 The problem of checking satisfiability of a Hi (OWL 2 QL) ontology is in \( AC^0 \) w.r.t. ABox complexity and in \( PTIME \) w.r.t. ontology complexity.

9.1.2 Answering instance queries

As for answering instance queries we consider Algorithm 2 presented at the end of Section 7.3. Given an ontology \( O \) and a boolean instance query \( Q \) over \( O \), such an algorithm performs two main steps:

1. It computes the metaground instance query \( MG(Q, O) \) as the union of all the metagroundings of \( Q \) with respect to \( O \);
2. It evaluates \( MG(Q, O) \) under the standard FOL semantics.

Note that the number of disjuncts of \( MG(Q, O) \), denoted by \(|MG(Q, O)|\), is bounded by the following function

\[
|MG(Q, O)| = \left| Exp_O^{Var_C}\right| \cdot \left| Exp_O^{Var_{OP}}\right| \cdot \left| V_O^{Var_{DP}}\right| \cdot \left| V_O^{Var_{DT}}\right|
\]

where \( Var_C \) (respectively \( Var_{OP}, Var_{DP}, \) and \( Var_{DT} \)) is the set containing all the variables occurring in class (respectively object property, data property, and datatype) position in \( Q \). Thus \( MG(Q, O) \) can be computed in polynomial time with respect to the size of \( O \) and, moreover, the evaluation of \( MG(Q, O) \) over \( O \) can be delegated to any standard OWL 2 QL reasoner. Therefore we can state the following theorem providing an upper bound for the problem of answering instance queries over general Hi (OWL 2 QL) ontologies.

Theorem 66 Let \( O \) be a satisfiable ontology. Answering instance queries over \( O \) is in \( AC^0 \) w.r.t. ABox complexity, \( PTIME \) w.r.t. ontology complexity, and NP-complete w.r.t. combined complexity.

9.1.3 Answering queries over TBox-complete ontologies

As for answering queries over ontologies that are TBox-complete, we consider Algorithm 3 presented at the end of Section 8.1. Given an ontology \( O \) and a boolean query \( Q \) over \( O \), such an algorithm performs two main steps:

1. It computes the metaground query \( MG(Q, O) \) as the union of all the metagroundings of \( Q \) with respect to \( O \);
2. It evaluates \( MG(Q, O) \) under the standard FOL semantics.

We have shown that \( MG(Q, O) \) can be computed in polynomial time with respect to the size of \( O \) and, moreover, the evaluation of \( MG(Q, O) \) over \( O \) can be delegated to any standard OWL 2 QL reasoner. Therefore we can state the following theorem providing an upper bound for the problem of answering general queries over TBox-complete Hi (OWL 2 QL) ontologies.

Theorem 67 Let \( O \) be a satisfiable TBox-complete ontology. Query answering over \( O \) is \( AC^0 \) in data complexity, \( PTIME \) in ontology complexity, and NP-complete in combined complexity.
9.1.4 Answering queries over general ontologies

We conclude by analyzing Algorithm 4 presented at the end of Section 8.2 to solve the problem of query answering over general ontologies. Given an ontology \( O \) and a boolean query \( Q \) over \( O \), such an algorithm performs four main steps:

1. It computes the metaground query \( MG(Q, O) \) as the union of all the metagroundings of \( Q \) with respect to \( O \);
2. It computes the set \( U^O \), containing all the negative axioms that are uncertain for \( O \);
3. It computes the set \( O^\sigma \), containing all the completions of \( O \) with respect to \( U^O \);
4. For every TBox-complete ontology \( O' \) in \( O^\sigma \), it evaluates \( MG(Q, O) \) over \( O' \) under the standard FOL semantics.

By exploiting the results given in the previous sections, we can state the following theorem.

**Theorem 68** Let \( O \) be a satisfiable ontology. Query answering over \( O \) is \( \text{AC}^0 \) in data complexity, \( \text{coNP} \) in ontology complexity, and \( \Pi^p_2 \) in combined complexity.

**Proof.** By Theorem 67 we know that given a query \( Q \) and a TBox-complete ontology, the problem of checking whether \( O \models Q \) is \( \text{AC}^0 \) in data complexity. Clearly, since the computation of \( O^\sigma \) for every subset \( \sigma \) of \( U^O \) does not depend on the size of the data, the data complexity of query answering is \( \text{AC}^0 \) also for general ontologies. On the contrary, it is easy to see that the problem is \( \text{coNP} \) in ontology complexity. Indeed, \( U^O \) can be computed in exponential time in the size of the ontology, and the size of \( U^O \) is exponential in the size of the ontology. Also, \( O \models Q \) if and only if for every subset \( \sigma \) of \( U^O \), \( O^\sigma \models Q \), and by Theorem 67 we know that the latter is verifiable in \( \text{PTIME} \) in ontology complexity. By following the same line of reasoning, given that by Theorem 67 \( O^\sigma \models Q \) is verifiable in nondeterministic polynomial time w.r.t. combined complexity, we have that query answering is in \( \text{coNP} \) in combined complexity.

9.2 Lower bounds

In this section we illustrate two complexity lower bounds matching the upper bounds illustrated in the previous section. We start by proving that query answering is in \( \text{Hil} \text{(OWL 2 QL)} \) is \( \text{coNP} \)-hard w.r.t. ontology complexity.

Let \( \text{THREE-POS} \) be the class of conjunctive propositional formulas expressed in terms of a nonempty set of clauses, each one with three positive propositional letters. It is well-known that the following problem, called \( \text{ONE-IN-3-POS-SAT} \) is \( \text{NP} \)-complete given a \( \text{THREE-POS} \) formula \( F \), check whether there exists a \( \text{ONE-IN-3} \) model of \( F \), i.e., a propositional assignment that makes exactly one propositional letter true for each clause of \( F \).

Let \( F \) be a formula in the class \( \text{THREE-POS} \) and let \( Ax \) be one of the metapredicate that are used to build TBox axioms (i.e., \( Ax \) is one among \( \text{SubClassOf}, \text{DisjointClasses}, \text{SubObjectPropertyOf}, \text{etc.} \)). In the following we show that the problem of checking
whether there exists a ONE-IN-3 model of $F$, can be reduced to the problem of checking whether an ontology $O_{Ax}^F$ implies a query $Q_{Ax}^F$, where $Q_{Ax}^F$ contains, besides ABox atoms, only TBox atoms of the form $Ax$. We start by showing a reduction where all the TBox atoms contained in the query have the form $\text{DisjointClasses}$.

Let $F$ be a formula in the class THREE-POS, and let $\{1..k\}$ denote the clauses in $F$. From $F$, we build an ontology $O_{dc}$ and a query $Q_{dc}$ as follows. The alphabet $V_{O_{dc}}$ of $O_{dc}$ is such that:

- $V_{I_{dc}}^O$ contains the entities $d$, $p_1$, $p_2$, and $p_3$ and some entities representing the clauses of $F$ and propositional letters occurring in them, precisely (i) two entities $c_i$ and $f_i$ for each clause $i$, and (ii) one entity $t_{i,j}$ for each pair $i,j$ such that $i \in \{1, \ldots, k\}$, and $j \in \{1, 2, 3\}$;
- $V_{C_{dc}}^O$ contains the entities $B$, $D$, and one entity $A_i$ for each propositional letter;
- $V_{OP_{dc}}^O$ contains the entities $R_1$, $R_2$, $R_3$, $R_4$, $P_1$, $P_2$, and $P_3$.

We now describe the axioms of $O_{dc}$ and the query $Q_{dc}$. In what follows, we denote by $A_{i,j}$ the class corresponding to the $j$-th literal in the $i$-th clause, where $i \in \{1, \ldots, k\}$, and $j \in \{1, 2, 3\}$.

The ontology $O_{dc}$ is formed by the following axioms

- $\text{DisjointClasses} (B D)$,
- $\text{ClassAssertion} (D d)$,
- $\text{ObjectPropertyAssertion} (R_2 t_0 t_0)$,
- $\text{ObjectPropertyAssertion} (R_3 t_0 D)$,
- $\text{ObjectPropertyAssertion} (R_4 t_0 D)$,
- $\text{ObjectPropertyAssertion} (R_1 t_0 c_1)$,
- $\text{ObjectPropertyAssertion} (R_1 t_0 c_2)$,
- $\ldots$,
- $\text{ObjectPropertyAssertion} (R_1 t_0 c_k)$,
- $\text{ObjectPropertyAssertion} (P_1 c_1 A_{1,1})$, $\text{ObjectPropertyAssertion} (P_2 c_1 A_{1,2})$, $\text{ObjectPropertyAssertion} (P_3 c_1 A_{1,2})$,
- $\text{ObjectPropertyAssertion} (P_1 c_2 A_{2,1})$, $\text{ObjectPropertyAssertion} (P_2 c_2 A_{2,2})$, $\text{ObjectPropertyAssertion} (P_3 c_2 A_{2,2})$,
- $\ldots$,
- $\text{ObjectPropertyAssertion} (P_1 c_k A_{k,1})$, $\text{ObjectPropertyAssertion} (P_2 c_k A_{k,2})$, $\text{ObjectPropertyAssertion} (P_3 c_k A_{k,2})$. 

• ObjectPropertyAssertion \( (P_1 f_1 B) \), ObjectPropertyAssertion \( (P_2 f_1 B) \), ObjectPropertyAssertion \( (P_3 f_1 B) \),
• ObjectPropertyAssertion \( (P_1 f_2 B) \), ObjectPropertyAssertion \( (P_2 f_2 B) \), ObjectPropertyAssertion \( (P_3 f_2 B) \),
• \ldots,
• ObjectPropertyAssertion \( (P_1 f_k B) \), ObjectPropertyAssertion \( (P_2 f_k B) \), ObjectPropertyAssertion \( (P_3 f_k B) \),
• ObjectPropertyAssertion \( (R_1 t_{1,1} f_1) \), ObjectPropertyAssertion \( (R_2 t_{1,1} p_1) \), ObjectPropertyAssertion \( (R_3 t_{1,1} A_{1,1}) \), ObjectPropertyAssertion \( (R_4 t_{1,1} A_{1,2}) \),
• ObjectPropertyAssertion \( (R_1 t_{1,2} f_1) \), ObjectPropertyAssertion \( (R_2 t_{1,2} p_2) \), ObjectPropertyAssertion \( (R_3 t_{1,2} A_{1,1}) \), ObjectPropertyAssertion \( (R_4 t_{1,2} A_{1,3}) \),
• ObjectPropertyAssertion \( (R_1 t_{1,3} f_1) \), ObjectPropertyAssertion \( (R_2 t_{1,3} p_3) \), ObjectPropertyAssertion \( (R_3 t_{1,3} A_{1,2}) \), ObjectPropertyAssertion \( (R_4 t_{1,3} A_{1,3}) \),
• \ldots,
• ObjectPropertyAssertion \( (R_1 t_{k,1} f_k) \), ObjectPropertyAssertion \( (R_2 t_{k,1} p_1) \), ObjectPropertyAssertion \( (R_3 t_{k,1} A_{k,1}) \), ObjectPropertyAssertion \( (R_4 t_{k,1} A_{k,2}) \),
• ObjectPropertyAssertion \( (R_1 t_{k,2} f_k) \), ObjectPropertyAssertion \( (R_2 t_{k,2} p_2) \), ObjectPropertyAssertion \( (R_3 t_{k,2} A_{k,1}) \), ObjectPropertyAssertion \( (R_4 t_{k,2} A_{k,3}) \),
• ObjectPropertyAssertion \( (R_1 t_{k,3} f_k) \), ObjectPropertyAssertion \( (R_2 t_{k,3} p_3) \), ObjectPropertyAssertion \( (R_3 t_{k,3} A_{k,2}) \), ObjectPropertyAssertion \( (R_4 t_{k,3} A_{k,3}) \).

and \( Q^F_{dc} \) is the following boolean conjunctive query

\[
\text{ASK} \{ \text{DisjointClasses (}$x_1$D$) \cdot \text{DisjointClasses (}$x_2$D$) \cdot \text{DisjointClasses (}$x_3$D$) \cdot \text{ObjectPropertyAssertion (}$P_1$sw}$x_1$) \cdot \text{ObjectPropertyAssertion (}$P_2$sw}$x_2$) \cdot \text{ObjectPropertyAssertion (}$P_3$sw}$x_3$) \cdot \text{ObjectPropertyAssertion (}$R_1$sv}$w$) \cdot \text{ObjectPropertyAssertion (}$R_2$sv}$p$) \cdot \text{ObjectPropertyAssertion (}$R_3$sv}$z$) \cdot \text{ObjectPropertyAssertion (}$R_4$sv}$y$) \cdot \text{ClassAssertion (}$z_1$y$) \}
\]

Note that the size of \( O^F_{dc} \) is polynomial with respect to the size of \( F \), and the size of \( Q^F_{dc} \) is constant.

