Ontology-Mediated Query Answering with Data-Tractable Description Logics*

Meghyn Bienvenu[1] and Magdalena Ortiz[2]

[1] LRI - CNRS & Université Paris Sud
meghyn@lri.fr
[2] Institute of Information Systems, Vienna University of Technology
ortiz@kr.tuwien.ac.at

Abstract. Recent years have seen an increasing interest in ontology-mediated query answering, in which the semantic knowledge provided by an ontology is exploited when querying data. Adding an ontology has several advantages (e.g. simplifying query formulation, integrating data from different sources, providing more complete answers to queries), but it also makes the query answering task more difficult. In this chapter, we give a brief introduction to ontology-mediated query answering using description logic (DL) ontologies. Our focus will be on DLs for which query answering scales polynomially in the size of the data, as these are best suited for applications requiring large amounts of data. We will describe the challenges that arise when evaluating different natural types of queries in the presence of such ontologies, and we will present algorithmic solutions based upon two key concepts, namely, query rewriting and saturation. We conclude the chapter with an overview of recent results and active areas of ongoing research.

1 Introduction

Since the seminal works in the field [144, 53, 110, 50], there has been steadily growing interest in *ontology-mediated query answering* (OMQA), in which the semantic knowledge provided by an ontology is exploited when querying data. Adding an ontology has several advantages. First, by providing an enriched vocabulary that closely matches users' conceptualization of the application domain, an ontology makes it easier for users to formulate their queries. Moreover, the ontology can be used to integrate different data sources through a single conceptual model, facilitating access to them in a uniform and transparent way. Finally, OMQA can provide users with more complete answers to their queries, by taking into account not only the facts explicitly stored in the data, but also facts that are implicit consequences of the data and the domain knowledge. Unfortunately, enriching data with domain knowledge also has a downside: it makes the query

* This work has been supported by ANR project PAGODA (ANR-12-JS02-007-01) and the Austrian Science Fund (FWF) project T515.

answering task significantly more difficult, both conceptually and algorithmically. The specific challenges that arise depend upon which languages are used for expressing the query and the ontological knowledge.

In this chapter, we consider ontologies formulated using *description logics (DLs)*, which are a family of decidable fragments of classical first-order predicate logic that are often used for knowledge representation and reasoning. DLs are arguably the most popular formalisms for representing ontological knowledge nowadays, notably providing the logical underpinnings for the W3C-standardized OWL web ontology languages [170]. There are a multitude of different DLs of varying expressivity, ranging from very simple to highly expressive. Significant research efforts have been devoted to understanding the computational complexity of different reasoning tasks, including the query answering tasks that are the focus of this chapter. The resulting complexity landscape can be used to select the most appropriate DL for a given application. In the case of OMQA applications involving large amounts of data, this complexity analysis has revealed *Horn DLs* – so named because they are expressible in the Horn fragment of first-order logic – as especially relevant, as query answering in the presence of Horn DL ontologies can be performed in polynomial time in the size of the data, for some important types of queries. Prominent Horn DLs include the logics of the DL-Lite and $\mathcal{EL}$ families, which are the basis of the OWL profiles known as OWL 2 QL and OWL 2 EL [159].

This chapter is organized as follows. We begin in Section 2 by recalling the syntax and semantics of description logic knowledge bases and introducing some popular Horn DLs. In Section 3, we formally introduce the problem of ontology-mediated query answering and compare it to the closely related problem of querying relational databases. We also explain how we will measure the complexity of query answering and briefly introduce the two main algorithmic techniques (*query rewriting* and *saturation*) that underlie most of the querying algorithms that have been proposed for Horn DL ontologies. The following three sections are devoted to different query languages: *instance queries* in Section 4, *conjunctive queries* in Section 5, and *navigational queries* in Section 6. In each of these sections, we will illustrate the kinds of natural queries that can be expressed in the query languages, describe the challenges that arise when evaluating them in the presence of ontologies, present algorithmic solutions involving query rewriting and/or saturation, and summarize what is known about the complexity of the query answering task for different DLs. In Section 7, we show the difficulties that arise when querying DL knowledge bases using more expressive query languages involving negation or recursion. The final section of this chapter provides an overview of recent work on OMQA and areas of ongoing research.

This chapter aims to provide a relatively detailed introduction to the area of ontology-mediated query answering with Horn DL ontologies. We chose to focus on Horn DLs in order to showcase the versatility of query rewriting and saturation techniques for handling a variety of different types of queries, including navigational queries that have only recently been considered for OMQA. Although we briefly discuss results and techniques for non-Horn DLs and try

to give a relatively complete picture of the complexity landscape, the present chapter should by no means be considered a comprehensive survey of the field. For a detailed treatment of OMQA as it relates to more expressive DLs, we refer to [168, 163] and references therein. For introductions to OMQA that focus on DL-Lite and the corresponding OWL 2 QL profile and provide more details on the use of database systems, we direct readers to the tutorials [128, 48].

2 Horn Description Logics

In this section, we give a short introduction to description logics and how they are used for describing ontological knowledge. We try to keep it concise, since extensive introductory texts on the topic have been published elsewhere. Readers less familiar with DLs may find useful the long and detailed introductions in [190], and in the first part of [168], or the short basic overview in [137].

2.1 Description Logic Basics

In description logics, a domain of interest is described using a *DL vocabulary* consisting of three countably infinite, pairwise disjoint sets of symbols:

- the set $\mathsf{N_C}$ of *concept names*, to capture *classes* of objects
- the set $\mathsf{N_R}$ of *role names*, to capture *binary relations* between objects
- the set $\mathsf{N_I}$ of *individual names* (often abbreviated to *individuals*), to refer to specific individual objects

Note that a DL vocabulary can be seen as a restricted first-order logic (FO) vocabulary containing only unary predicates (concept names), binary predicates (role names), and constants (individual names).

From a DL vocabulary, we can build expressions that reflect the knowledge about our domain. In general, we use two kinds of statements:

- *Terminological axioms* specify general properties of concepts and roles, and constrain the way *all* objects in the domain can participate in the different concepts and roles.
- *Assertions* are facts about *specific* objects in the domain, that is, they assert that an individual participates in some concept, or that some role holds between a pair of individuals.

Each DL offers a different syntax for the terminological axioms and different combinations of *concept constructors* and *role constructors* that allow us to build *complex concept and roles* from the symbols in the vocabulary. Table 1 summarizes some DL concept and role constructors, as well as the most common forms of axioms and assertions.

As the inverse role constructor occurs in many of the DLs considered in this chapter, we introduce some dedicated notation and terminology. We use $\mathsf{N_R^\pm}$ to denote the set $\mathsf{N_R} \cup \{r^- \mid r \in \mathsf{N_R}\}$ and use the generic term *role* to refer to elements of $\mathsf{N_R^\pm}$. We define the *inverse* $\mathsf{inv}(R)$ of a role as follows: if R is a role name $r \in \mathsf{N_R}$, then $\mathsf{inv}(R) = r^-$, and if R is of the form r^-, then $\mathsf{inv}(R) = r$.

Name	Syntax	Semantics	
Top concept	$\top$	$\Delta^{\mathcal{I}}$	Concepts
Bottom concept	$\bot$	$\emptyset$	
Nominal	$\{a\}$	$\{a^{\mathcal{I}}\}$	
Negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	
Conjunction	$C_1 \sqcap C_2$	${C_1}^{\mathcal{I}} \cap {C_2}^{\mathcal{I}}$	
Disjunction	$C_1 \sqcup C_2$	${C_1}^{\mathcal{I}} \cup {C_2}^{\mathcal{I}}$	
Existential restriction	$\exists R.C$	$\{d_1 \mid \text{there exists } (d_1, d_2) \in R^{\mathcal{I}} \text{ with } d_2 \in C^{\mathcal{I}}\}$	
Universal restriction	$\forall R.C$	$\{d_1 \mid d_2 \in C^{\mathcal{I}} \text{ for all } (d_1, d_2) \in R^{\mathcal{I}}\}$	
(Qualified) number restrictions	$\geqslant m\, R.C$	$\{d_1 \mid m \leq \lvert\{d_2 \mid (d_1, d_2) \in R^{\mathcal{I}} \text{ and } d_2 \in C^{\mathcal{I}}\}\rvert\}$	
	$\leqslant m\, R.C$	$\{d_1 \mid m \geq \lvert\{d_2 \mid (d_1, d_2) \in R^{\mathcal{I}} \text{ and } d_2 \in C^{\mathcal{I}}\}\rvert\}$	
Inverse	r^-	$\{(d_2, d_1) \mid (d_1, d_2) \in r^{\mathcal{I}}\}$	Roles
Role negation	$\neg R$	$(\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}) \setminus R^{\mathcal{I}}$	
Concept inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$	TBox Axioms
Role inclusion	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$	
Transitivity axiom	$\mathsf{trans}(R)$	$R^{\mathcal{I}} \circ R^{\mathcal{I}} \subseteq R^{\mathcal{I}}$	
Concept assertion	$A(a)$	$a^{\mathcal{I}} \in A^{\mathcal{I}}$	ABox Assertions
Role assertion	$r(a, b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$	

Table 1: Syntax and semantics of DL concept and role constructors, TBox axioms, and ABox assertions. Here a, b denote individual names, A denotes a concept name, $C_{(i)}$ denotes a (complex) concept, $r \in \mathsf{N_R}$ denotes a role name, R denotes a role, and $m \in \mathbb{N}$ denotes a natural number.

Definition 1. *A* TBox *is a finite set of terminological axioms and an* ABox *is a finite set of assertions. A* knowledge base (KB) $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ *is composed of a TBox* $\mathcal{T}$ *and an ABox* $\mathcal{A}$*.*

A signature *is a set of concept and role names, and the* signature of a TBox $\mathcal{T}$*, written* $\mathsf{sig}(\mathcal{T})$*, is the set of concept names and role names that occur in* $\mathcal{T}$*. Signatures of ABoxes and KBs are defined and denoted analogously. Finally, for a given signature* Σ*, we say that an ABox* $\mathcal{A}$ *is a* Σ-ABox *if* $\mathsf{sig}(\mathcal{A}) \subseteq \Sigma$*.*

Example 1. For the examples in this chapter, we will consider the domain of food, dishes and menus offered by restaurants. The vocabulary we use to model this domain contains concept names for food items, like IceCream or Meat, and for more general types of food, such as vegetarian-friendly options (VegFriendly) or spicy dishes (SpicyDish). We also use concept names for notions like Restaurant, Menu and Dish. The role name hasIngredient is used to relate dishes and their ingredients, and contains is a generalization (or *superrole*) of hasIngredient that

can also relate foods with components (such as lactose or gluten) that would not typically be considered as ingredients. The role name hasCourse relates menus to the dishes they contain as courses, and we may also have specialized versions of this role like hasDessert and hasMain. We can also use role names to say that a restaurant offers some menu, or that it serves a dish. For individuals that represent specific menus, dishes, and restaurants, we use italic, lower-case letters.

With this vocabulary in place, we can write ABox assertions such as:

$$\begin{array}{lll} \mathsf{offers}(r, m) & \mathsf{hasMain}(m, p_1) & \mathsf{PenneArrabiata}(p_1) \\ \mathsf{hasDessert}(m, d_1) & \mathsf{IceCream}(d_1) & \mathsf{serves}(r, p_2) \\ \mathsf{PizzaCalabrese}(p_2) & \mathsf{serves}(r, d_2) & \mathsf{Tiramisu}(d_2) \end{array}$$

which intuitively express that some restaurant (r) offers a menu (m) with penne arrabiata and ice cream, and it also serves pizza calabrese and tiramisu.

We give some examples of TBox axioms that express general knowledge about this domain. Here $C \equiv D$ is shorthand for the pair of axioms $C \sqsubseteq D$ and $D \sqsubseteq C$.