**Theorem 69** There exists a ONE-IN-3 model of \( F \) if and only if \( O^F_{dc} \not\models Q^F_{dc} \).

**Proof.** By Theorem 63 we know that the certain answer to \( Q^F_{dc} \) with respect to \( O^F_{dc} \) is false if and only if there is a complete ontology \( O \) derived from \( O^F_{dc} \) such that for each metagrounding \( Q \) of \( O^F_{dc} \), the certain answer to \( Q \) with respect to \( O \) is false.

\[ \Rightarrow \text{Suppose that there is a complete ontology } O \text{ derived from } O^F_{dc} \text{ such that for each metagrounding } Q \text{ of } O^F_{dc}, \text{ the certain answer to } Q \text{ with respect to } O \text{ is false. We first show that, for each } i \in \{1..k\}, \text{ we have exactly one } A_{i,j} \text{ such that } O \not\models \text{DisjointClasses}(A_{i,j} D). \]

Indeed, consider any \( c_i \).
If every \( A_{i,j} \) is such that \( \mathcal{O} \models \text{DisjointClasses}(A_{i,j} \ D) \), then it is easy to see that the certain answer of the following metagrounding of \( Q^F_{dc} \) with respect to \( \mathcal{O} \) is true.

\[
\begin{align*}
\text{ObjectPropertyAssertion}(P_1 c_i A_{i,1}). \quad \text{ObjectPropertyAssertion}(P_2 c_i A_{i,2}). \\
\text{ObjectPropertyAssertion}(P_3 c_i A_{i,3}). \quad \text{ObjectPropertyAssertion}(R_1 t_0 c_i). \\
\text{ObjectPropertyAssertion}(R_2 t_0 t_0). \quad \text{ObjectPropertyAssertion}(R_3 t_0 D). \\
\text{DisjointClasses}(A_{i,1} D). \quad \text{DisjointClasses}(A_{i,2} D). \\
\text{ClassAssertion}(D d). \quad \text{ClassAssertion}(D d).
\end{align*}
\]

If there is exactly one \( A_{i,j} \) such that \( \mathcal{O} \models \text{DisjointClasses}(A_{i,j} \ D) \), then it is easy to see that the certain answer of the following metagrounding of \( Q^F_{dc} \) with respect to \( \mathcal{O} \) is true.

\[
\begin{align*}
\text{ObjectPropertyAssertion}(P_1 f_i B). \quad \text{ObjectPropertyAssertion}(P_3 f_i B). \\
\text{ObjectPropertyAssertion}(P_3 f_i A_{i,j}). \quad \text{ObjectPropertyAssertion}(R_1 t_{i,j} f_i). \\
\text{ObjectPropertyAssertion}(R_2 t_{i,j} p_j). \quad \text{ObjectPropertyAssertion}(R_3 t_{i,j} A_{i,k}). \\
\text{ObjectPropertyAssertion}(R_4 t_{i,j} A_{i,k}). \quad \text{DisjointClasses}(B D). \\
\text{DisjointClasses}(B D). \quad \text{DisjointClasses}(A_{i,j} D). \\
\text{ClassAssertion}(A_{i,k} y_1). \quad \text{ClassAssertion}(A_{i,k} y_2).
\end{align*}
\]

It follows that, for each \( c_i \), we have exactly one \( A_{i,q} \) such that \( \mathcal{O} \not\models \text{DisjointClasses}(A_{i,q} \ D) \). It is possible to show that the propositional interpretation that assigns true to the propositional letters corresponding to such \( A_{i,q} \)'s is a model of \( F \) where, for each clause, exactly one literal is true.

\[ \Leftrightarrow \text{ Suppose that there exists a model } M \text{ of } F \text{ where, for each clause, exactly one literal is true. We show that there is a complete ontology derived from } \mathcal{O}^F_{dc} \text{ such that for each metagrounding } q \text{ of } Q^F_{dc}, \text{ the certain answer to } q \text{ with respect to } \mathcal{O} \text{ is false.} \]

Let \( \mathcal{O} \) be the complete ontology obtained from \( \mathcal{O}^F_{dc} \) where, for each \( c_i \), all the assertions \( \text{DisjointClasses}(A_{i,j} \ D) \) are in \( \mathcal{O} \) except for the \( A_{i,j} \) corresponding to the literal which is true in \( M \). We show that, for each metagrounding \( q \) of \( Q^F_{dc} \), the certain answer to \( q \) with respect to \( \mathcal{O} \) is false. It is sufficient to note that

\[ \begin{align*}
\text{if } q \text{ has } \$w = c_i \text{ for some } i, \text{ then the atom } \text{DisjointClasses}(A_{i,j} \ D) \text{ is false in } \mathcal{O}; \\
\text{if } q \text{ has } \$w = f_i \text{ for some } i, \text{ then } \$z_1 = A_{i,q}, \text{ and } \$z_2 = A_{i,t} \text{ for } q, t \neq j, \text{ and there is a model of } \mathcal{O} \text{ where the conjunction of the atoms } \text{ClassAssertion}(A_{i,q} \ y_1) \text{ and } \text{ClassAssertion}(A_{i,t} \ y_2) \text{ are false.}
\end{align*} \]

The above theorem shows that we can reduce the ONE-IN-3-POS-SAT problem to the problem of checking whether or not an \( \text{Hi} \) (OWL 2 QL) ontology entails a conjunctive query containing, besides ABox atoms, only TBox atoms of the form \( \text{DisjointClasses} \). We next shows two reductions where the considered conjunctive query contains atoms of the form \( \text{SubObjectPropertyOf} \) and \( \text{IrreflexiveObjectProperty} \) respectively.
Let $F$ be a formula in the class THREE-POS, and let $\{1..k\}$ denote the clauses in $F$. From $F$, we build an ontology $O_{sop}^F$, and a query $Q_{sop}^F$ as follows. The alphabet $V_{O_{sop}^F}$ of $O_{sop}^F$ is such that:

- $V_{O_{sop}^F}$ contains the entities $d_1$, $d_2$, $p_1$, $p_2$, and $p_3$ and some entities representing the clauses of $F$ and propositional letters occurring in them, precisely (i) two entities $c_i$ and $f_i$ for each clause $i$, and (ii) one entity $t_{i,j}$ for each pair $i$, $j$ such that $i \in \{1, \ldots, k\}$, and $j \in \{1,2,3\}$;

- $V_{O_{sop}^F}$ contains the entities $B$, $D$, $R_1$, $R_2$, $R_3$, $R_4$, $P_1$, $P_2$, $P_3$, and one entity $A_i$ for each propositional letter;

We now describe the axioms of $O_{sop}^F$ and the query $Q_{sop}^F$. In what follows, we denote by $A_{i,j}$ the object property corresponding to the $j$-th literal in the $i$-th clause, where $i \in \{1, \ldots, k\}$, and $j \in \{1,2,3\}$.

The ontology $O_{sop}^F$ is formed by the following axioms:

- SubObjectPropertyOf ($B$ $D$),
- ObjectPropertyAssertion ($D$ $d_1$ $d_2$),
- ObjectPropertyAssertion ($R_2$ $t_0$ $t_0$),
- ObjectPropertyAssertion ($R_3$ $t_0$ $D$),
- ObjectPropertyAssertion ($R_4$ $t_0$ $D$),
- ObjectPropertyAssertion ($R_1$ $t_0$ $c_1$),
- ObjectPropertyAssertion ($R_1$ $t_0$ $c_2$),
- ....
- ObjectPropertyAssertion ($R_1$ $t_0$ $c_k$),
- ObjectPropertyAssertion ($P_1$ $c_1$ $A_{1,1}$), ObjectPropertyAssertion ($P_2$ $c_1$ $A_{1,2}$), ObjectPropertyAssertion ($P_3$ $c_1$ $A_{1,2}$),
- ObjectPropertyAssertion ($P_1$ $c_2$ $A_{2,1}$), ObjectPropertyAssertion ($P_2$ $c_2$ $A_{2,2}$), ObjectPropertyAssertion ($P_3$ $c_2$ $A_{2,2}$),
- ....
- ObjectPropertyAssertion ($P_1$ $c_k$ $A_{k,1}$), ObjectPropertyAssertion ($P_2$ $c_k$ $A_{k,2}$), ObjectPropertyAssertion ($P_3$ $c_k$ $A_{k,2}$),
- ObjectPropertyAssertion ($P_1$ $f_1$ $B$), ObjectPropertyAssertion ($P_2$ $f_1$ $B$), ObjectPropertyAssertion ($P_3$ $f_1$ $B$),
- ObjectPropertyAssertion ($P_1$ $f_2$ $B$), ObjectPropertyAssertion ($P_2$ $f_2$ $B$), ObjectPropertyAssertion ($P_3$ $f_2$ $B$),
... 

- ObjectPropertyAssertion \((P_1 f_k B)\), ObjectPropertyAssertion \((P_2 f_k B)\), ObjectPropertyAssertion \((P_3 f_k B)\).
- ObjectPropertyAssertion \((R_1 t_{i,1} f_1)\), ObjectPropertyAssertion \((R_2 t_{i,1} p_1)\), ObjectPropertyAssertion \((R_3 t_{i,1} A_{i,1})\), ObjectPropertyAssertion \((R_4 t_{i,1} A_{i,2})\).
- ObjectPropertyAssertion \((R_1 t_{i,2} f_1)\), ObjectPropertyAssertion \((R_2 t_{i,2} p_2)\), ObjectPropertyAssertion \((R_3 t_{i,2} A_{i,1})\), ObjectPropertyAssertion \((R_4 t_{i,2} A_{i,3})\).
- ObjectPropertyAssertion \((R_1 t_{i,3} f_1)\), ObjectPropertyAssertion \((R_2 t_{i,3} p_3)\), ObjectPropertyAssertion \((R_3 t_{i,3} A_{i,2})\), ObjectPropertyAssertion \((R_4 t_{i,3} A_{i,3})\).
- ...

- ObjectPropertyAssertion \((R_1 t_{i,j} f_k)\), ObjectPropertyAssertion \((R_2 t_{i,j} p_1)\), ObjectPropertyAssertion \((R_3 t_{i,j} A_{i,k})\), ObjectPropertyAssertion \((R_4 t_{i,j} A_{i,k})\).
- ObjectPropertyAssertion \((R_1 t_{i,j} f_k)\), ObjectPropertyAssertion \((R_2 t_{i,j} p_2)\), ObjectPropertyAssertion \((R_3 t_{i,j} A_{i,k})\), ObjectPropertyAssertion \((R_4 t_{i,j} A_{i,k})\).
- ObjectPropertyAssertion \((R_1 t_{i,j} f_k)\), ObjectPropertyAssertion \((R_2 t_{i,j} p_3)\), ObjectPropertyAssertion \((R_3 t_{i,j} A_{i,k})\), ObjectPropertyAssertion \((R_4 t_{i,j} A_{i,k})\).

and \(Q_{\text{sop}}^F\) is the following boolean conjunctive query

\[
\text{ASK} \{ \text{ObjectPropertyAssertion}(P_1 \$w \$x_1), \text{ObjectPropertyAssertion}(P_2 \$w \$x_2), \text{ObjectPropertyAssertion}(P_3 \$w \$x_3), \text{ObjectPropertyAssertion}(R_1 \$v \$w), \text{ObjectPropertyAssertion}(R_2 \$v \$p), \text{ObjectPropertyAssertion}(R_3 \$v \$s_1), \text{ObjectPropertyAssertion}(R_4 \$v \$s_2), \text{ObjectPropertyAssertion}(s_{z_1} \$y_1 \$y_2), \text{SubObjectPropertyOf}(s_{z_2} \$y_3 \$y_4), \text{SubObjectPropertyOf}(s_{x_2} D). \text{SubObjectPropertyOf}(s_{x_3} D). \}
\]

Note that the size of \(O_{\text{sop}}^F\) is polynomial with respect to the size of \(F\), and the size of \(Q_{\text{sop}}^F\) is constant.

**Theorem 70** There exists a ONE-IN-3 model of \(F\) if and only if \(O_{\text{sop}}^F \not\models Q_{\text{sop}}^F\).

**Proof.** By Theorem 63, we know that the certain answer to \(Q_{\text{sop}}^F\) with respect to \(O_{\text{sop}}^F\) is false if and only if there is a complete ontology \(O\) derived from \(O_{\text{sop}}^F\) such that for each metagrounding \(Q\) of \(Q_{\text{sop}}^F\), the certain answer to \(Q\) with respect to \(O\) is false.

\(\Rightarrow\) Suppose that there is a complete ontology \(O\) derived from \(O_{\text{sop}}^F\), such that for each metagrounding \(Q\) of \(Q_{\text{sop}}^F\), the certain answer to \(Q\) with respect to \(O\) is false. We first show that, for each \(i \in \{1..k\}\), we have exactly one \(A_{i,j}\) such that \(O \not\models \text{SubObjectPropertyOf}(A_{i,j} D)\). Indeed, consider any \(c_i\).

- If every \(A_{i,j}\) is such that \(O \models \text{SubObjectPropertyOf}(A_{i,j} D)\), then it is easy to see that the certain answer of the following metagrounding of \(Q_{\text{sop}}^F\) with respect to \(O\) is true.
• If there is exactly one \( A_{i,j} \) such that \( \mathcal{O} \models \text{SubObjectPropertyOf}(A_{i,j} D) \), then it is easy to see that the certain answer of the following metagrounding of \( Q^F_{sop} \) with respect to \( \mathcal{O} \) is true.

\[
\begin{align*}
\text{ObjectPropertyAssertion}(P_1 c_i A_{i,1}) &. \text{ObjectPropertyAssertion}(P_2 c_i A_{i,2}). \\
\text{ObjectPropertyAssertion}(P_3 c_i A_{i,3}) &. \text{ObjectPropertyAssertion}(R_1 t_0 c_i). \\
\text{ObjectPropertyAssertion}(R_2 t_0 t_0) &. \text{ObjectPropertyAssertion}(R_3 t_0 D). \\
\text{ObjectPropertyAssertion}(R_4 t_0 D). &. \text{SubObjectPropertyOf}(A_{i,3} D). \\
\text{SubObjectPropertyOf}(A_{i,1} D). &. \text{SubObjectPropertyOf}(A_{i,2} D). \\
\text{ObjectPropertyAssertion}(D d_1 d_2). &. \text{ObjectPropertyAssertion}(D d_1 d_2). 
\end{align*}
\]

It follows that, for each \( c_i \), we have exactly one \( A_{i,q} \) such that \( \mathcal{O} \not\models \text{DisjointClasses}(A_{i,q} D) \). It is possible to show that the propositional interpretation that assigns true to the propositional letters corresponding to such \( A_{i,q} \)'s is a model of \( F \) where, for each clause, exactly one literal is true.

\[\iff\] Suppose that there exists a model \( M \) of \( F \) where, for each clause, exactly one literal is true. We show that there is a complete ontology derived from \( \mathcal{O}^{F}_{sop} \) such that for each metagrounding \( q \) of \( Q^F_{sop} \), the certain answer to \( q \) with respect to \( \mathcal{O} \) is false.

Let \( \mathcal{O} \) be the complete ontology obtained from \( \mathcal{O}^{F}_{sop} \) where, for each \( c_i \), all the assertions \( \text{SubObjectPropertyOf}(A_{i,j} D) \) are in \( \mathcal{O} \) except for the \( A_{i,j} \) corresponding to the literal which is true in \( M \). We show that, for each metagrounding \( q \) of \( Q^F_{sop} \), the certain answer to \( q \) with respect to \( \mathcal{O} \) is false. It is sufficient to note that

- if \( q \) has \( \$w = c_i \) for some \( i \), then the atom \( \text{SubObjectPropertyOf}(A_{i,j} D) \) is false in \( \mathcal{O} \);
- if \( q \) has \( \$w = f_i \) for some \( i \), then \( \$z_1 = A_{i,q} \), and \( \$z_2 = A_{i,j} \) for \( q, t \neq j \), and there is a model of \( \mathcal{O} \) where the conjunction of the atoms \( \text{ObjectPropertyAssertion}(A_{i,q} \$y_1 \$y_2) \) and \( \text{ObjectPropertyAssertion}(A_{i,t} \$y_3 \$y_4) \) are false.

\[\boxdot\]

Next we show the reduction where, besides ABox atoms, only TBox atoms of the form \( \text{IrreflexiveObjectProperty} \) occur in the considered conjunctive query. Let \( F \) be a formula in the class THREE-POS, and let \( \{1..k\} \) denote the clauses in \( F \). From \( F \), we build an ontology \( \mathcal{O}^F_{irr} \), and a query \( Q^F_{irr} \) as follows. The alphabet \( V_{\mathcal{O}^F_{irr}} \) of \( \mathcal{O}^F_{irr} \) is such that:

- \( V^{\mathcal{O}^F_{irr}}_F \) contains the entities \( d, p_1, p_2, \) and \( p_3 \) and some entities representing the clauses of \( F \) and propositional letters occurring in them, precisely (i) two entities \( c_i \) and \( f_i \) for each clause \( i \), and (ii) one entity \( t_{i,j} \) for each pair \( i,j \) such that \( i \in \{1, \ldots, k\} \), and \( j \in \{1, 2, 3\} \);
9. Complexity analysis

- $V_{OF}^{irr}$ contains the entities $B$, $D$, $R_1$, $R_2$, $R_3$, $R_4$, $P_1$, $P_2$, $P_3$, and one entity $A_i$ for each propositional letter;

We now describe the axioms of $O_{irr}^F$ and the query $Q_{irr}^F$. In what follows, we denote by $A_{i,j}$ the object property corresponding to the $j$-th literal in the $i$-th clause, where $i \in \{1, \ldots, k\}$, and $j \in \{1, 2, 3\}$.