$$\exists \mathsf{hasCourse}.\top \sqsubseteq \mathsf{Menu} \tag{1}$$

$$\exists \mathsf{hasCourse}^{-}.\top \sqsubseteq \mathsf{Dish} \tag{2}$$

$$\mathsf{hasDessert} \sqsubseteq \mathsf{hasCourse} \tag{3}$$

$$\mathsf{hasMain} \sqsubseteq \mathsf{hasCourse} \tag{4}$$

$$\mathsf{Menu} \sqsubseteq \leqslant 1\, \mathsf{hasMain}.\top \tag{5}$$

$$\mathsf{FullMenu} \equiv \geqslant 3\, \mathsf{hasCourse}.\top \tag{6}$$

$$\mathsf{PizzaCalabrese} \sqsubseteq \mathsf{Pizza} \sqcap \exists \mathsf{hasIngredient}.\mathsf{PizzaDough} \tag{7}$$

$$\mathsf{PenneArrabiata} \sqsubseteq \exists \mathsf{hasIngredient}.\mathsf{Pasta} \tag{8}$$

$$\mathsf{PizzaDough} \sqcup \mathsf{Tiramisu} \sqcup \mathsf{Pasta} \sqsubseteq \exists \mathsf{contains}.\mathsf{Gluten} \tag{9}$$

$$\mathsf{GlutenFree} \equiv \forall \mathsf{contains}.\neg \mathsf{Gluten} \tag{10}$$

$$\mathsf{hasIngredient} \sqsubseteq \mathsf{contains} \tag{11}$$

$$\mathsf{trans}(\mathsf{contains}) \tag{12}$$

The concept inclusions (1) and (2) state respectively that the domain of hasCourse consists of menus, and its range consists of dishes. The role inclusions (3) and (4) express that hasDessert and hasMain are specializations (or *subroles*) of the role hasCourse, since desserts and mains are types of courses. Concept inclusion (5) stipulates that a menu can only have one main course. The axiom (6) defines full menus as menus that have at least three courses. Axiom (7) states that pizza calabrese is a kind of pizza that has as ingredient pizza dough. The following axiom (8) says that penne arriabiata has pasta as an ingredient. In (9), it is stated that pizza dough, tiramisu and pasta all contain gluten, and (10) defines the gluten-free as the class of entities not containing gluten. The role inclusion (11) expresses that hasIngredient is a subrole of contains, and (12) asserts the transitivity of the relation contains. ▲

There are a wide range of DLs offering different shapes of axioms and different concept and role constructors. For example, the well-known description

logic $\mathcal{ALC}$ allows only for concept inclusions $C \sqsubseteq D$ as TBox axioms, where C and D are complex concepts built using negation, conjunction, disjunction, existential restrictions, and universal restrictions, from the *atomic concepts* that include concept names, top and bottom. The DL $\mathcal{S}$ extends $\mathcal{ALC}$ with transitivity axioms. The presence of additional constructors or axiom types is denoted by additional letters in the name of the logics. For example, the letter $\mathcal{H}$ denotes the presence of role inclusions in the TBox. The letter $\mathcal{I}$ denotes that inverse roles can be used as a role constructor in the TBox axioms, $\mathcal{O}$ denotes the presence of nominals as a concept constructor, and $\mathcal{Q}$ denotes the presence of qualified number restrictions. In this way, we obtain a large number of different DLs like $\mathcal{ALCI}$, $\mathcal{ALCHQ}$, $\mathcal{SHIQ}$, $\mathcal{SHOIQ}$, and so on[3]. We note that the knowledge base in Example 1 is a $\mathcal{SHIQ}$ knowledge base.

Before moving on to the semantics of DLs, a small remark is in order concerning the syntax of ABox assertions. For a DL $\mathcal{L}$, an $\mathcal{L}$ ABox is sometimes defined as a set of assertions of the forms $C(a)$ and $R(a, b)$, where C and R are *possibly complex* concepts and roles in $\mathcal{L}$. In this chapter, for simplicity, ABox assertions take the forms $A(a)$ and $r(a, b)$ only, independently of the DL in question. For our purposes, this simplification is without any loss of generality. Indeed, it is well known that complex assertions $C(a)$ can be replaced by assertions $A_C(a)$ for a fresh concept name A_C, provided that $A_C \sqsubseteq C$ is added to the TBox. Role assertions $r^-(a, b)$ can be replaced by $r(b, a)$, and as inverses are the only role constructor in almost all DLs we consider (the only exception is DL-Lite, discussed further), this is the only kind of complex assertions that could occur. It will be clear from what follows that these transformations preserve the semantics of KBs and that they have no impact on the computational complexity of any of the reasoning and query answering problems we consider.

2.2 Semantics

The semantics of DLs is defined using the notion of interpretations.

Definition 2 (Interpretation, models). *An* interpretation $\mathcal{I}$ *is a pair* $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *where* $\Delta^{\mathcal{I}}$ *is a non-empty set called the* domain*, and* $\cdot^{\mathcal{I}}$ *is an interpretation function that maps:*

- *each concept name* $A \in \mathsf{N_C}$ *to a set* $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$*,*
- *each role name* $r \in \mathsf{N_R}$ *to a set of pairs* $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$*, and*
- *each individual* $a \in \mathsf{N_I}$ *to some* $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$*, in such a way that* $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ *whenever* $a \neq b$*.*

The interpretation function is extended to complex concept and roles as specified in the upper right portion of Table 1.

[3] The order of the letters is irrelevant, although some orderings are more frequent in the literature than others, e.g., $\mathcal{SHIQ}$, vs. $\mathcal{SHQI}$. We also point out that some DLs impose additional restrictions, for example, restricting the interaction of number restrictions and transitive roles in $\mathcal{SHIQ}$ and $\mathcal{SHOIQ}$.

Note that this is essentially the traditional notion of interpretation from first-order logic, but restricted to unary and binary predicates, and constants.

Using interpretations, we can define the notions of models, satisfiability, and entailment. The *satisfaction* $\mathcal{I} \models \xi$ of a TBox axiom or ABox assertion ξ in an interpretation $\mathcal{I}$ is defined in the lower right part of Table 1. An interpretation $\mathcal{I}$ is called a *model of a TBox* $\mathcal{T}$, written $\mathcal{I} \models \mathcal{T}$, if $\mathcal{I} \models \xi$ for every axiom ξ in$\mathcal{T}$. Similarly, $\mathcal{I}$ is a *model of an ABox* $\mathcal{A}$, written $\mathcal{I} \models \mathcal{A}$, if $\mathcal{I} \models \xi$ for every assertion ξ in $\mathcal{A}$. If both $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \models \mathcal{A}$, then we call $\mathcal{I}$ a *model of the KB* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, and we write $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$ (or, $\mathcal{I} \models \mathcal{K}$). We call a KB $\mathcal{K}$ *satisfiable* (or *consistent*) if it has at least one model. Entailment is defined in the expected way: a TBox axiom or ABox assertion ξ is said to be *entailed* from a KB $\mathcal{K}$ (in symbols: $\mathcal{K} \models \xi$) if $\mathcal{I} \models \xi$ for every model $\mathcal{I}$ of $\mathcal{K}$. We can also define the corresponding notions of entailment w.r.t. TBoxes and ABoxes: $\mathcal{T} \models \xi$ if $(\mathcal{T}, \emptyset) \models \xi$, and $\mathcal{A} \models \xi$ if $(\emptyset, \mathcal{A}) \models \xi$.

Remark 1. Note that, by definition, interpretations give meaning to all the symbols in the infinite vocabulary $\mathsf{N_C} \cup \mathsf{N_R} \cup \mathsf{N_I}$. Alternatively, interpretations can be defined for a possibly finite signature that contains all the relevant symbols, including the signature of the KB at hand. Since the interpretation of all symbols not occurring in a KB or query is irrelevant to their satisfaction, both semantics are equivalent. Moreover, in definitions and examples, we will allow interpretations to be (finitely) specified for the relevant signature only, and disregard the interpretation of all irrelevant symbols.

Remark 2. Observe that in Definition 2 we require distinct constants to be interpreted as different objects in $\Delta^{\mathcal{I}}$. That is, we make the *unique name assumption (UNA)*. This assumption is sometimes made in DLs, and sometimes not. We have chosen to adopt the UNA because it is closer to the intended semantics of ABoxes as data repositories, and hence more natural for the OMQA setting we consider. We should emphasize that *this assumption is not central to the results and techniques presented in this chapter*, which are valid both with or without the UNA. It can be noted however that there do exist cases, not covered in this chapter, in which the complexity of query answering depends on whether the UNA is adopted (see e.g., [10]).

2.3 Some Popular Horn Description Logics

In this chapter, we focus on a specific class of DLs known as *Horn DLs*, whose core feature is that they are incapable of expressing any form of disjunction. This lack of disjunction means that Horn DL knowledge bases can be translated into the Horn fragment of first-order logic.

We first introduce two important sub-families of Horn DLs based upon the 'lightweight' logics DL-Lite and $\mathcal{EL}$, which support efficient reasoning at the cost of limited expressiveness.

DL-Lite Family The constructors available in the different DLs of the DL-Lite family were selected in order to express the main features present in conceptual and data models, like ISA-relations between classes, class disjointness, domain and range restrictions on roles, functionality constraints, and mandatory (non-)participation constraints. This makes (large fragments of) the formalisms used in databases and software engineering, like entity-relationship and UML class diagrams, expressible as DL-Lite knowledge bases. At the same time, the DL-Lite family (first proposed in [50, 52]) was designed to support efficient reasoning in data-oriented applications, even in the presence of large amounts of data. Due to their carefully tailored expressivity and good computational properties, DLs of the DL-Lite family have become extremely popular as ontology languages. This can be witnessed by the recent inclusion of the OWL 2 QL profile [159], based upon DL-Lite, in the latest version of the OWL standard.

In the DL-Lite family, there are no universal restrictions, and existential restrictions can only be of the form $\exists R.\top$ and are thus abbreviated to $\exists R$. The basic DL-Lite dialect, sometimes denoted DL-Lite$_{core}$, only allows *concept inclusions* of the forms $B_1 \sqsubseteq B_2$ and $B_1 \sqsubseteq \neg B_2$, where each B_i is either a concept name or an existential restriction $\exists R$ with $R \in \mathsf{N}_\mathsf{R}^{\pm}$. We note that *negative concept inclusions* of the form $B_1 \sqsubseteq \neg B_2$ can also be written as *(concept) disjointness constraints* $B_1 \sqcap B_2 \sqsubseteq \bot$; both syntaxes are widely used. Axioms of the form $\exists r \sqsubseteq B$ are often called *domain restrictions*, since they enforce that the domain of role r is contained in the concept B; similarly, axioms of the form $\exists r^- \sqsubseteq B$ are called *range restrictions*. In DL-Lite, it is also common to allow negative concept and role assertions $\neg B(a)$ and $\neg R(a, b)$ in the ABox. We do not allow them explicitly here, since $\neg B(a)$ can be simulated using an assertion $\bar{B}(a)$ for a fresh concept name $\bar{B}(a)$, and adding $\bar{B} \sqsubseteq \neg B$ to the TBox. Likewise, $\neg R(a, b)$ can be simulated via $\bar{R}(a, b)$ and $\bar{R} \sqsubseteq \neg R$, where $\bar{R}$ fresh role name.

One of the most popular dialects of DL-Lite is DL-Lite$_\mathcal{R}$, which additionally allows for role inclusions of the forms $R \sqsubseteq S$ and $R \sqsubseteq \neg S$, where $R, S \in \mathsf{N}_\mathsf{R}^{\pm}$. We will focus on DL-Lite$_\mathcal{R}$ when discussing the DL-Lite family in this chapter. Another prominent dialect of DL-Lite is DL-Lite$_\mathcal{F}$, which extends DL-Lite by allowing axioms of the form (funct P), which are just an abbreviation of $\top \sqsubseteq \leqslant 1\, R.\top$. There are a great many other DL-Lite dialects that have been considered, see [10] for a detailed discussion.

Example 2. Among the axioms in Example 1, only (1) and (2) are expressible in the core dialect of DL-Lite. They would usually be written as follows:

$$\exists \mathsf{hasCourse} \sqsubseteq \mathsf{Menu} \qquad \exists \mathsf{hasCourse}^- \sqsubseteq \mathsf{Dish}$$

In DL-Lite$_\mathcal{R}$, we can also have (3), (4) and (11), and we can simulate (7) using an additional role name hasIngredientPizzaDough as follows:

$$\begin{aligned} \mathsf{PizzaCalabrese} &\sqsubseteq \mathsf{Pizza} \\ \mathsf{PizzaCalabrese} &\sqsubseteq \exists \mathsf{hasIngredientPizzaDough} \\ \exists \mathsf{hasIngredientPizzaDough}^- &\sqsubseteq \mathsf{PizzaDough} \\ \mathsf{hasIngredientPizzaDough} &\sqsubseteq \mathsf{hasIngredient} \end{aligned}$$

We can simulate the existential restrictions in the right-hand-side of axioms (8) and (9) in a similar fashion, and replace the axiom with disjunction on the left-hand side by three axioms, each with one concept name. However, we cannot express qualified existential restrictions on the left-hand side of axioms, nor number restrictions as in (5) and (6), universal restrictions as in (10), or transitivity axioms.