The ontology $O_{irr}^F$ is formed by the following axioms

- IrreflexiveObjectProperty ($B$),
- ObjectPropertyAssertion ($D$ $d$ $d$),
- ObjectPropertyAssertion ($R_2$ $t_0$ $t_0$),
- ObjectPropertyAssertion ($R_3$ $t_0$ $D$),
- ObjectPropertyAssertion ($R_4$ $t_0$ $D$),
- ObjectPropertyAssertion ($R_1$ $t_0$ $c_1$),
- ObjectPropertyAssertion ($R_1$ $t_0$ $c_2$),
- ....,
- ObjectPropertyAssertion ($R_1$ $t_0$ $c_k$),
- ObjectPropertyAssertion ($P_1$ $c_1$ $A_{1,1}$), ObjectPropertyAssertion ($P_2$ $c_1$ $A_{1,2}$), ObjectPropertyAssertion ($P_3$ $c_1$ $A_{1,2}$),
- ObjectPropertyAssertion ($P_1$ $c_2$ $A_{2,1}$), ObjectPropertyAssertion ($P_2$ $c_2$ $A_{2,2}$), ObjectPropertyAssertion ($P_3$ $c_2$ $A_{2,2}$),
- ....,
- ObjectPropertyAssertion ($P_1$ $c_k$ $A_{k,1}$), ObjectPropertyAssertion ($P_2$ $c_k$ $A_{k,2}$), ObjectPropertyAssertion ($P_3$ $c_k$ $A_{k,2}$),
- ObjectPropertyAssertion ($P_1$ $f_1$ $B$), ObjectPropertyAssertion ($P_2$ $f_1$ $B$), ObjectPropertyAssertion ($P_3$ $f_1$ $B$),
- ObjectPropertyAssertion ($P_1$ $f_2$ $B$), ObjectPropertyAssertion ($P_2$ $f_2$ $B$), ObjectPropertyAssertion ($P_3$ $f_2$ $B$),
- ....,
- ObjectPropertyAssertion ($P_1$ $f_k$ $B$), ObjectPropertyAssertion ($P_2$ $f_k$ $B$), ObjectPropertyAssertion ($P_3$ $f_k$ $B$),
- ObjectPropertyAssertion ($R_1$ $t_{1,1}$ $f_1$), ObjectPropertyAssertion ($R_2$ $t_{1,1}$ $p_1$), ObjectPropertyAssertion ($R_3$ $t_{1,1}$ $A_{1,1}$), ObjectPropertyAssertion ($R_4$ $t_{1,1}$ $A_{1,2}$),
- ObjectPropertyAssertion ($R_1$ $t_{1,2}$ $f_1$), ObjectPropertyAssertion ($R_2$ $t_{1,2}$ $p_2$), ObjectPropertyAssertion ($R_3$ $t_{1,2}$ $A_{1,1}$), ObjectPropertyAssertion ($R_4$ $t_{1,2}$ $A_{1,3}$),
9.2 Lower bounds

• ObjectPropertyAssertion (R₁ t₁₃ f₁), ObjectPropertyAssertion (R₂ t₁₃ p₁), ObjectPropertyAssertion (R₃ t₁₃ A₁₂), ObjectPropertyAssertion (R₄ t₁₃ A₁₃),
• ...
• ObjectPropertyAssertion (R₁ tₖ₁ fₖ), ObjectPropertyAssertion (R₂ tₖ₁ p₁), ObjectPropertyAssertion (R₃ tₖ₁ Aₖ₁), ObjectPropertyAssertion (R₄ tₖ₁ Aₖ₂),
• ObjectPropertyAssertion (R₁ tₖ₂ fₖ), ObjectPropertyAssertion (R₂ tₖ₂ p₂), ObjectPropertyAssertion (R₃ tₖ₂ Aₖ₁), ObjectPropertyAssertion (R₄ tₖ₂ Aₖ₃),
• ObjectPropertyAssertion (R₁ tₖ₃ fₖ), ObjectPropertyAssertion (R₂ tₖ₃ p₃), ObjectPropertyAssertion (R₃ tₖ₃ Aₖ₂), ObjectPropertyAssertion (R₄ tₖ₃ Aₖ₃).

and \(Q^{F}_{irr}\) is the following boolean conjunctive query

\[
\text{ASK} \{ \text{ObjectPropertyAssertion}(P₁ \text{ w } x₁). \text{ObjectPropertyAssertion}(P₂ \text{ w } x₂). \text{ObjectPropertyAssertion}(P₃ \text{ w } x₃). \text{ObjectPropertyAssertion}(R₁ \text{ w } y₁). \text{ObjectPropertyAssertion}(R₂ \text{ w } y₂). \text{ObjectPropertyAssertion}(R₃ \text{ w } z₁). \text{ObjectPropertyAssertion}(R₄ \text{ w } z₂). \text{IrreflexiveObjectProperty}(x₁). \text{IrreflexiveObjectProperty}(x₂). \text{IrreflexiveObjectProperty}(x₃). \}
\]

Note that the size of \(O^{F}_{irr}\) is polynomial with respect to the size of \(F\), and the size of \(Q^{F}_{irr}\) is constant.

**Theorem 71** There exists a ONE-IN-3 model of \(F\) if and only if \(O^{F}_{irr} \not\models Q^{F}_{irr}\).

**Proof.** By Theorem 63 we know that the certain answer to \(Q^{F}_{irr}\) with respect to \(O^{F}_{irr}\) is false if and only if there is a complete ontology \(O\) derived from \(O^{F}_{irr}\) such that for each metagrounding \(Q\) of \(O^{F}_{irr}\), the certain answer to \(Q\) with respect to \(O\) is false.

\[\Rightarrow\] Suppose that there is a complete ontology \(O\) derived from \(O^{F}_{irr}\) such that for each metagrounding \(Q\) of \(O^{F}_{irr}\), the certain answer to \(Q\) with respect to \(O\) is false. We first show that, for each \(i \in \{1..k\}\), we have exactly one \(A_{i,j}\) such that \(O \not\models \text{IrreflexiveObjectProperty}(A_{i,j})\). Indeed, consider any \(c_i\).

• If every \(A_{i,j}\) is such that \(O \models \text{IrreflexiveObjectProperty}(A_{i,j} D)\), then it is easy to see that the certain answer of the following metagrounding of \(Q^{F}_{irr}\) with respect to \(O\) is true.

\[
\begin{align*}
\text{ObjectPropertyAssertion}(P₁ cᵢ Aᵢ₁). & \quad \text{ObjectPropertyAssertion}(P₂ cᵢ Aᵢ₂). \\
\text{ObjectPropertyAssertion}(P₃ cᵢ Aᵢ₃). & \quad \text{ObjectPropertyAssertion}(R₁ t₀ cᵢ). \\
\text{ObjectPropertyAssertion}(R₂ t₀ t₀). & \quad \text{ObjectPropertyAssertion}(R₃ t₀ D). \\
\text{ObjectPropertyAssertion}(R₄ t₀ D). & \quad \text{IrreflexiveObjectProperty}(Aᵢ₃). \\
\text{IrreflexiveObjectProperty}(Aᵢ₁). & \quad \text{IrreflexiveObjectProperty}(Aᵢ₂). \\
\text{ObjectPropertyAssertion}(D d D). & \quad \text{ObjectPropertyAssertion}(D d D)
\end{align*}
\]
• If there is exactly one $A_{i,j}$ such that $\mathcal{O} \models \text{SubObjectPropertyOf}(A_{i,j}, D)$, then it is easy to see that the certain answer of the following metagrounding of $Q^F_{irr}$ with respect to $\mathcal{O}$ is true.

\begin{align*}
\text{ObjectPropertyAssertion}(P_1 f_i B) \land \text{ObjectPropertyAssertion}(P_2 f_i B) \land \text{ObjectPropertyAssertion}(P_3 f_i A_{i,j}) \land \text{ObjectPropertyAssertion}(R_1 t_{i,j} f_i) \land \text{ObjectPropertyAssertion}(R_2 t_{i,j} p_j) \land \text{ObjectPropertyAssertion}(R_3 t_{i,j} A_{i,h}) \land \text{IrreflexiveObjectProperty}(B) \land \text{IrreflexiveObjectProperty}(A_{i,j}) \land \text{ObjectPropertyAssertion}(A_{i,h}) \land \text{ObjectPropertyAssertion}(\exists y_1 \exists y_1) \land \text{ObjectPropertyAssertion}(\exists y_2 \exists y_2)
\end{align*}

It follows that, for each $c_i$, we have exactly one $A_{i,q}$ such that $\mathcal{O} \not\models \text{IrreflexiveObjectProperty}(A_{i,q})$. It is possible to show that the propositional interpretation that assigns true to the propositional letters corresponding to such $A_{i,q}$’s is a model of $F$ where, for each clause, exactly one literal is true.

Suppose that there exists a model $M$ of $F$ where, for each clause, exactly one literal is true. We show that there is a complete ontology derived from $\mathcal{O}$ where, for each clause, exactly one literal is true.

Let $\mathcal{O}$ be the complete ontology obtained from $\mathcal{O}_{irr}$ where, for each $c_i$, all the assertions $\text{IrreflexiveObjectProperty}(A_{i,j})$ are in $\mathcal{O}$ except for the $A_{i,j}$ corresponding to the literal which is true in $M$. We show that, for each metagrounding $q$ of $Q^F_{irr}$, the certain answer to $q$ with respect to $\mathcal{O}$ is false. It is sufficient to note that

• if $q$ has $\exists w = c_i$, then the atom $\text{IrreflexiveObjectProperty}(A_{i,j})$ is false in $\mathcal{O}$;

• if $q$ has $\exists w = f_i$, then $\exists z_1 = A_{i,q}$, and $\exists z_2 = A_{i,t}$ for $q, t \neq j$, and there is a model of $\mathcal{O}$ where the conjunction of the atoms $\text{ObjectPropertyAssertion}(A_{i,q} \exists y_1 \exists y_1)$ and $\text{ObjectPropertyAssertion}(A_{i,t} \exists y_2 \exists y_2)$ are false.

By Theorems 69, 70 and 71, it follows that query answering in $\Pi_2$-hard in ontology complexity. By combining this result with Theorem 68, we can state the following corollary.

**Corollary 72** Let $\mathcal{O}$ be a satisfiable ontology. Query answering over $\mathcal{O}$ is coNP-complete in ontology complexity.

Next, we prove that query answering in $\Pi_2$-hard w.r.t. combined complexity. Let us consider the problem of checking the satisfiability of a 2-QBF, i.e., a Quantified Boolean Formula of the form $\forall x_1, \ldots, x_n \exists y_1, \ldots, y_m c_1 \land \cdots \land c_k$, where each $c_i$ is a clause with exactly three literals on the propositional letters $x_1, \ldots, x_n, y_1, \ldots, y_m$.

Given a 2-QBF $F$, we obtain a new formula $F'$ as follows

1. we introduce a letter $\bar{p}$ for each letter $p$ of $F$;
2. we substitute every negative literal $\neg p$ in $F$ with $\bar{p}$;
3. we add every \( \bar{p} \) to the list of existentially quantified variable;

4. we add to \( F \) suitable clauses (called auxiliary clauses) to sanction that exactly one between \( p, \bar{p} \) is true.

It is easy to see that \( F \) is satisfiable if and only if \( F' \) is satisfiable, and that the size of \( F' \) is linear with respect to the size of \( F \). Starting from \( F' \), we define an Hi (\( \text{OWL 2 QL} \)) ontology \( \mathcal{O}_F \) as follows. The alphabet of \( \mathcal{O}_F \), is such that \( E, D \) are classes, we have one class for each universally quantified variable, we have object properties \( In, Has, Val \), and all other elements are individuals. The axioms of \( \mathcal{O}_F \) are as follows.

- the axiom \( \text{DisjointClasses}(D D) \) is in \( \mathcal{O}_F' \);
- the axiom \( E(a) \) is in \( \mathcal{O}_F' \);
- for \( i \in \{1, \ldots, k\} \) and \( j_1, j_2, j_3 \in \{1, 0\} \) such that at least one of the \( j_h \)'s is equal to 1, the axiom \( \text{ObjectPropertyAssertion}(In c_i t_{j_1,j_2,j_3}^i) \) is in \( \mathcal{O}_F' \);
- for each letter \( p \) of \( F' \), and for \( j_1, j_2 \in \{1, 0\} \) such that at least one of the \( j_h \)'s is equal to 1, the axiom \( \text{ObjectPropertyAssertion}(In v^p \tau_{j_1,j_2}^p) \) is in \( \mathcal{O}_F' \);
- for each axiom \( \text{ObjectPropertyAssertion}(In c_1 t_{j_1,j_2,j_3}^i) \), we add to \( \mathcal{O}_F' \) the axioms
  \[
  \text{ObjectPropertyAssertion}(Has t_{j_1,j_2,j_3}^i \tau(i, 1)),
  \text{ObjectPropertyAssertion}(Has t_{j_1,j_2,j_3}^i \tau(i, 2)), \text{ and }
  \text{ObjectPropertyAssertion}(Has t_{j_1,j_2,j_3}^i \tau(i, 3))
  \]
  where \( \tau(i, h) = p \), if the \( h \)-th literal in the clause \( i \) is the universal quantified variable \( p \), and \( \tau(i, h) = v_{j_1,j_2,j_3}^{i,h} \) otherwise.
- for each axiom \( \text{ObjectPropertyAssertion}(In v^p \tau_{j_1,j_2}^p) \) we add to \( \mathcal{O}_F' \) the axioms
  \[
  \text{ObjectPropertyAssertion}(Has t_{j_1,j_2}^p \tau(p, 1)), \text{ and }
  \text{ObjectPropertyAssertion}(Has t_{j_1,j_2}^p \tau(p, 2))
  \]
  where \( \tau(p, 1) = p \), if \( p \) is a universal quantified variable, \( \tau(p, 1) = v_{j_1,j_2}^{p,1} \) otherwise, and \( \tau(p, 2) = v_{j_1,j_2}^{p,2} \);
- for each \( v_{j_1,j_2,j_3}^{i,h} \), if \( j_h \) is 1, then the axiom \( \text{ObjectPropertyAssertion}(Val v_{j_1,j_2,j_3}^{i,h} E) \) is in \( \mathcal{O}_F' \), else the axiom \( \text{ObjectPropertyAssertion}(Val v_{j_1,j_2,j_3}^{i,h} D) \) is in \( \mathcal{O}_F' \);
- for each \( v_{j_1,j_2}^{p,h} \), if \( j_h \) is 1, then the axiom \( \text{ObjectPropertyAssertion}(Val v_{j_1,j_2}^{p,h} E) \) is in \( \mathcal{O}_F' \), else the axiom \( \text{ObjectPropertyAssertion}(Val v_{j_1,j_2}^{p,h} D) \) is in \( \mathcal{O}_F' \);
- for each universally quantified letter \( p \) of \( F' \) we add the axiom \( \text{ObjectPropertyAssertion}(Val p p) \).