In DL-Lite$_{\mathcal{F}}$, we have a restricted form of number restrictions, and we can express (5) using $\mathsf{func}(\mathsf{hasMain})$. ▲

$\mathcal{EL}$ Family Like DL-Lite, the $\mathcal{EL}$ description logic offers tractable reasoning at the cost of limited expressivity. The constructors offered by $\mathcal{EL}$ and its extensions [19] make them particularly well suited for medical and life science terminologies. Several important large-scale ontologies are written using DLs of the $\mathcal{EL}$ family, including the Gene Ontology [205], the NCI thesaurus [199] (a cancer ontology), and most notably, SNOMED CT, a comprehensive medical ontology[4] that is used by the health-care systems of several countries (including the US and UK).

$\mathcal{EL}$ is the fragment of $\mathcal{ALC}$ that allows for arbitrary use of concept conjunction and existential restrictions, but no negation, concept disjunction, or universal restrictions. Atomic concepts consist of concept names and $\top$, but do not include $\bot$. As $\mathcal{EL}$ cannot express any form of contradictory or negative information, $\mathcal{EL}$ KBs are trivially satisfiable. In order to be able to express disjointness of concepts, it is common to extend $\mathcal{EL}$ by allowing $\bot$ as an atomic concept. The resulting DL, which has essentially the same computational properties as plain $\mathcal{EL}$ [19], is usually denoted $\mathcal{EL}_\bot$.

Like $\mathcal{ALC}$, $\mathcal{EL}$ and $\mathcal{EL}_\bot$ can be extended with the additional features from Table 1 to obtain DLs like $\mathcal{ELH}$, $\mathcal{ELHO}$, $\mathcal{ELI}_\bot$, $\mathcal{ELHI}_\bot$, etc. Some additions, like role hierarchies, have no impact on the positive computational properties of $\mathcal{EL}$ and $\mathcal{EL}_\bot$, while others, like inverse roles, significantly increase the complexity of reasoning. A detailed discussion of the complexity of reasoning in $\mathcal{EL}$ and extensions can be found in [19, 20].

The OWL 2 EL profile [159] is based on the $\mathcal{EL}$ family, and in particular on a DL sometimes called $\mathcal{ELRO}^+$, which is in turn a variant of $\mathcal{EL}^{++}$ [19]. Both of these languages extend $\mathcal{ELH}_\bot$ with additional features like nominals and *complex role inclusion axioms* of the form $r_1 \cdot \ldots \cdot r_n \sqsubseteq r$, which intuitively mean that the pairs of objects in the composition of the r_i also belong to r. In particular, transitivity can be expressed using axioms of the form $r \cdot r \sqsubseteq r$. $\mathcal{ELRO}^+$ also allows range restrictions (without allowing inverses in general). We do not consider these logics in detail here, but we remark that OWL 2 EL corresponds to the so-called *regular* fragment of $\mathcal{ELRO}^+$, which imposes a regularity condition on the complex role inclusions to ensure that the set $\{r_1 \cdot \ldots \cdot r_n \mid \mathcal{T} \models r_1 \cdot \ldots \cdot r_n \sqsubseteq r\}$ of role chains implying a role name r can be represented using a finite automaton.

[4] `http://www.ihtsdo.org/snomed-ct`

TBox axioms	DL-Lite$_\mathcal{R}$	$\mathcal{EL}$	$\mathcal{ELHI}_\bot$	Horn-$\mathcal{SHIQ}$
$A_1 \sqcap \cdots \sqcap A_n \sqsubseteq B$	✓$(n = 1)$	✓	✓	✓
$A_1 \sqcap \cdots \sqcap A_n \sqsubseteq \bot$	✓$(n = 2)$		✓	✓
$A \sqsubseteq \exists R.B$	✓$(B = \top)$	✓$(R \in \mathsf{N_R})$	✓	✓
$\exists R.B \sqsubseteq A$	✓$(B = \top)$	✓$(R \in \mathsf{N_R})$	✓	✓
$A \sqsubseteq \leqslant 1\, R.B$				✓
$A \sqsubseteq \geqslant m\, R.B$				✓
$R \sqsubseteq S$	✓		✓	✓
$R \sqsubseteq \neg S$	✓			
$\mathsf{trans}(R)$				✓
ABox assertions				
$A(a)$	✓	✓	✓	✓
$r(a, b)$	✓	✓	✓	✓

Table 2: Syntax of normalized Horn KBs. ✓(*conds*) means that only axioms satisfying *conds* are allowed for the form in question. Here $A_{(i)}$ and B denote atomic concepts from $\mathsf{N_C} \cup \{\top\}$, while R and S are from $\mathsf{N_R} \cup \{r^- \mid r \in \mathsf{N_R}\}$.

Example 3. Of the TBox axioms in the example, (1), (7), and (8), are all in the core $\mathcal{EL}$, while (9) can be split into three $\mathcal{EL}$ axioms. $\mathcal{ELH}$ can also express (3), (4) and (11). The axiom restricting the range of $\mathsf{hasCourse}$ is expressible in $\mathcal{ELI}$ and $\mathcal{ELRO}^+$. The inclusion $\mathsf{GlutenFree} \sqsubseteq \forall \mathsf{contains}.\neg\mathsf{Gluten}$ can be equivalently expressed in $\mathcal{ELI}_\bot$ as

$$\mathsf{Gluten} \sqcap \exists \mathsf{contains}^-.\mathsf{GlutenFree} \sqsubseteq \bot$$

However, the other direction, $\forall \mathsf{contains}.\neg\mathsf{Gluten} \sqsubseteq \mathsf{GlutenFree}$ is not expressible in any DL of the $\mathcal{EL}$ family. The transitivity axiom (12) is expressible in (regular) $\mathcal{ELRO}^+$ and $\mathcal{EL}^{++}$. ▲

It is sometimes convenient to assume that $\mathcal{EL}$ and $\mathcal{ELHI}_\bot$ TBoxes are in a normal form that only allows axioms of the forms indicated in Table 2. It is well known that TBoxes can be efficiently transformed into this normal form by introducing fresh concept names. For the sake of comparison, we have also included in the table the syntax of DL-Lite$_\mathcal{R}$, assuming a similar normalization.

We introduce another important Horn DL, called Horn-$\mathcal{SHIQ}$, that is not part of the DL-Lite or $\mathcal{EL}$ families.

Horn-$\mathcal{SHIQ}$ The description logic Horn-$\mathcal{SHIQ}$ is the disjunction-free fragment of the well-known DL $\mathcal{SHIQ}$. It supports many of the expressive features of $\mathcal{SHIQ}$, like transitivity and number restrictions, but as we will see later, it is better behaved computationally. The formal definition of Horn-$\mathcal{SHIQ}$ syntax is rather complicated, since fully eliminating disjunction requires taking into

account complex interactions between constructors, and in particular, which subformulas occur implicitly under the scope of negation. The full definition can be found in [136]. Here we instead give only the definition of *normalized* TBoxes, which allow for axioms of all the forms listed in Table 2 except for $R \sqsubseteq \neg S$.[5] Additionally, in a (Horn-)$\mathcal{SHIQ}$ TBox $\mathcal{T}$, all roles R occurring in a number restriction $\geqslant n\, R.C$ or $\leqslant n\, R.C$ must be *simple*, which means that there does not exist a role S such that $\mathsf{trans}(S) \in \mathcal{T}$ and $\mathcal{T} \models S \sqsubseteq R$. We note that axioms of the form $A \sqsubseteq \forall R.B$ are usually allowed in Horn-$\mathcal{SHIQ}$. We have omitted them from our syntax, but they can be equivalently expressed as $\exists \mathsf{inv}(R).A \sqsubseteq B$.

Example 4. Most of the axioms in our example TBox are expressible in Horn-$\mathcal{SHIQ}$, with the exception of axioms (6) and (10), for which we can only express one half of the stated equivalences:

$$\mathsf{FullMenu} \sqsubseteq \geqslant 3\, \mathsf{hasCourse}.\top \qquad (6')$$

$$\exists \mathsf{contains}^{-}.\mathsf{GlutenFree} \sqcap \mathsf{Gluten} \sqsubseteq \bot \qquad (10')$$

The other halves of (6) and (10)

$$\forall \mathsf{contains}.\neg \mathsf{Gluten} \sqsubseteq \mathsf{GlutenFree} \qquad \geqslant 3\, \mathsf{hasCourse}.\top \sqsubseteq \mathsf{FullMenu}$$

cannot be expressed in Horn-$\mathcal{SHIQ}$, nor in any other Horn DL. ▲

Horn Logics and Universal Models We have mentioned that the main distinguishing feature of Horn DLs is that they can be viewed as subsets of the well-known Horn fragment of first-order logic. Semantically, the crucial property this ensures is that, for every satisfiable KB $\mathcal{K}$, there exists a *universal* model of $\mathcal{K}$ that can be homomorphically embedded into any other model. Intuitively, this model satisfies all the constraints expressed by $\mathcal{K}$ in the minimal, most general way, and it witnesses all entailments. We will present later in the chapter a concrete way of constructing such a model and explain how it can be exploited for answering different kinds of queries.

Unfortunately, this crucial property is lost in non-Horn DLs, which provide means of expressing disjunctive knowledge. The complexity results we will discuss in this chapter illustrate how the lack of a universal model has a negative impact on the complexity of reasoning.

Example 5. Consider the KB consisting of one ABox assertion and two TBox axioms:

$$\mathsf{PastaDish}(d) \qquad \mathsf{PastaDish} \sqsubseteq \mathsf{Dish} \qquad \mathsf{PastaDish} \sqsubseteq \exists \mathsf{hasIngredient}.\mathsf{Pasta}$$

The following interpretation, with $\Delta^{\mathcal{I}} = \{o_d, o_p\}$, is a model of this KB:

$$d^{\mathcal{I}} = o_d \qquad \mathsf{hasIngredient}^{\mathcal{I}} = \{(o_d, o_p)\} \qquad \mathsf{PastaDish}^{\mathcal{I}} = \{o_d\}$$
$$\mathsf{Dish}^{\mathcal{I}} = \{o_d\} \qquad \mathsf{Pasta}^{\mathcal{I}} = \{o_p\}$$

[5] $\mathcal{SHIQ}$ does not support negative role inclusions. These could be added at no computational cost, and they are expressible in extensions of $\mathcal{SHIQ}$ for which reasoning has the same complexity, like $\mathcal{ZIQ}$ [57] and the *simple* fragment of $\mathcal{SRIQ}$ [109].

This model is universal: every model of the KB contains an object interpreting d as an instance of Dish and $\mathsf{PastaDish}$, related via $\mathsf{hasIngredient}$ to some object that is an instance of Pasta. This is the minimal structure that needs to be present in an interpretation for it to be a model of the KB.

Now suppose we add the axiom $\mathsf{Pasta} \sqsubseteq \mathsf{freshPasta} \sqcup \mathsf{driedPasta}$. We exhibit two interpretations $\mathcal{I}_1$ and $\mathcal{I}_2$, with the same domain as $\mathcal{I}$, which are both models of the extended KB:

$\mathsf{hasIngredient}^{\mathcal{I}_1} = \{(o_d, o_p)\}$ $\quad$ $\mathsf{PastaDish}^{\mathcal{I}_1} = \{o_d\}$ $\quad$ $\mathsf{freshPasta}^{\mathcal{I}_1} = \{o_p\}$
$\mathsf{Dish}^{\mathcal{I}_1} = \{o_d\}$ $\quad$ $\mathsf{Pasta}^{\mathcal{I}_1} = \{o_p\}$ $\quad$ $\mathsf{driedPasta}^{\mathcal{I}_1} = \emptyset$

$\mathsf{hasIngredient}^{\mathcal{I}_2} = \{(o_d, o_p)\}$ $\quad$ $\mathsf{PastaDish}^{\mathcal{I}_2} = \{o_d\}$ $\quad$ $\mathsf{freshPasta}^{\mathcal{I}_1} = \emptyset$
$\mathsf{Dish}^{\mathcal{I}_2} = \{o_d\}$ $\quad$ $\mathsf{Pasta}^{\mathcal{I}_2} = \{o_p\}$ $\quad$ $\mathsf{driedPasta}^{\mathcal{I}_2} = \{o_p\}$

Observe that $\mathcal{I}_1$ is not (homomorphically) contained in $\mathcal{I}_2$, and $\mathcal{I}_2$ is not (homomorphically) contained in $\mathcal{I}_1$, which shows that there is no universal model of this KB. ▲

3 Ontology-Mediated Query Answering

In this section, we will formally define the problem of ontology-mediated query answering, discuss how the complexity of this task can be measured, and introduce two key algorithmic techniques for OMQA. This will lay the necessary foundations for later sections, in which we will present concrete algorithms and complexity results for different query languages and DLs.