Finally, we define the conjunctive query \( Q_{F'} \). In the following we use the variable \$l_{i,j}$ to denote the variable corresponding to the \( j \)-th letter occurring into the \( i \)-th clause of \( F' \), and we use the variable \$l_{p,j}$ to denote the variable corresponding to the \( j \)-th letter occurring into the auxiliary clause of \( F' \) associated to the propositional letter \( p \).
for each clause $c_i$ we have the following atoms in $Q_{F'}$, forming what we call the $c_i$-subquery:

- $\text{ObjectPropertyAssertion}(\text{In} \ c_i \ \$w_i)$
- $\text{ObjectPropertyAssertion}(\text{Has} \ \$w_i \ \$s_{i,1}) \ \text{ObjectPropertyAssertion}(\text{Val} \ \$s_{i,1} \ \$l_{i,1})$
- $\text{ObjectPropertyAssertion}(\text{Has} \ \$w_i \ \$s_{i,2}) \ \text{ObjectPropertyAssertion}(\text{Val} \ \$s_{i,2} \ \$l_{i,2})$
- $\text{ObjectPropertyAssertion}(\text{Has} \ \$w_i \ \$s_{i,3}) \ \text{ObjectPropertyAssertion}(\text{Val} \ \$s_{i,3} \ \$l_{i,3})$
- $\text{ObjectPropertyAssertion}(\text{Has} \ \$w_i \ \$z_i) \ \text{ObjectPropertyAssertion}(\text{Val} \ \$z_i \ \$y_i)$
- $\text{ClassAssertion}(\$y_i \ \$l_{i})$

for each letter $p$, and associated auxiliary clause $c^p$, we have the following atoms in $Q_{F'}$, forming what we call the $p$-subquery:

- $\text{ObjectPropertyAssertion}(\text{In} \ c^p \ \$w_p)$
- $\text{ObjectPropertyAssertion}(\text{Has} \ \$w_p \ \$s_{p,1}) \ \text{ObjectPropertyAssertion}(\text{Val} \ \$s_{p,1} \ \$l_{p,1})$
- $\text{ObjectPropertyAssertion}(\text{Has} \ \$w_p \ \$s_{p,2}) \ \text{ObjectPropertyAssertion}(\text{Val} \ \$s_{p,2} \ \$l_{p,2})$
- $\text{ObjectPropertyAssertion}(\text{Has} \ \$w_p \ \$s_{p,3}) \ \text{ObjectPropertyAssertion}(\text{Val} \ \$s_{p,3} \ \$l_{p,3})$
- $\text{ObjectPropertyAssertion}(\text{Has} \ \$w_p \ \$z_p) \ \text{ObjectPropertyAssertion}(\text{Val} \ \$z_p \ \$y_p)$
- $\text{ClassAssertion}(\$y_{p,1} \ \$l_{p,1}) \ \text{DisjointClasses}(\$y_{p,2} \ \$y_{p,2})$

Note that both $O_{F'}$ and $Q_{F'}$ have a size that is polynomial with respect to the size of $F$.

**Theorem 73** $F$ is satisfiable if and only if $O_{F'} \models Q_{F'}$.

**Proof.** $\Rightarrow$ Suppose that $F'$ is satisfiable, i.e., for every assignment $\alpha$ to the universally quantified variables of $F'$, the formula obtained from $F'$ by substituting each occurrence of the universally quantified variable $v$ with the value that $\alpha$ assigns to $v$ is satisfiable. We show that the certain answer to $Q_{F'}$ w.r.t. $O_{F'}$ is true, by showing that for any complete ontology derived from $O_{F'}$, the certain answer to the query is true.

Consider any assignment $\alpha$, and let $O_{F'}^\alpha$ be the ontology obtained from $O_{F'}$ by asserting that $v$ is empty for each class $v$ corresponding to the universally quantified variable $v$ to which $\alpha$ assigns false, and by asserting that $v$ is non-empty for each class $v$ corresponding to the universally quantified variable $v$ to which $\alpha$ assigns true. We show that the certain answer to $Q_{F'}$ w.r.t. $O_{F'}^\alpha$ is true. Consider any assignment $\beta$ to the existentially quantified variables of $F'$ such that $\alpha \cup \beta$ makes the body of $F'$ (i.e., the set of clauses constituting $F'$) true, and define the binding $B$ for the variables of $Q_{F'}$ as follows:

- For each clause $c_i$ and the associated $c_i$-subquery of $Q_{F'}$, $\$w_i$ is bound to the object $t_{j_1,j_2,j_3}^i$ such that $j_1, j_2, j_3$ is the combination of values that $\alpha \cup \beta$ assigns to the propositional letters appearing in clause $c_i$, whereas $\$s_{i,h}$ is bound to the appropriate $\tau(i, h)$, and each of the $\$l_{i,1}$, $\$l_{i,2}$, and $\$l_{i,3}$ is bound to $E$ or to $D$ according to whether the value the propositional letter corresponding to of $\$l_{i,j}$ assigned by $\alpha \cup \beta$ is true or false, respectively. Finally, $z_i$ and $y_i$ are bound to the values that makes $y_i$ any propositional letter that is true according to $\alpha \cup \beta$ (there must be at least one).

- For each propositional letter $p$ and the associated $p$-subquery of $Q_{F'}$, $\$w_p$ is bound to the object $t_{j_1,j_2}^p$ such that $j_1, j_2$ is the combination of values that $\alpha \cup \beta$ assigns to the propositional letters $p$ and $p$, whereas $\$s_{p,1}$ and $\$s_{p,2}$ are bound to the appropriate
\( \tau(p, h) \), and \( \$l_{i,1} \) and \( \$l_{i,2} \) are bound to \( E \) or to \( D \) according to whether the value of \( p \) assigned by \( \alpha \cup \beta \) is true or false, respectively. Finally, \( \$y_{p,1} \) and \( \$y_{p,2} \) are bound to the propositional letters \( p \) and \( \bar{p} \) respectively, if \( p \) is assigned true by \( \alpha \cup \beta \), otherwise they are bound to the propositional letters \( \bar{p} \) and \( p \), respectively.

It is immediate to verify that the binding \( B \) makes the answer to \( Q_{F'} \) true.

\( \iff \) Suppose that the certain answer to \( Q_{F'} \) w.r.t. \( O_{F'} \) is true. We show that \( F' \) is satisfiable. If the certain answer to \( Q_{F'} \) w.r.t. \( O_{F'} \) is true, then it follows that for every choice we make (empty or nonempty) for each class \( v \) corresponding to the universally quantified variable \( v \), there exists a binding for the variables of \( Q_{F'} \) such that \( Q_{F'} \) is true. Consider any such binding \( B \). By essentially reasoning in reverse with respect to the other direction of the theorem, one can easily verify that from \( B \) we can derive an assignment for the existentially quantified variables of \( F' \) that makes \( F' \) true. This proves that if the certain answer to \( Q_{F'} \) w.r.t. \( O_{F'} \) is true, then \( F' \) is satisfiable.

\( \square \)

By combining Theorem 73 with Theorem 68 we can state the following corollary.

**Corollary 74** Let \( O \) be a satisfiable ontology. Query answering over \( O \) is \( \Pi_{2}^{p} \)-complete in combined complexity.
Part IV

Practical applicability, implementation and evaluation
Chapter 10

An optimized algorithm for query answering

In the previous sections we have discussed the idea of answering queries over TBox-complete ontologies through an algorithm, that we will call \textsc{Naive} in the following, based on metagrounding. Note that all techniques that have been introduced in the previous sections for answering boolean queries can easily be adapted to the case of non-boolean queries where a subset of the variables must be returned as part of the answers. Despite the low theoretical data complexity, using the \textsc{Naive} algorithm may result in a very inefficient query answering process, that is unfeasible in real world scenarios. Indeed, the number of metagroundings that are needed in practice could be so huge that the idea of generating all of them, and computing their answers with respect to the ontology may result in an impractical method.

In this chapter we present a new algorithm, called \textsc{Lazy MetaGrounding} (LMG), already sketched in [57], for answering metaqueries over OWL 2 QL ontologies under MMS, and formally prove its properties. The algorithm is based on two important assumptions, that we now discuss.

1. LMG has been designed to work on TBox-complete ontologies, which is the class of ontologies that ensures tractability of query answering, as we saw in the previous chapter. One might observe that this is a strong assumption, limiting the applicability of our technique in practice. On the contrary, we argue that TBox-complete ontologies are neither unreasonable, nor expensive to be devised in practice. Indeed, from one hand, many ontologies derived from Entity-Relationship schemas, Object-oriented schemas or UML diagrams are actually TBox-complete. On the other hand, it is easy to see that, given a non TBox-complete ontology \(\mathcal{O}\) with a set \(\mathcal{S}\) of negative axioms that are uncertain, an automated, rather inexpensive procedure can be devised, that adds suitable facts \(\mathcal{O}\) to make the negative axioms that are not in \(\mathcal{S}\) violated. One can show that, in most cases, such an addition achieves TBox-completeness without changing the intended meaning of the ontology \(\mathcal{O}\).

2. Current highly optimized OWL 2 QL reasoners either do not allow answering queries with metavariables (e.g., [21]), or they do not consider substitutions of variables with complex expressions to compute the certain answers to queries with metavariables (e.g., [73]). Moreover, OWL 2 QL reasoners that allow answering metaqueries, only
accept queries that are legal for the Direct Semantics entailment regime of SPARQL, which means that any metavariable must occur only in one type of position. For all these reasons, similarly to the NAIVE algorithm, LMG aims at exploiting OWL 2 QL reasoners as blackboxes that compute answers to first order queries. This means that LMG must be designed in such a way that the queries it sends to OWL 2 QL reasoners must contain neither TBox atoms nor ABox atoms where predicate-variables occur.

In a nutshell, based on the above assumptions, LMG reduces the problem of computing the certain answers to a query under MS to the problem of computing the certain answers to various first-order queries over an OWL 2 QL ontology $O_{ext}$ that is obtained by extending the vocabulary $V_O$ and the set of ABox axioms of $O$ as follows. Intuitively, $O_{ext}$ encodes as ABox axioms over a special set of predicates the closure of the TBox axioms logically implied by $O$.

1. First, it adds to $V_O$ a class or an object property, for each type of OWL 2 QL TBox axioms. More precisely, it adds the predicates occurring in class or object property position in the the column $\tau(\alpha)$ of Table 10.1. For example the predicate ISAC is added to $V_O$ and it will be used to encode axioms of the form $\text{SubClassOf}(e_1 e_2)$ whereas the predicate IRR is added to $V_O$ and it will be used to encode axioms of the form $\text{IrreflexiveObjectProperty}(e_1)$.

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$\tau(\alpha)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SubClassOf($e_1 e_2$)</td>
<td>$\text{ObjectPropertyAssertion}(\text{ISAOP} e_1 e_2)$</td>
</tr>
<tr>
<td>SubObjectPropertyOf($e_1 e_2$)</td>
<td>$\text{ObjectPropertyAssertion}(\text{ISAOP} e_1 e_2)$</td>
</tr>
<tr>
<td>DisjointClasses($e_1 e_2$)</td>
<td>$\text{ObjectPropertyAssertion}(\text{DISJC} e_1 e_2)$</td>
</tr>
<tr>
<td>SubDataPropertyOf($e_1 e_2$)</td>
<td>$\text{ObjectPropertyAssertion}(\text{ISADP} e_1 e_2)$</td>
</tr>
<tr>
<td>DisjointDataProperties($e_1 e_2$)</td>
<td>$\text{ObjectPropertyAssertion}(\text{DISJDP} e_1 e_2)$</td>
</tr>
<tr>
<td>DataPropertyRange($e_1 e_2$)</td>
<td>$\text{ObjectPropertyAssertion}(\text{RANGE} e_1 e_2)$</td>
</tr>
</tbody>
</table>

Table 10.1. Function $\tau$

2. Second, for each TBox axiom $\alpha$ such that $\alpha$ is logically implied by $O$, it adds to $O$ the ABox axiom $\tau(\alpha)$, as defined in Table 10.1. Observe that the set of such ABox axioms can be built by first using an OWL 2 QL reasoner to infer all axioms logically implied and then by applying $\tau$. Note, in particular, that, doing so, for each expression $e \in Exp^O_O$, $O_{ext}$ includes an axiom $\text{ClassAssertion}(\text{ISAC} e \text{owl:Thing})$ and similar observations holds for expressions in $Exp^O_{OP}$, $V^O_{DP}$, and $V^O_{DT}$.

The rest of the chapter is organized in two sections. In Section 10.1 we describe the LMG algorithm, whereas in Section 10.2 we discuss its formal properties, namely, termination, computational cost, and correctness.

---

We assume here that OWL 2 QL reasoners admit occurrences, in individual positions, of entities belonging to the OWL 2 QL reserved vocabulary, which is in fact not allowed by the standard. Note, however, that such an assumption does not cause any loss of generality, since easy tricks can be devised to obtain a legal OWL 2 QL ontology that is equivalent to the one comprising such “non-legal” assertions, e.g., by introducing new entities that are equivalent to the entities belonging to the reserved vocabulary.
10.1 The LMG algorithm

Before delving into the details of the algorithm, we have first to introduce the notion of predicate-grounding of a conjunctive query, that is, a relaxed version of the notion of metagrounding of a conjunctive query given by Definition 45.

Definition 75 Let \(q\) be a conjunctive query over \(\mathcal{O}\). A predicate-grounding of \(q\) with respect to \(\mathcal{O}\) is a \(\mu\)-instantiation of \(q\), for some \(\mu: \vec{x} \leftarrow \vec{a}\), such that \(\vec{x} = (x_1, x_2, \ldots, x_n)\) are all the predicate-variables occurring in \(q\), and for every \(i \in \{1, \ldots, n\}\), if \(x_i\) occurs in class (resp. object property, data property, or datatype) position in \(q\), then \(a_i\) is any expression in \(Exp^C\) (resp. \(Exp^OP\), \(V^OP\), or \(V^DT\)).

Observe that if \(q\) does not contain any TBox atom, then each atom in a predicate-grounding of \(q\) is an ABox atom without predicate variables, and thus, it can be correctly evaluated by standard OWL 2 QL reasoners over \(\mathcal{O}_{ext}\).