As ontology-mediated query answering is closely related to the more well-studied problem of querying relational databases, the first part of this section will recall some key notions from databases and discuss the important differences between the two settings.

3.1 Databases and ABoxes

Recall from the preceding section that in description logics, data is stored in the ABox as a set of assertions of the forms $A(a)$ and $r(a, b)$, where A is a concept name, r a role name, and a, b are individuals. From the first-order logic point of view, ABox assertions are simply *facts* built from unary and binary relation symbols and constants. ABoxes provide an *incomplete description* of the considered application domain, in the sense that everything stated in the ABox is assumed to be true, but facts which are not present in the ABox are not assumed to be false. This is known as the *open-world assumption* and is a desirable property in our setting as it allows us to leave the truth of some facts unspecified and to be able to infer new pieces of information from the explicit information asserted in ABox and TBox.

Relational databases constitute one of the most common ways of storing data in modern information systems. *Relational database instances* (which we will often abbreviate to databases) can be defined similarly to ABoxes as finite sets of

facts $P(a_1, \ldots, a_n)$, where P is a relation symbol of arity $n \geq 0$. In addition to allowing facts of arbitrary arity, databases differ from ABoxes in another important respect: they are interpreted under the *closed-world assumption*, meaning that all facts that are contained in the database are assumed true and those that are absent are *assumed to be false*. Concretely, this means that every database instance $\mathcal{D}$ corresponds to the *unique first-order interpretation* $\mathcal{I}_\mathcal{D}$ whose domain $\Delta^{\mathcal{I}_\mathcal{D}}$ contains all constants appearing in $\mathcal{D}$ and which interprets every relation symbol P as $\{\boldsymbol{a} \mid P(\boldsymbol{a}) \in D\}$ and every constant as itself.

Remark 3. Databases make the *standard names assumption*, which consists in interpreting constants as themselves. Note that this is strictly stronger than the unique names assumption discussed in Section 2.1.

The next example illustrates the difference between ABoxes and databases.

Example 6. Let $\mathcal{D}$ and $\mathcal{A}_1$ be respectively the database and ABox corrresponding to the following set of facts (assertions):

$$\mathsf{Cake}(d_1) \quad \mathsf{IceCream}(d_2) \quad \mathsf{Dessert}(d_3) \quad \mathsf{hasDessert}(m, d_4)$$

The database $\mathcal{D}$ corresponds to the interpretation $\mathcal{I}_\mathcal{D}$ defined as follows:

- $\Delta^{\mathcal{I}_\mathcal{D}} = \{m, d_1, d_2, d_3, d_4\}$
- $\mathsf{Cake}^{\mathcal{I}_\mathcal{D}} = \{d_1\}$
- $\mathsf{IceCream}^{\mathcal{I}_\mathcal{D}} = \{d_2\}$
- $\mathsf{Dessert}^{\mathcal{I}_\mathcal{D}} = \{d_3\}$
- $\mathsf{hasDessert}^{\mathcal{I}_\mathcal{D}} = \{(m, d_4)\}$
- $c^{\mathcal{I}_\mathcal{D}} = c$ for every $c \in \{m, d_1, d_2, d_3, d_4\}$

According to the interpretation $\mathcal{I}_D$, there are five entities (m, d_1, d_2, d_3, d_4), and only d_3 belongs to $\mathsf{Dessert}$.

The interpretation $\mathcal{I}_\mathcal{D}$ is a model of the ABox $\mathcal{A}_1$, but $\mathcal{A}_1$ has (infinitely) many other models, including the interpretation $\mathcal{J}$ defined as follows[6]:

- $\Delta^{\mathcal{I}_\mathcal{D}} = \{m, m', d_1, d_2, d_3, d_4, d_7, e, f_4, g\}$
- $\mathsf{Cake}^{\mathcal{I}_\mathcal{D}} = \{d_1, d_3, e\}$
- $\mathsf{IceCream}^{\mathcal{I}_\mathcal{D}} = \{d_2, f_4\}$
- $\mathsf{Dessert}^{\mathcal{I}_\mathcal{D}} = \{d_1, d_2, d_3, d_4, e, f_4\}$
- $\mathsf{hasDessert}^{\mathcal{I}_\mathcal{D}} = \{(m, d_4), (m', d_3)\}$
- $c^{\mathcal{I}_\mathcal{D}} = c$ for every $c \in \{m, d_1, d_2, d_3, d_4\}$

The interpretation $\mathcal{J}$ contains additional domain elements that are not explicitly mentioned in $\mathcal{A}_1$, and it also makes true some assertions that are not present in $\mathcal{A}_1$. For example, there are now six entities (including the ABox individuals d_1, d_2, d_3, and d_4) that belong to the class $\mathsf{Dessert}$. Since some models of $\mathcal{A}_1$ state that d_1 is a dessert, and others do not, the truth of this assertion is left undefined by $\mathcal{A}_1$. However, if we add a TBox containing the information that all cakes are desserts ($\mathsf{Cake} \sqsubseteq \mathsf{Dessert}$), then this will eliminate some of the models and allow us to infer that d_1 is a dessert. ▲

[6] Note that for simplicity, and to facilitate the comparison with $\mathcal{I}_D$, the interpretation $\mathcal{J}$ interprets individuals as themselves. However, we could instead have chosen domain elements distinct from the individual names.

3.2 Querying Databases

Queries provide the means of accessing the data stored in database instances. There are several different query languages that have been proposed for relational databases, each providing a formal syntax for constructing queries and a semantics that defines the result of evaluating a query on a given database. Formally, the semantics of a query q specifies for every database $\mathcal{D}$ the set $\mathsf{ans}(q, \mathcal{D})$ of *answers of q over $\mathcal{D}$*, where the answers take the form of tuples of constants from $\mathcal{D}$. Since databases correspond to interpretations, we can alternatively view database queries as *mappings from interpretations to tuples of domain elements*. Note that although databases correspond to finite interpretations, it is typically straightforward to extend the semantics of queries to arbitrary interpretations, and so we may assume that ans is defined for all interpretations, finite or infinite. Thus, for the purposes of this chapter, a query q of arity $n \geq 0$ associates with every interpretation $\mathcal{I}$ a subset $\mathsf{ans}(q, \mathcal{I}) \subseteq (\Delta^{\mathcal{I}})^n$. Note that when $n = 0$, the query q is called *Boolean*, and $\mathsf{ans}(q, \mathcal{I})$ can take one of two values: the empty tuple $()$ (meaning q holds in $\mathcal{I}$) or $\emptyset$ (q does not hold).