In a nutshell, given a query \(Q = q_1 \cup \ldots \cup q_n\) and an ontology \(\mathcal{O}\), LMG(\(Q, \mathcal{O}_{ext}\)) works as follows:

1. it iterates over the various conjunctive queries in \(Q\), and for each \(q_i \in Q\), it applies the following three functions:
   - \(\text{PREPARE}(q_i)\), modifying \(q_i\) by adding suitable atoms according to specific rules, and by substituting every TBox atom \(\alpha\) with the ABox atom \(\tau(\alpha)\).
   - \(\text{BUILDQUERYPLAN}(q_i)\), computing a sequence \(S_i\) of annotated queries, which represents the query plan associated to \(q_i\). The query plan specifies how to split \(q_i\) into a sequence of metaqueries, and how to compute for each such metaquery appropriate predicate-groundings so as to decrease the overall number of predicate-ground queries to be evaluated with respect to blind metagrounding.
   - \(\text{EXECUTEQUERYPLAN}(S_i, \mathcal{O}_{ext})\), executing the queries in \(S_i\) over \(\mathcal{O}_{ext}\), and returning \(Ans_i\), that is the result of the last query of \(S_i\).

2. it returns \(\bigcup_{i \in \{1, \ldots, n\}} Ans_i\) as final result.

We now describe the three functions \(\text{PREPARE}, \text{BUILDQUERYPLAN}\) and \(\text{EXECUTEQUERYPLAN}\). In the following, we make use of the notion of final atom, defined recursively as follows: (i) every atom with at least one argument that is both a distinguished variable, and not a metavariable, is final; (ii) if \(a\) is a final atom, and \(a'\) shares with \(a\) an existential variable that is not a metavariable, then \(a'\) is a final atom. Note that, by definition, a duplicated atom is final.

We now describe the three functions \(\text{PREPARE}, \text{BUILDQUERYPLAN}\) and \(\text{EXECUTEQUERYPLAN}\). In the following, we make use of the notion of final atom, defined recursively as follows: (i) every atom with at least one argument that is both a distinguished variable, and not a metavariable, is final; (ii) if \(a\) is a final atom, and \(a'\) shares with \(a\) an existential variable that is not a metavariable, then \(a'\) is a final atom. Note that, by definition, a duplicated atom is final.

\(\text{PREPARE}\). Given a conjunctive query \(q\), we obtain the query \(q'\) by adding some atoms to \(q\), according to the following steps:
1. For each metavariable $x$ appearing in class position (respectively in object property, data property and datatype position) and not appearing in any TBox atom of $q$, we add the atom $\text{ObjectPropertyAssertion}(\text{ISAC } x \text{ owl:Thing})$ (respectively, $\text{ObjectPropertyAssertion}(\text{ISAOP } x \text{ owl:topObjectProperty})$, $\text{ObjectPropertyAssertion}(\text{ISADP } x \text{ owl:topDataProperty})$, $\text{ObjectPropertyAssertion}(\text{ISADT } x \text{ rdfs:Literal})$. Intuitively, the role of such atoms is to prevent the algorithm from generating more metavariable groundings than what is permitted by the alphabet.

2. For each atom $a$ of $q$ that contains as arguments both a distinguished variable $x$ that is not a predicate variable, and a predicate variable $y$, we add to the query an atom $a'$ obtained from $a$ by substituting $x$ with a fresh variable $x'$. We call the original atom $a$ a duplicated atom, and the new atom $a'$, its duplicate. Intuitively, the role of $a'$ is to allow getting the right values for $y$ without computing the bindings for $x$. It is immediate to verify that $q$ and $q'$ are equivalent, in the sense that, given the ontology $\mathcal{O}$, the set of certain answers to $q$ over $\mathcal{O}$ is exactly the set of certain answers to $q'$ over $\mathcal{O}$.

3. We replace each TBox atom $\alpha$, with an ABox atom $\tau(\alpha)$.

The following example illustrates the behaviour of $\text{Prepare}$.

**Example 76** Consider the ontology shown in Table 5.2 and the following query $q$, already discussed in Section 5.1, asking for financial instruments, together with their duration and the law which established the types they are instance of, that belong to a type which is disjoint from the type "Commercial paper" and which can be classified with respect to both the financial market of issuance and the payment mode:

```sql
select $fi$ $dr$ $lw$ where{
ClassAssertion(:Category_of_type_of_f_i_wrt_pay_mode $cat_1$) .
ClassAssertion(:Category_of_type_of_f_i_wrt_market $cat_2$) .
DisjointClasses(:Commercial_paper $tp$) .
ObjectPropertyAssertion(:established_by $tp$ $lw$) .
ClassAssertion($cat_1$ $tp$) .
ClassAssertion($cat_2$ $tp$) .
DataPropertyAssertion(:duration $fi$ $dr$) .
ClassAssertion($tp$ $fi$)
}
```

The application of the function $\text{Prepare}$ over $q$ consists of the following two steps:

1. It adds to $q$ the two atoms $\text{ObjectPropertyAssertion}(\text{ISAC }$ $\text{cat}_1$ $\text{owl:Thing})$ and $\text{ObjectPropertyAssertion}(\text{ISAC }$ $\text{cat}_2$ $\text{owl:Thing})$, as both variables $\text{cat}_1$ and $\text{cat}_2$ occur in class position in $q$ whereas they do not occur in any TBox atom of $q$;

2. It adds to $q$ the new atom $a' = \text{ObjectPropertyAssertion}(:\text{established_by }$ $\text{tp}$ $\text{lw}')$ as the duplicate of $a = \text{ObjectPropertyAssertion}(:\text{established_by }$ $\text{tp}$ $\text{lw})$, since $a$ contains both a distinguished variable (i.e., $\text{lw}$) and a predicate variable (i.e., $\text{tp}$) of $q$;
3. It replaces the TBox atom `DisjointClasses(:Commercial_paper $tp)` with the ABox atom `ObjectPropertyAssertion(DISJC :Commercial_paper $tp)`.

Thus the function `PREPARE` returns the following query $q'$:

```sql
select $fi $dr $lw where{
    ClassAssertion(:Category_of_type_of_f_i_wrt_pay_mode $cat_1).
    ClassAssertion(:Category_of_type_of_f_i_wrt_market $cat_2).
    ObjectPropertyAssertion(ISAC $cat_1 owl:Thing).
    ObjectPropertyAssertion(ISAC $cat_2 owl:Thing).
    ObjectPropertyAssertion(DISJC :Commercial_paper $tp).
    ObjectPropertyAssertion(:established_by $tp $lw).
    ObjectPropertyAssertion(:established_by $tp $lw').
    ClassAssertion($cat_1 $tp).
    ClassAssertion($cat_2 $tp).
    DataPropertyAssertion(:duration $fi $dr).
    ClassAssertion($tp $fi)
}
```

**BUILDQUERYPLAN.** Given a conjunctive query $q$, the function builds a sequence $S = (q^0, \ldots, q^m)$ of conjunctive queries, where the body of each $q^i$ comprises a subset of the atoms of $q$. To do so, it executes two steps:

(α) It builds a graph $G(q)$, marks some of the variables in the atoms of the query, and annotates the atoms of $q$ as follows:

1. The nodes of $G(q)$ are the atoms of $q$.
2. The edges of $G(q)$ and the marking of the variables in the atoms of $q$ are determined as follows. There is an edge from atom $a_1$ to atom $a_2$ if and only if $a_2$ is different from $a_1$ and at least one of the following occurs:
   (a) $a_1$ is not final, and it contains a variable $x$ in argument position that occurs in predicate position in $a_2$; in this case, $x$ is marked in $a_2$;
   (b) both $a_1$ and $a_2$ are not final, and there exists a variable $x$ occurring either in argument position both in $a_1$ and $a_2$, or in predicate position both in $a_1$ and $a_2$ (note that in this case, there will be an edge also from $a_2$ to $a_1$).

   Intuitively, edges encode an ordering among atoms, based on the idea that an edge from $a_1$ to $a_2$ in $G(q)$ without the inverse edge indicates that the evaluation of $a_1$ provides the instantiations for a variable of $a_2$, and two edges between $a_1$ and $a_2$ indicate that the two atoms should be evaluated together.
3. It annotates each non-final atom $a$ with an integer $d(a)$ denoting its depth, by using the following strategy:
   (a) $k \leftarrow 0$;
   (b) do the following until the graph contains only nodes corresponding to final atoms:
(b.1) assign \( k \) to each atom \( a \) occurring in a strongly connected component \( C \) of \( G(q) \) such that every atom in \( C \) has all of its incoming edges from \( C \);

(b.2) remove from \( G(q) \) all atoms annotated at step (b.1), together with all edges outgoing from such atoms;

(b.3) \( k \leftarrow k + 1 \).

(c) assigns \( k - 1 \) to all final atoms.

Intuitively, the depth induces a partition of the atoms of \( q \), and establishes an ordering for the evaluation of conjunctions of atoms: atoms with lower depth should be evaluated before atoms with higher depth, and atoms with the same depth should be evaluated together.

(\( \beta \)) It returns the sequence \( S = (q^0(w_0), \ldots, q^m(w_m)) \), where \( m \) is the maximal depth of atoms in \( G(q) \), and, for each \( j \) from 0 to \( m \), the conjunctive query \( q_j \) is such that (i) \( q_j \) is the conjunction of all atoms at depth \( j \), (ii) \( w_j \) contains each variable occurring in at least one body of queries \( S = (q^0(w_0), \ldots, q^j(w_j)) \), and is either a distinguished term in \( q \), or a marked variable in some queries \( S = (q^{j+1}(w_0), \ldots, q^m(w_m)) \), and (iii) \( w_m \) additionally contains all expressions that are distinguished in \( q \).

Intuitively, every query in the sequence has the role of computing some bindings for future predicate-variables, and in doing so it also remembers values that are to be provided in the final result.

The following example illustrates the behaviour of \textsc{BuildQueryPlan}.

\textbf{Example 77} Consider the query \( q' \) returned by the function \textsc{Prepare} in Example 76. Figure 10.1 synthesizes the steps performed by \textsc{BuildQueryPlan} to define and annotate the graph associated to the query, where "CA", "OPA", and "DPA", are abbreviations for \texttt{ClassAssertion}, \texttt{ObjectPropertyAssertion}, and \texttt{DataPropertyAssertion}, respectively, and ":CatP" and ":CatM" are abbreviations for \texttt{:Category_of_type_of_f_i_wrt_pay_mode} and \texttt{:Category_of_type_of_f_i_wrt_market}, respectively. In Figure 10.1(a) the initial graph is depicted. Figures from 10.1(b) to 10.1(d) show how the atoms are progressively annotated with depth and variables are marked within atoms. Note that the duplicated atom \texttt{ObjectPropertyAssertion(:established_by $tp $lw)} has been assigned a depth equal to 2 as it is a final atom. The final state of the graph is depicted in Figure 10.1(e).

Starting from the annotated graph depicted in Figure 10.1(e) the \textsc{BuildQueryPlan} function returns the sequence of annotated queries \( S = (q^0($cat_1 $cat_2), q^1($tp), q^2($fi $dr $lw)) \), where:

\( q^0: \text{select} \ $cat_1 \ $cat_2 \text{ where}\{
  \text{ClassAssertion(:Category_of_type_of_f_i_wrt_pay_mode $cat_1) .}
  \text{ClassAssertion(:Category_of_type_of_f_i_wrt_market $cat_2) .}
\)

\( q^1: \text{select} \ $cat_1 \ $cat_2 \text{ where}\{
  \text{ObjectPropertyAssertion($established_by $tp $lw) .}
\)

\( q^2: \text{select} \ $fi \ $dr \ $lw \text{ where}\{
  \text{ObjectPropertyAssertion($established_by $tp $lw) .}
\)

In general, \( q^j \) may differ from the syntax of conjunctive queries provided in Section 6.3 because of the presence of variables in its head that do not occur in its body. Note, however, that by construction, this can happen only for marked variables, which are expected to be bound to expressions when the query plan will be executed.
10.1 The LMG algorithm

Figure 10.1. Computation of the labeled graph associated to the query of Example 77
ObjectPropertyAssertion(ISAC $cat_1 owl:Thing).
ObjectPropertyAssertion(ISAC $cat_2 owl:Thing).

q^1: select $tp where
ObjectPropertyAssertion(DISJC :Commercial_paper $tp) .
ObjectPropertyAssertion(:established_by $tp $lw′) .
ClassAssertion($cat_1 $tp). ClassAssertion($cat_2 $tp) .

q^2: select $fi $dr $lw where
ObjectPropertyAssertion(:established_by $tp $lw) .
DataPropertyAssertion(:duration $fi $dr) .
ClassAssertion($tp $fi)

Note that the heads of the queries are built as follows:

- Variables $cat_1 and $cat_1 occur in the head of q^0 because they both occur in
  the body of q^0 and are marked in q^1;

- Variable $tp occurs in the head of q^1 because it occurs in the body of q^1 and it is
  marked in q^2;

- Variables $fi, $dr and $lw occur in the head of q^2 because they are distinguished
  terms of the query given as input.

\[ \square \]

**EXECUTEQUERYPLAN.** Based on the sequence of queries \( S = (q^0, \ldots, q^m) \) returned by **BUILDQUERYPLAN(q)**, the function **EXECUTEQUERYPLAN(S, O^{ext})** progressively builds the answers to \( q \). In particular, for each \( j \geq 0 \), it computes appropriate predicate-groundings of \( q^j \), evaluates each of them by using an OWL 2 QL reasoner, and stores the resulting tuples into an intermediate relation \( A_j \). Note, in particular, that if \( j > 0 \), the predicate-groundings of \( q^j \) are computed using \( A_{j-1} \). Hence, the goal of \( A_j \) is twofold: on one hand it maintains a projection of the answers to \( q \) computed so far (if any); on the other hand, it provides substitutions for predicate-variables of \( q^{j+1} \) to obtain predicate-ground queries evaluable using standard OWL 2 QL reasoners.

In order to precisely determine which are the appropriate predicate-grounding of \( q_j \) to build on the basis of \( A_{j-1} \), we need a specific notation. Consider a tuple \( \vec{t} \in A_{j-1} \) with schema \( w_{j-1} \). We denote by \( \gamma(\vec{t}, q_j, O) \) the set of predicate-ground queries such that if \( q^j = \mu^j(q^j) \) is predicate-ground, where \( \mu^j : w_{j-1} \leftarrow \vec{t} \), then \( \gamma(\vec{t}, q_j, O) \) contains only \( q^j \). otherwise \( \gamma(\vec{t}, q_j, O) \) contains all possible predicate-groundings of \( q^j \) with respect to \( O \). For example, if the tuple \( \vec{t} = (c_1, c_2) \) is in \( A_{j-1} \) with schema \( (x_1, x_2) \), the classes in the ontology \( O \) are \( c_1, c_2, c_3, \ldots \), and \( q_j \) is

\[
\text{select } x_4 \text{ where } \{ \text{ClassAssertion}(x_1x_4). \text{ClassAssertion}(x_2x_4). \text{ClassAssertion}(x_3x_5) \}.
\]

then \( \gamma(\vec{t}, q_j, O) \) contains the following queries:
select $x_4$ where \{ ClassAssertion($c_1 \, x_4$). ClassAssertion($c_2 \, x_4$). ClassAssertion($c_3 \, x_5$) \}.

select $x_4$ where \{ ClassAssertion($c_1 \, x_4$). ClassAssertion($c_2 \, x_4$). ClassAssertion($c_2 \, x_5$) \}.

select $x_4$ where \{ ClassAssertion($c_1 \, x_4$). ClassAssertion($c_2 \, x_4$). ClassAssertion($c_3 \, x_5$) \}.

We are now able to present the function \texttt{EXECUTEQUERYPLAN}. Given a sequence of queries $S = (q^0(w_0^i), \ldots, q^m(w_m^i))$ and an ontology $O$, \texttt{EXECUTEQUERYPLAN}(S, O^{ext}) proceeds as follows:

1. It evaluates $q^0$ over $O^{ext}$ and stores the results into the relation $A_0$ with schema ($w_0^i$), and it sets $j = 1$. Note that, by construction, $q^0$ is a first-order query and thus it can be correctly evaluated using any OWL 2 QL reasoner.

2. While $A_{j-1} \neq \emptyset$ and $j \leq m$, it does the following:
   
   (a) if we denote by $A_t^j$ the union of the sets of tuples returned by the evaluation of all queries in $\gamma(t, q_j, O)$ using an OWL 2 QL reasoner, then it stores in the relation $A_j$ with schema ($w_j^i$) the union of the various $A_t^j$'s for all tuples $t$ in $A_{j-1}$;
   
   (b) it sets $j = j + 1$.

3. It returns the set of tuples in $A_{j-1}$ that contains only IRIs, i.e., it disregards the tuples of $A_{j-1}$ containing complex expressions.

The following example illustrates the behavior of \texttt{EXECUTEQUERYPLAN}.

\textbf{Example 78} Consider the query $q$ of Example 76, the sequence of annotated queries $S = (q^0($\texttt{cat\_1} $\texttt{cat\_2}$), $q^1($\texttt{tp}$), $q^2($\texttt{fi} $\texttt{dr} $\texttt{lw}$))$ computed by \texttt{BUILDQUERYPLAN}(q), and the ontology $O$ in Table 5.2. Starting from $i = 0$, \texttt{EXECUTEQUERYPLAN}(q, $O^{ext}$) considers each $q^i$ at a time. Specifically, it first evaluates $q^0$ over $O^{ext}$ and stores the result in the relation $A_0$ with schema ($\texttt{cat\_1}$, $\texttt{cat\_1}$), thus obtaining

$$A_0 = \{(\text{:Type\_of\_f\_i\_with\_coupon, :Type\_of\_foreign\_f\_i}),$$
$$\quad (\text{:Type\_of\_f\_i\_without\_coupon, :Type\_of\_foreign\_f\_i}),$$
$$\quad (\text{:Type\_of\_f\_i\_without\_coupon, :Type\_of\_Italian\_f\_i}),$$
$$\quad (\text{:Type\_of\_f\_i\_with\_coupon, :Type\_of\_Italian\_f\_i})\}.$$ 

Then it builds the empty relation $A_1$ with schema ($\texttt{tp}$), and based on the projection of $A_0$ onto $\texttt{cat\_1}$ and $\texttt{cat\_2}$ it constructs the queries

\begin{verbatim}
q^1_0: select $\texttt{tp}$ where{
    ClassAssertion(:Type\_of\_f\_i\_with\_coupon $\texttt{tp}$).
    ClassAssertion(:Type\_of\_foreign\_f\_i $\texttt{tp}$).
    ObjectPropertyAssertion(DISJC :Commercial\_paper $\texttt{tp}$).
    ObjectPropertyAssertion(:established\_by $\texttt{tp}$ $\texttt{lw}$')
}
q^1_1: select $\texttt{tp}$ where{
    ClassAssertion(:Type\_of\_f\_i\_without\_coupon $\texttt{tp}$).
    ClassAssertion(:Type\_of\_foreign\_f\_i $\texttt{tp}$).
    ObjectPropertyAssertion(\texttt{disjc} :\texttt{Commercial}\_\texttt{paper} $\texttt{tp}$).
    ObjectPropertyAssertion(\texttt{established}\_\texttt{by} $\texttt{tp}$ $\texttt{lw}$')
}
q^1_2: select $\texttt{tp}$ where{
    ClassAssertion(:Type\_of\_f\_i\_with\_coupon $\texttt{tp}$).
    ClassAssertion(:Type\_of\_Italian\_f\_i $\texttt{tp}$).
    ObjectPropertyAssertion(DISJC :Commercial\_paper $\texttt{tp}$).
    ObjectPropertyAssertion(:established\_by $\texttt{tp}$ $\texttt{lw}$')
}
q^1_3: select $\texttt{tp}$ where{
    ClassAssertion(:Type\_of\_f\_i\_without\_coupon $\texttt{tp}$).
    ClassAssertion(:Type\_of\_Italian\_f\_i $\texttt{tp}$).
    ObjectPropertyAssertion(\texttt{disjc} :\texttt{Commercial}\_\texttt{paper} $\texttt{tp}$).
    ObjectPropertyAssertion(\texttt{established}\_\texttt{by} $\texttt{tp}$ $\texttt{lw}$')
}
\end{verbatim}
10. An optimized algorithm for query answering

\[
\begin{align*}
q_1^1 &:\text{select } \text{where} \\
&\{ \\
&\text{ClassAssertion}(\text{:Type_of_f_i_without_coupon } \text{$tp}) . \\
&\text{ClassAssertion}(\text{:Type_of_foreign_f_i } \text{$tp}) . \\
&\text{ObjectPropertyAssertion}(\text{DISJC :Commercial_paper } \text{$tp}) . \\
&\text{ObjectPropertyAssertion}(\text{:established_by } \text{$tp} \text{$lw}) \}
\end{align*}
\]

\[
q_1^2: \text{select } \text{where} \\
\{ \\
\text{ClassAssertion}(\text{:Type_of_f_i_without_coupon } \text{$tp}) . \\
\text{ClassAssertion}(\text{:Type_of_Italian_f_i } \text{$tp}) . \\
\text{ObjectPropertyAssertion}(\text{DISJC :Commercial_paper } \text{$tp}) . \\
\text{ObjectPropertyAssertion}(\text{:established_by } \text{$tp} \text{$lw}) \}
\]

evaluates them over $O^{ext}$ and stores their answers into $A_1$. It’s easy to verify that the only query with a non empty answer is $q_1^1$, thus it poses $A_1=$\{\{\text{:BTP}\}\}. Then it builds the empty relation $A_2$ with schema ($fi$, $dr$, $lw$), and based on the projection of $A_1$ onto $\text{$tp$}$, it constructs the query

\[
q_2^2: \text{select } \text{$fi$ $dr$ $lw$ where} \\
\{ \\
\text{ObjectPropertyAssertion}(\text{:established_by :BTP } \text{$lw}) . \\
\text{DataPropertyAssertion}(\text{:duration $fi$ $dr}) . \\
\text{ClassAssertion}(\text{:BTP $fi$}) \}
\]

evaluates them over $O^{ext}$ and stores their answers into $A_2$, thus obtaining $A_2=$\{\{\text{:IT0005069395,"6"^xsd:integer,:dr_144bis}\}\} and returns it as answer to $q$.

10.2 Properties of LMG

In this section we discuss the properties of the LMG algorithm, namely termination, complexity, and correctness.

**Theorem 79.** For any ontology $O$ and query $Q$, LMG($Q, O^{ext}$) terminates, and its cost is PTIME with respect to the size of $O$, and EXPTIME with respect to the size of $Q$.

**Proof.** The algorithm considers every conjunctive query $q$ in $Q$, and therefore it is sufficient to show termination of the execution of the two functions $\text{BUILDQUERYPLAN}(q)$ and $\text{EXECUTEQUERYPLAN}$ on a conjunctive query $q$. Thus, termination of LMG follows from the fact that the function $\text{BUILDQUERYPLAN}(q)$ considers every pairs of atoms in $q$, constructs and traverse a graph with a finite number of nodes, and produces a query.
plan consisting of a number of queries that at most coincides with the depth of the graph. \textsc{ExecuteQueryPlan} executes a number of steps that coincides with the number of queries in the query plan associated to \( q \), where each step essentially consists in the evaluation over the ontology of a finite number of metaground queries under the first-order semantics.

As for the worst-case analysis of the cost of the algorithm, it is sufficient to observe the following for a single conjunctive query \( q \):

- The cost of \textsc{BuildQueryPlan}(\( q \)) is polynomial with respect to the size of \( q \). Indeed, both the cost of constructing the graph \( G(q) \), and the cost of building the query plan \( S \) is at most quadratic with respect to the size of the conjunctive query \( q \).

- The cost of \textsc{ExecuteQueryPlan}(\( S, O^{\text{ext}} \)) can be characterized by observing that in the worst case, the number of queries generated by \textsc{ExecuteQueryPlan}(\( S, O^{\text{ext}} \)) is at most \(|O|^n\), where \(|O|\) is the size of the \( O \), and \( n \) is the arity of \( q \). Each such query is evaluated by a first-order reasoner, and the cost of the evaluation is \textsc{PTIME} with respect to \( D \) and \textsc{NP} with respect to the size of the query.

\[ \Box \]

We now turn our attention to correctness. In what follows, we will use the following definitions and notations.

- given a substitution of variables \( \mu : \bar{x} \leftarrow \bar{t} \), we call \textit{domain} of \( \mu \), denoted \textit{domain}(\( \mu \)), the set of variables in \( \bar{x} \);
- given two substitutions \( \mu_1 \) and \( \mu_2 \), we say that \( \mu_1 \) and \( \mu_2 \) are \textit{compatible}, if for every variable \( x \in \text{domain}(\mu_1) \cap \text{domain}(\mu_2) \), we have that \( \mu_1(x) = \mu_2(x) \); then, if \( \mu_1 \) and \( \mu_2 \) are compatible and \( \text{domain}(\mu_1) \subseteq \text{domain}(\mu_2) \), we write \( \mu_1 \subseteq \mu_2 \);
- given two compatible assignments \( \mu_1 \) and \( \mu_2 \), we denote by \([\mu_1, \mu_2]\) the assignment obtained as the extension of \( \mu_1 \) to the variables of in \( \text{domain}(\mu_2) \setminus \text{domain}(\mu_1) \);
- given a substitution \( \mu_1 \) and a tuple of variables \( \bar{x} \), such that every variable in \( \bar{x} \) belongs to \( \text{domain}(\mu_1) \), we say that a substitution \( \mu_2 \) is the \textit{restriction} of \( \mu_1 \) to \( \bar{x} \), if \( \text{domain}(\mu_2) \subseteq \text{domain}(\mu_1) \) and for every \( x' \in \bar{x} \), \( \mu_2(x') = \mu_1(x') \);
- given a (possibly empty) tuple of terms \( \bar{w} \), a relation \( R \) with schema \( (\bar{w}) \), and a tuple \( \bar{a} \) in \( R \), we denote by denoted by \( \mu_{\bar{a}} \), the substitution \( \bar{w} \leftarrow \bar{a} \); note that if \( \bar{w} \) is empty and \( \bar{a} \) is the empty tuple, then \( R \) is either empty or it contains \( \bar{a} \), in which case \( \mu_{\bar{a}} \) has an empty domain and, thus, for every query \( q \), \( \mu_{\bar{a}}(q) = q \);
- given a query \( q \), we will denote by \textit{body}(\( q \)) the formula \( \exists \bar{y} \text{ body}(q) \) where \( \bar{y} \) are all variables in the body of \( q \) that are not distinguished terms of \( q \).

For the correctness of LMG, we base our arguments on the correctness of the \textsc{Naive} algorithm \([57]\). In particular, we prove a fundamental, preliminary lemma.

\textbf{Lemma 80} \textit{Let} \( O \) \textit{be an ontology,} \( q(\bar{x}_d) \) \textit{a conjunctive query over} \( O \), \textit{and} \( \bar{a} \) \textit{a tuple of entity names in} \( V^O_N \). \textit{Then,} \( \bar{a} \) \textit{is returned by} \( \text{LMG}(q(\bar{x}_d)), O^{\text{ext}} \) \textit{if and only if} \( \bar{a} \) \textit{is returned by} \( \text{Naive}(q(\bar{x}_d), O) \).

\textit{Proof.} \text{To simplify the proof and without loss of generality, in the following we assume that} \( \bar{x}_d \) \text{contains only variable, i.e., it does not comprise any entity name in} \( V^O_N \). \text{Indeed, by definition of certain answers, if} \( \bar{x}_d \) \text{comprises an entity name, then such name is included in every tuple that is a certain answer to} \( q \). \text{But then, by correctness of} \textsc{Naive}, \text{it is included in every tuple that is an answer to} \( \text{Naive}(q(\bar{x}_d), O) \). \text{It is easy to see}
that by construction, such entity name is in the head of every query \( q^j \) in the sequence produced by \textsc{BuildQueryPlan}(q), and, therefore, is included in every tuple returned by \textsc{LMG}(q(x_d), O^{ext})�.

“Only-if-part”. We show that if \( \text{LMG}(q(x_d), O^{ext}) \) returns \( \tilde{a} \), then \( \tilde{a} \) is returned by \textsc{Naive}(q(x_d), O). Let \( S = (q^0(w_0), \ldots, q^m(w_m)) \) be the sequence of annotated queries obtained by the execution of \textsc{BuildQueryPlan}(q(x_d)). Intuitively, we show that by keeping track of the substitutions for metavariables progressively computed to obtain \( \tilde{a} \), we get a substitution that once applied to \( q \) provides a metagrounding whose evaluation over \( O \) returns \( \tilde{a} \). We do so by induction on \( m \).

**Base step.** If \( m = 0 \), then \( S = \{ q^0 \} \). It is easy to see that, by construction, \( q(x_d) \) is predicate-ground. Moreover, either \( q \) is a first-order query and \( q^0 = q \), or \( q \) contains at least one TBox atom and \( q^0 \) is obtained from \( q \) by replacing every TBox atom \( \alpha \) with \( \tau(\alpha) \). We next show that in both cases, for every \( \tilde{a} \in A_0 \), \( \tilde{a} \) is returned by \textsc{Naive}(q, O). The proof is trivial if \( q \) is first-order. Otherwise, the proof follows from the fact that \( O^{ext} \models \mu(q) \), where \( \mu : x_d \leftarrow \tilde{a} \) if and only if \( O^{ext} \models \mu(q_e) \) and \( O^{ext} \models \mu(q_i) \), where \( q_e \) is first-order and consists of the ABox atoms of \( q \), and \( q_i \) consists of the TBox atoms of \( q \). But then, \( O^{ext} \models \mu(q) \) if and only if \( O^{ext} \models \mu(\tau(\alpha_1)), O^{ext} \models \mu(\tau(\alpha_2)), \ldots, O^{ext} \models \mu(\tau(\alpha_m)) \), where \( \alpha_1, \alpha_2, \ldots, \alpha_m \) are all TBox atoms in \( q_i \). By induction, this implies that \( O \models \mu(\alpha_1), O \models \mu(\alpha_2), \ldots, O \models \mu(\alpha_m) \), from which it follows that \( \tilde{a} \) is returned by \textsc{Naive}(q, O).

**Inductive step.** If \( m > 0 \), then \( \tilde{a} \) is in \( A_m \), and \( A_m \) is computed on the basis of \( A_{m-1} \) and \( q^m(w_m) \). If \( q' \) is the query constituted by the conjunction of \( q^0(w_0), \ldots, q^{m-1}(w_{m-1}) \), then we know that \( \text{LMG}(q', O^{ext}) \) has returned the set of tuples in \( A_{m-1} \) as result. By inductive hypothesis, every tuple \( \tilde{a}' \) in \( A_{m-1} \) is returned by \textsc{Naive}(q', O) for a certain instantiation \( \mu' \) of the metavariables of \( q' \). Since \( \tilde{a} \) is in \( A_m \), we have that there is a \( \tilde{a}' \) in \( A_{m-1} \) and a metaground query \( q'' \) in \( \gamma(\tilde{a}', q_j, O) \) such that \( \tilde{a} \) is in the answer of \( q'' \) with respect to \( O \). If we denote by \( \mu'' \) the metavariables instantiation corresponding to \( q'' \), it is immediate to see that \( \tilde{a} \) is returned by \textsc{Naive}(q, O) for the metavariable instantiation obtained as concatenation of \( \mu' \) and \( \mu'' \).

“If-part”. We now prove the “if direction” of the lemma. Suppose that \( \tilde{a} \) is returned by \textsc{Naive}(q(x_d), O), and let \( S \) be the sequence of queries returned by \textsc{BuildQueryPlan}(q). We prove that \( \tilde{a} \) is returned by \textsc{ExecuteQueryPlan}(S, O^{ext}). Intuitively, we show that the query plan encoded by \( S \), once executed by the function \textsc{ExecuteQueryPlan}(S, O^{ext}), computes \( \tilde{a} \) by progressively providing substitutions for the distinguished variables and the metavariables of \( q \) that are “coherent” with both \( \tilde{a} \) and the metagrounding whose evaluation returns \( \tilde{a} \) using \textsc{Naive}. In more details, in order for \( \tilde{a} \) to be returned by \textsc{Naive}(q, O), \( \tilde{a} \) must be returned by \( a \) the evaluation over \( O \) of a metagrounding of \( q \) with respect to \( O \) under the first-order semantics. Let \( q_A \) be one such metagrounding and let \( \mu_A = \tilde{X} \leftarrow \tilde{A} \) be such that \( q_A = \mu_A(q) \), where \( \tilde{X} \) consists of all metavariables occurring in the body of \( q \) and \( \tilde{A} \) is a tuple if expressions in \( Exp^O \). Then, we have that \( O \models \mu_A(body(\mu_A(q))) \), from which it follows that \( \mu_{\tilde{a}} = \mu_A(q) \) are such that

\[
O \models [\mu_A, \mu_{\tilde{a}}](body(q)) \quad (10.1)
\]

Now, let \( S \) have the form \( (q^0(w_0), \ldots, q^m(w_m)) \). We next show that:

\[
\forall j \in \{0, \ldots, m\}, \exists \tilde{a} \in A_j \text{ such that } \mu_{\tilde{a}} \subseteq [\mu_A, \mu_{\tilde{a}}]. \quad (10.2)
\]
Properties of LMG 161

Note that by showing (10.2), we conclude the proof. Indeed, (10.2) implies that for $j = m$, there exists $\vec{a}^m \in A_m$ such that $\mu_{\vec{a}^m} \subseteq [\mu_A, \mu_\vec{a}]$. But then, since $\text{domain}(\mu_{\vec{a}^m}) = \text{domain}(\mu_\vec{a})$, we obtain that $\vec{a}^m = \vec{a}$, and since $\vec{a}$ comprises only entity names in $V^c_N$, $\vec{a}$ is returned by $\text{EXECUTE}_\text{QUERYPLAN}(S, O^{ext})$.

Thus, in the following, we show (10.2), by induction on the size of $S$.

**Base step.** Let $\mu$ be the restriction of $[\mu_A, \mu_\vec{a}]$ to $\vec{w}_0$ and let $\vec{a}_0 = \mu(\vec{w}_0)$. Clearly, $\mu = \mu_{\vec{a}_0}$ and $\mu_{\vec{a}_0} \subseteq [\mu_A, \mu_\vec{a}]$. We next show that $\vec{a}_0 \in A_0$. We know that $A_0$ stores the result of the evaluation of $q^0$ over $O$ under the first-order semantics. But then, by combining (10.1) with the fact that $\mu_{\vec{a}_0}$ is the restriction of $[\mu_A, \mu_\vec{a}]$ to $\vec{w}_0$ and that the body of $q^0$ contains a subset of the atoms of $q$, we get that $O \models \mu_{\vec{a}_0}(\text{body}(q^0))$. Thus, by the definition of certain answer, we can conclude that $\vec{a}_0$ is a certain answer to $q^0$ over $O$, and thus belongs to $A_0$.

**Inductive step.** Let $k$ be such that $0 < k < m$. We assume that (10.2) is true for $j = k$ and show that it is true for $j = k + 1$. Thus, let $\vec{a}_k \in A_k$ be such that $\mu_{\vec{a}_k} \subseteq [\mu_A, \mu_\vec{a}]$ and let us assume, by contradiction, that there exists no tuple $\vec{a}_k+1 \in A_k+1$ such that $\mu_{\vec{a}_k+1} \subseteq [\mu_A, \mu_\vec{a}]$.

Since (10.2) is true for $j = k$, then $\mu_{\vec{a}_k}$ and $[\mu_A, \mu_\vec{a}]$ are compatible. Therefore, let $\mu$ be the restriction of $\mu_{\vec{a}_k}$ and $[\mu_A, \mu_\vec{a}]$ to $\vec{w}_{k+1}$.

Clearly, it follows that $\mu \subseteq [\mu_A, \mu_\vec{a}]$. Now, let $\vec{t} = \mu(\vec{w}_{k+1})$. From the hypothesis that there exists no tuple $\vec{a}_{k+1}$ in $A_{k+1}$ such that $\mu_{\vec{a}_{k+1}} \subseteq [\mu_A, \mu_\vec{a}]$, we have that $\vec{t} \notin A_{j+1}$, or, equivalently, that $O \not\models \mu(\text{body}(q^{k+1}))$. However, by combining (10.1) with the observation that, by construction, (i) the atoms in the body of $q^{k+1}$ are a subset of the atoms in the body of $q$ and (ii) all existential variables in $q^{k+1}$, do not occur in any other atom of $q$, we have that $O \models [\mu_A, \mu_\vec{a}](\text{body}(q^{k+1}))$, which, considering that $\mu$ is the restriction of $[\mu_A, \mu_\vec{a}]$ to $\vec{w}_{k+1}$, leads to $O \models \mu(\text{body}(q^{k+1}))$, and, hence, to a contradiction.

Our final correctness theorem easily follows from the above lemma and the correctness of $\text{NAIVE}$.

**Theorem 81** Let $O$ be a TBox-complete ontology, let $Q$ be a query, and $\text{Ans}$ be the set returned by LMG$(Q, O^{ext})$. Then $\text{Ans}$ is the certain answers to $Q$ over $O$.

**Proof.** Since $Q$ is a union of conjunctive queries, LMG iterates over every conjunctive query in the union to produce the result, and the same is true by $\text{NAIVE}$. Since we have shown in lemma [80] that the two algorithms behave the same for conjunctive queries, the thesis follows from the correctness of $\text{NAIVE}$. □
Chapter 11

Implementation and evaluation

In this Chapter we present a prototype system, called MQ-MASTRO, that enables metasyncueling over TBox-complete ontologies by implementing the algorithms that have been described in the previous sections. We start by describing the general framework of the system in Section 11.1. Then we show how we used such a framework to provide a plugin for the open source ontology editor PROTEGE in Section 11.2. Finally, we present a series of experiments testing the performances of MQ-MASTRO in Section 11.3.

11.1 The MQ-MASTRO system

MQ-MASTRO is a prototype system that provides, by implementing the LMG algorithm described in Section 10.1, a service for answering metaqueries over TBox-complete ontology under MMS. The architecture of MQ-MASTRO is depicted in Figure 11.1 and it is composed of four layers:

1. Inputs: ontologies, and (meta)queries;

2. MQ-MASTRO core: metaquery answering engine;

3. High-level APIs: enabling communication between MQ-MASTRO core layer and the standard OWL 2 QL reasoner layer;


MQ-MASTRO is available as a Java library implementing the OWL API [44], which are the reference interfaces for Java applications to create, manipulate, and reason over OWL 2 ontologies. In fact, MQ-MASTRO extends the OWL API by defining a SPARQL query answering service over ontologies. The system uses Apache Jena API [25] to parse the queries presented as input, and is able to serialize SPARQL queries that have to be sent to the standard OWL 2 QL reasoner using both Apache Jena API and Sesame API [17], which are two widely used Java APIs for creating and manipulating SPARQL queries. To the best of our knowledge, MQ-MASTRO is the first system allowing sound and complete metaquerying over OWL 2 ontologies. In particular, the system assumes the ontology to be
expressed in the OWL 2 QL profile, and the query to be expressed in SPARQL. The system has been designed to use any OWL 2 QL reasoner implementing the OWL API and offering first-order query answering capabilities through Jena or Sesame APIs.

![Figure 11.1. Architecture of the MQ-MASTRO system](image)

### 11.2 The MQ-MASTRO plugin for PROTÉGÉ

In addition to the Java library version, MQ-MASTRO is available as a plugin for the open source ontology editor PROTÉGÉ. The architecture of the plugin is depicted in Figure 11.2. Such a plugin has been implemented by using MASTRO [21] as off-the-shelf first-order OWL 2 QL reasoner which in turn use the H2 Database Engine [3] to store the input ontologies.

Being an implementation of the OWL.Reasoner interface of the OWL API, MQ-MASTRO plugin offers, through suitable standard user interfaces offered by PROTÉGÉ, all the reasoning services of a standard OWL 2 QL reasoner such as, for example, ontology satisfiability checking. In addition it offers specific services that are made available to users through a suitable panel integrated into the PROTÉGÉ graphical interface. Such a panel is depicted in Figure 11.3. In particular users can exploit the panel to perform the following operations:

---

1Note that we are actually considering a subset of the queries that can be expressed using the query language presented in Section 6.3. In particular, MQ-MASTRO accepts queries whose terms are restricted to belong to \( V_N \cup V \) (instead of \( \text{Exp}^O \cup \text{Exp}_N^O \)) and \( V \cup V \).

2http://protege.stanford.edu/

1. Verification of TBox-completeness of OWL 2 QL ontologies;

2. Evaluation of metaqueries, expressed in SPARQL, over TBox-complete ontologies under the MMS semantics;

3. Saving and loading of SPARQL queries from and to text files.

Figure 11.2. Architecture of the MQ-MASTRO plugin for PROTEGE

Figure 11.3. Graphical interface of the MQ-MASTRO plugin for PROTEGE
11.3 Evaluation

In this section we present a series of experiments testing the performances of query answering with MQ-MASTRO, within the real world scenario mentioned in Section 5.1. In the experiments, we considered four ontologies, all obtained as progressive extensions of the ontology partially described in Section 5.1, and we evaluated 9 metaqueries over each of such ontologies. Furthermore, in order to provide evidence of the effectiveness of the optimization strategies adopted by LMG, we developed a second version of MQ-MASTRO, called MQ-MASTRO_{naive}, which implements the NAIVE algorithm for metaquerying, and also uses MASTRO as underlying OWL 2 QL reasoner, and compared the performances of query answering in MQ-MASTRO, and MQ-MASTRO_{naive}.

Let us first provide a brief description of the ontologies and the queries used in the experiments. As for the ontologies, Ont_0 specifies a scenario with \( \approx 26000 \) financial instruments, 23 types of financial instruments and 24 categories of type of financial instrument. Ont_1, Ont_2, and Ont_3 are all extensions of Ont_0, constructed as follows:

- Ont_1 is obtained from Ont_0 by adding only individuals and ABox axioms. In particular, Ont_1 extends Ont_0 with \( \approx 7000 \) fresh individuals representing financial instruments along with \( \approx 35000 \) ABox axioms describing their properties.

- Ont_2 is obtained from Ont_0 by adding new classes with the corresponding instances and TBox axioms (for example, the subset and disjoint axioms):
  - two classes representing new types of financial instruments, namely :BTP_i and :BTP_Italia;
  - two classes representing new categories of type of financial instruments, namely :Type_of_f_i_born_in_the_money_market and :Type_of_f_i_born_in_the_financial_market, such that every type of financial instrument is an instance of one of these two classes;
  - one class, namely :Category_of_type_of_f_i_wrt_birth_market, containing as instances the newly introduced categories, thus representing a new classification criterion of types of financial instruments.

- Ont_3 is obtained by adding to Ont_2 \( \approx 4000 \) new instances of :BTP_i and :BTP_Italia, along with \( \approx 20000 \) ABox axioms describing their properties.

In Table [11.1] we summarize the characteristics of the test ontologies in terms of (i) the number of classes (denoted by \( V_O^C \)), (ii) number of metaclasses, i.e. classes occurring also in individual position (\( Mt-Cl \)), (iii) number of object and data properties (\( V_O^OP \) and \( V_D^DP \), respectively), (iv) the number of individuals (\( Ind^O \)), (v) the number of TBox axioms, and (vi) the number of ABox axioms.

As for the queries used in the experiments, they are listed in Table [11.2] and are described as follows:

- \( Q_0 \), \( Q_1 \), and \( Q_2 \) ask for pairs, triples and quadruples of IRIs, respectively, connected through chains of instanceOf relationship expressed as ClassAssertion atoms, of increasing length (from 1 up to 3 atoms).
11.3 Evaluation

- $Q_3$ and $Q_4$ are similar to $Q_1$ and $Q_2$, respectively, but the `instanceOf` chain in both queries starts with a specific entity, namely `Type_of_f_i` and `Category_of_type_of_f_i` respectively.
- $Q_5$ is obtained from $Q_2$ by adding a TBox atom of type `SubClassOf` in the body, constraining the starting node of the `instanceOf` chain to be a subclass of `Category_of_type_of_f_i`.
- $Q_6$ asks for financial instruments, together with their duration, that belong to a type which can be classified with respect to both the financial market of issuance and the payment mode; thus, the query is constituted by two joining `instanceOf` chains.
- $Q_7$ asks for financial instruments along with their duration and type, such that their type is an Italian type of financial instrument; thus, it is constituted by an `instanceOf` chain of length 2, plus an atom asking for a specific data property of the ending node of the chain.
- $Q_8$ asks for financial instruments along with the law ruling the types they are instances of; thus it is constituted by an `instanceOf` chain of length 1, plus an atom asking for a specific metaproperty of the starting node of the chain.

Tables 11.3 and 11.4 summarize the performances of query answering using LMG and NAIVE, respectively. For every pair $\langle Ont_i, Q_j \rangle$, the performance of query answering is expressed in terms of both the time needed to compute the certain answers to $Q_j$ over $Ont_i$, and the number of metaground queries sent to MASTRO. From the analysis of the experiments results, one can derive three main observations.

First, LMG largely outperforms NAIVE for every test query, over every ontology. The values reported in the tables point out the smaller amount of metaground queries that LMG requires to evaluate. The only exception is $Q_0$. In fact, looking at the structure of $Q_0$ one can easily notice that answering $Q_0$ by LMG is reduced to the evaluation of the same set of metaground queries of NAIVE. It is our opinion that these results supports the conclusion that MQ-MASTRO is the first system making metaquerying practical over ontologies expressed in (fragments of) OWL 2.

Second, we notice that while for a given query, both using LMG and NAIVE, the time of evaluation obviously increases with the size of the ontology, the number of metaground queries to be evaluated does not change moving from $Ont_0$ to $Ont_1$, or from $Ont_2$ to $Ont_3$. This can be explained by the fact that $Ont_1$ and $Ont_3$ are respectively obtained from $Ont_0$ and $Ont_2$ by adding ABox axioms introducing new individuals, which do not occur as predicates in $O$. Clearly, this addition has no impact on the number of metaground queries.

### Table 11.1. Test ontologies features

| $O$ | $V_C^O$ | $|Mt-Cl|$ | $V_{OP}^O$ | $|V_{DP}^O|$ | $|Ind|$ | $|TBox\ ax.|$ | $|ABox\ ax.|$ |
|-----|----------|-------------|------------|-------------|--------|------------|-------------|
| $Ont_0$ | 92 | 40 | 5 | 6 | $\approx 26000$ | 479 | $\approx 137000$ |
| $Ont_1$ | 92 | 40 | 5 | 6 | $\approx 33000$ | 479 | $\approx 172000$ |
| $Ont_2$ | 97 | 43 | 5 | 6 | $\approx 26000$ | 542 | $\approx 137000$ |
| $Ont_3$ | 97 | 43 | 5 | 6 | $\approx 30000$ | 542 | $\approx 157000$ |
Table 11.2. Test queries

<table>
<thead>
<tr>
<th>Query</th>
<th>SQL Query</th>
</tr>
</thead>
</table>
| Q0    | `SELECT $x $y WHERE { ClassAssertion($y $x))} |}
| Q1    | `SELECT $x $y $z WHERE { ClassAssertion($z $y). ClassAssertion($y $x))} |}
| Q2    | `SELECT $x $y $z $w WHERE { ClassAssertion($w $z). ClassAssertion($z $y). ClassAssertion($y $x))} |}
| Q3    | `SELECT $x $y WHERE { ClassAssertion(:Type_of_f_i $y). ClassAssertion($y $x))} |}
| Q4    | `SELECT $x $y $z WHERE { ClassAssertion(:Category_of_type_of_f_i $z). ClassAssertion($z $y). ClassAssertion($y $x))} |}
| Q5    | `SELECT $x $y $z $w WHERE { SubClassOf($w :Category_of_type_of_f_i). ClassAssertion($w $z). ClassAssertion($z $y). ClassAssertion($y $x))} |}
| Q6    | `SELECT $x $y WHERE { ClassAssertion(:Category_of_type_of_f_i_wrt_market $w). ClassAssertion(:Category_of_type_of_f_i_wrt_pay_mode $z). ClassAssertion($w $y). ClassAssertion($z $y). ClassAssertion($y $x))} |}
| Q7    | `SELECT $y $x $z WHERE { ClassAssertion(:Type_of_Italian_f_i $y). ClassAssertion($y $x). DataPropertyAssertion(:duration $x $z)} |}
| Q8    | `SELECT $x $y WHERE { ClassAssertion($z $x). ObjectPropertyAssertion(:ruled_by $z $y)} |}

to be evaluated, still having impact on the evaluation time of some of such metagroundings, and, hence, on the overall evaluation time. On the other hand, we notice that, when moving from Ont0 to Ont2, the number of metaground queries increases much more for Naive than for LMG, and that, for every query Qi, the increase is exponential in the number of metavariables of Qi. This can be explained by the observation that Ont2 is obtained from Ont0 by adding five classes, and this addition has much more impact on Naive than on LMG, since Naive blindly uses the new classes to instantiate variables in class position.

Third, for a given ontology, the performances of LMG vary depending on the position of variables within the query. For example, despite the similarity of queries Q2 and Q5, we notice that the performances of answering the two queries significantly differ using LMG, while they coincide using Naive. To explain why, let us introduce the notion of witnessed metavariable in a query. Given a query Q and a variable $x$, we say that $x$ is a witnessed metavariable in Q, if $x$ occurs as a metavariable in some atoms of Q and at least one of
### 11.3 Evaluation

<table>
<thead>
<tr>
<th>Query</th>
<th>(Q_0)</th>
<th>(Q_1)</th>
<th>(Q_2)</th>
<th>(Q_3)</th>
<th>(Q_4)</th>
<th>(Q_5)</th>
<th>(Q_6)</th>
<th>(Q_7)</th>
<th>(Q_8)</th>
</tr>
</thead>
<tbody>
<tr>
<td># of MG queries</td>
<td>92</td>
<td>320</td>
<td>640</td>
<td>23</td>
<td>161</td>
<td>229</td>
<td>32</td>
<td>18</td>
<td>21</td>
</tr>
<tr>
<td>Time (s)</td>
<td>2.7</td>
<td>31</td>
<td>31</td>
<td>0.6</td>
<td>1.8</td>
<td>1.9</td>
<td>0.13</td>
<td>0.43</td>
<td>0.19</td>
</tr>
</tbody>
</table>

**Table 11.3. Performance of LMG algorithm**

<table>
<thead>
<tr>
<th>Query</th>
<th>(Q_0)</th>
<th>(Q_1)</th>
<th>(Q_2)</th>
<th>(Q_3)</th>
<th>(Q_4)</th>
<th>(Q_5)</th>
<th>(Q_6)</th>
<th>(Q_7)</th>
<th>(Q_8)</th>
</tr>
</thead>
<tbody>
<tr>
<td># of MG queries</td>
<td>92</td>
<td>640</td>
<td>23</td>
<td>161</td>
<td>229</td>
<td>32</td>
<td>18</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>Time (s)</td>
<td>2.7</td>
<td>31</td>
<td>0.6</td>
<td>1.8</td>
<td>1.9</td>
<td>0.13</td>
<td>0.43</td>
<td>0.19</td>
<td></td>
</tr>
</tbody>
</table>

**Table 11.4. Performance of NAIVE algorithm**

the following is verified:

- \(\$x\) occurs in individual position in some other atom of \(Q\);
- \(\$x\) occurs in at least one TBox atom of \(Q\), and such atom is used to connect \(\$x\) with an entity of the ontology.

Intuitively, a metavariable is witnessed if the query contains some atom that LMG can exploit to obtain suitable instantiations for such metavariable. For example, coming back to queries \(Q_2\) and \(Q_3\) in Table 11.2, the metavariable \(\$w\) is witnessed in \(Q_5\) whereas it is not in \(Q_2\). In Table 11.5 the number of metavariables and the number of witnessed metavariables occurring in each test query are shown. Combining the information of Table 11.5 and Table 11.3 one can easily verify that, for a fixed ontology, the worst performances of LMG coincide with the evaluation of queries where some non-witnessed metavariables occur. Let us consider again \(Q_2\) and \(Q_5\). We point out that the test ontologies are such that the evaluation of \(Q_2\) and \(Q_5\) provides the same results. However, LMG performs much worse for \(Q_2\) than for \(Q_5\), due to the blind instantiation of the metavariable \(\$w\), which leads to the...
Table 11.5. Test queries (witnessed) metavariables

<table>
<thead>
<tr>
<th></th>
<th>$Q_0$</th>
<th>$Q_1$</th>
<th>$Q_2$</th>
<th>$Q_3$</th>
<th>$Q_4$</th>
<th>$Q_5$</th>
<th>$Q_6$</th>
<th>$Q_7$</th>
<th>$Q_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td># of metavariables</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td># of witnessed metavariables</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

evaluation of several metaground queries that, in fact, are not useful in retrieving answers to $Q_2$. Clearly, since NAIVE does not follow any strategy to instantiate metavariables, $Q_2$ and $Q_5$ lead to exactly the same set of metaground queries, and thus require the same time for their evaluation.
Chapter 12

Conclusion and future work

In this dissertation we have studied several issues related to the problem of metamodeling and metaquerying in the context of the semantic web. The importance of metaquerying in such a context is witnessed by the fact that OWL 2 has a native syntactic support, called punning, by which the same identifier can be used to denote ontology elements of different categories, such as a class and an individual. However, the official model-theoretic semantics of OWL 2, the so-called Direct Semantics, treats punning in a way that is not adequate for metamodeling, as such a semantics sanctions that an individual and a class with the same identifier are actually different elements. Thus we have presented the MMS semantics, that is, an Hilog-style semantics where every object in an interpretation can be polymorphic, in the sense that it can be seen simultaneously as an individual, a class and a relation. The novel aspect of MMS with respect to the pre-existing Hilog-style semantics is the partiality of the interpretation functions it is equipped with. Indeed, in all these semantics, a total function \( ext \) is defined on the interpretation domain, in such a way that \( ext(o) \) is the set of instances of an object \( o \) when seen as a class (analogous extension functions specify the instances of \( o \) when seen as a relation), whereas in our semantics such functions are not total and we have shown that this property is crucial for obtaining decidability and tractability results for the query answering problem. We applied the MMS semantics to interpret standard OWL 2 QL ontologies with metamodeling features and to define a new SPARQL entailment regime enabling the evaluation of a broad range of metaqueries over such ontologies.

We have carried out an extensive analysis of the computational complexity of the MMS semantics for OWL 2 QL, with a particular emphasis on the (meta)query answering problem. We have shown that the problem of checking satisfiability of a \( \mathcal{H}(\text{OWL 2 QL}) \) ontology is \( \mathsf{AC}^0 \) w.r.t. ABox complexity and \( \mathsf{PTIME} \) w.r.t. ontology complexity, thus matching the complexity of ontology satisfiability for OWL 2 QL. Motivated primarily by the prospects of reusing the existing highly optimized standard OWL 2 QL reasoners, we developed several algorithms for query answering based on the metagrounding technique, by which the problem of computing the answers to a metaquery is reduced to compute the answers to all the standard first-order queries that is possible to obtain by suitably instantiating its metavariables. In particular we have shown that answering instance queries (i.e., metaqueries with ABox atoms only where variables in predicate position may possibly occur) over general \( \mathcal{H}(\text{OWL 2 QL}) \) ontologies has the same complexity of answering standard first-order conjunctive queries over OWL 2 QL ontologies, that is, \( \mathsf{AC}^0 \) w.r.t. ABox complexity, \( \mathsf{PTIME} \) w.r.t. ontology complexity and \( \mathsf{NP} \)-complete w.r.t. combined complexity.
On the contrary, we have shown that for general queries, (i.e., metaqueries containing both ABox and TBox atoms) the metagrounding technique provides a sound a complete method only for the case of the so-called TBox-complete ontologies, where, in a nutshell, one does not have uncertainty about the negative axioms of the ontology, in the sense that for every possible negative statement (such as two classes are disjoint) about classes and relations of the ontology, either such a statement or its negation is logically implied by the ontology. As for the case of TBox-complete ontologies, we developed an algorithm, called Naïve, showing that answering general queries over such ontologies is \(AC^0\) w.r.t. ABox complexity, PTIME w.r.t. ontology complexity and NP-complete w.r.t. combined complexity. Then we moved to study the complexity of query answering for the case of non-TBox-complete ontologies, and we proved show that answering general queries is in this case \(AC^0\) w.r.t. data complexity, coNP-complete w.r.t. to ontology complexity and \(\Pi_2^P\)-complete w.r.t. combined complexity, thus providing the first decidability (even tractability w.r.t. data complexity) results for general metaqueries posed to ontologies expressed in (fragments of) OWL 2. Furthermore, based on the observation that are several ontologies used in practice (such as those derived by Entity-Relationship model or UML diagrams) that are TBox-complete, we have devised an optimized sound and complete algorithm, called LMG, to overcome the computational cost of Naïve, whose main idea is to replace the blind instantiation process behind metagrounding by a more clever strategy that, guided by the structure of the considered metaquery, considers only those instantiations that are potentially of interest for producing query results. To show that our approach is practically relevant, we have implemented our algorithms into a reasoning system MQ-Mastro that is, for the best of our knowledge, the first system allowing to perform the kind of proper metaquerying that we have described in this thesis over ontologies expressed in (fragments of) OWL 2. Then we conducted a performance evaluation of the system, and our measurements showed that LMG algorithm can be used to solve the metaquerying problem in real world application scenarios.

We plan to continue our work along several directions. We aim at studying interesting subclasses of queries, and characterize their complexity. For example, we already know that answering queries whose TBox atoms are all ground is PTIME in ontology complexity, exactly like in [33]. We also plan to increase the expressive power of \(Hi(\text{OWL 2 QL})\) by enlarging both the ontology and the query language in several ways. As for the ontology language we plan, for example, to consider mechanisms for expressing integrity constants by which one can prevent certain ontology elements to play some roles into an ontology (for instance, by stating that a given element can be only interpreted as a class and not as an individual), and to allow a restricted use of some of the built-in predicates, such as for example the built-in predicate \(\text{rdf:type}\) that is used to formalize \textit{instanceOf} relations between ontology elements, as argument of ontology axioms, so as to define, for example, a particular notion of \textit{instanceOf} that is local to the considered ontology (e.g., by introducing, for instance, in the ontology alphabet an element identified by \(\text{my:type}\) and by stating that the couples of elements connected \(\text{my:type}\) are a subset of those connected by \(\text{rdf:type}\)). As for the query language we plan to enlarge it syntax by allowing, for example, the use of SPARQL operators such as \texttt{Optional} and \texttt{Filter}) while keeping the same computational complexity properties. Furthermore, we aim at studying further optimization strategies for answering metaqueries over TBox-complete ontologies, targeted toward storage structures for both the TBox and the ABox different from the one considered here. Finally, an important theoretical challenge is to study the metaquerying problem
for OWL 2 profiles, such as the EL profile, where one cannot relies on the metagrounding technique because of the infiniteness of the set of expressions that can be built starting from any ontology vocabulary.
Bibliography