We introduce next two specific database query languages that will play an important role in this chapter, namely, first-order queries and Datalog queries.

First-order Queries A common way of specifying queries is to use formulas from some logic, with first-order logic being the standard choice. A *first-order (FO) query* is a first-order formula built from relational atoms $P(t_1, \ldots, t_n)$ and equality atoms $t_1 = t_2$ (with each t_i a constant or variable) using the Boolean connectives $(\vee, \wedge, \neg, \rightarrow)$ and universal and existential quantifiers ($\forall x$, $\exists x$). The free variables of FO queries will be called *answer variables*, and the *arity* of an FO query is defined as its number of answer variables. Given an interpretation $\mathcal{I}$, an FO query φ with answer variables $(x_1, \ldots, x_n)$, and a tuple $(e_1, \ldots, e_n)$ of elements from $\Delta^{\mathcal{I}}$, we use $\mathcal{I} \models \varphi[(x_1, \ldots, x_n) \mapsto (e_1, \ldots, e_n)]$ to denote that the FO formula φ is satisfied in $\mathcal{I}$ under the variable assignment that maps each x_i to e_i. The semantics of an FO query φ with answer variables $\boldsymbol{x} = (x_1, \ldots, x_n)$ is defined using this notion of satisfaction:

$$\mathsf{ans}(q, \mathcal{I}) = \{\boldsymbol{e} = (e_1, \ldots, e_n) \in (\Delta^{\mathcal{I}})^n \mid \mathcal{I} \models \varphi[\boldsymbol{x} \mapsto \boldsymbol{e}]\}$$

An FO query φ is called *domain independent* if $\mathsf{ans}(\varphi, \mathcal{I}) = \mathsf{ans}(\varphi, \mathcal{J})$ for every pair of interpretations $\mathcal{I}, \mathcal{J}$ such that $\cdot^{\mathcal{I}} = \cdot^{\mathcal{J}}$, i.e., $\mathcal{I}$ and $\mathcal{J}$ interpret all predicate and constant symbols identically.

We remark that domain-independent first-order queries provide the logical underpinnings of SQL, which is the most widely used query language in commercial database systems. Every domain independent first-order quey can be translated into an equivalent SQL query, which means that such queries can be evaluated using standard relational database management systems.

Remark 4. Unless stated otherwise, all first-order queries considered in this chapter are domain independent.

Datalog Queries Datalog is a rule-based formalism that originated from work on logic programming and has been extensively studied within the database community as a powerful language for expressing recursive queries (see e.g. [66] and Chapters 12-13 of [1]). A *Datalog rule* takes the form

$$P_n(\boldsymbol{t}_n) \leftarrow P_1(\boldsymbol{t}_1), \ldots, P_{n-1}(\boldsymbol{t}_{n-1})$$

where each $P_i(\boldsymbol{t}_i)$ is a relational atom. We call $P_n(\boldsymbol{t}_n)$ the *head* of the rule and $P_1(\boldsymbol{t}_1), \ldots, P_{n-1}(\boldsymbol{t}_{n-1})$ the rule *body*. Datalog rules are required to satisfy the following *safety condition*: every variable that appears in the rule head must also occur in one of the atoms of the rule body. A *Datalog program* consists of a finite set of Datalog rules, and a *Datalog query* is a pair (Π, Q) where Π is a Datalog program and Q is a relation symbol that appears in Π.

Every Datalog rule can be viewed as a first-order sentence[7]: simply reverse the direction of the implication symbol, take the conjunction of the body atoms, and quantify universally over all variables. For example, consider the following Datalog rule and its corresponding FO sentence:

$$Q(v,u) \leftarrow P(v,x), T(x,u,w) \qquad \rightsquigarrow \qquad \forall xuvw\ (P(v,x) \wedge T(x,u,w)) \rightarrow Q(v,u)$$

We will say that a first-order interpretation $\mathcal{I}$ is a *model of a Datalog rule* if $\mathcal{I}$ is a model of the corresponding FO sentence, and we call $\mathcal{I}$ a *model of a Datalog program* Π just in the case that $\mathcal{I}$ is a model of every rule in Π.

Recall that we view sets of facts as interpretations, hence the semantics of a Datalog program is defined relative to a given interpretation that corresponds to an (extensional) database. Given an interpretation $\mathcal{J}$ and a Datalog program Π, we will call an interpretation $\mathcal{I}$ a *minimal model of* Π *relative to* $\mathcal{J}$ just in the case that the following conditions hold:

1. $\Delta^{\mathcal{I}} = \Delta^{\mathcal{J}}$
2. $c^{\mathcal{I}} = c^{\mathcal{J}}$ for every constant c
3. $\mathcal{I}$ is a model of Π
4. for every other interpretation $\mathcal{I}'$ that satisfies the three preceding conditions, we have that $P^{\mathcal{I}} \subseteq P^{\mathcal{I}'}$ for every relation symbol P.

In fact, it can be shown that for every interpretation $\mathcal{J}$ and Datalog program Π there is a *unique minimal model* of Π relative to $\mathcal{I}$. Using the minimal model, we may define the semantics of Datalog queries as follows:

$$\mathsf{ans}((\Pi, Q), \mathcal{I}) = Q^{\mathcal{J}} \qquad \text{where } \mathcal{I} \text{ is the minimal model of } \Pi \text{ relative to } \mathcal{J}$$

Remark 5. Datalog queries can also be given a *procedural semantics*, in which the minimal model is computed by an exhaustive application of the Datalog rules starting from the initial set of facts in the database (or the initial relations in the interpretation).

[7] More precisely, Datalog rules correspond to function-free *Horn clauses*.

3.3 Querying Description Logic Knowledge Bases

In principle, any query language defined for databases can be used to query description logic knowledge bases. From the syntactic point of view, the only difference is that in place of arbitrary relation symbols, we will use concept and role names. It is less obvious how to lift the semantics of queries to DL knowledge bases, as DL KBs typically have multiple models, whereas the semantics of queries only states how to obtain answers from a single interpretation. The solution is to adopt so-called certain answer semantics[8], in which we consider those answers that hold with respect to each of the KB's models. The intuition is that since we do not know which of the KB's models provides the correct description of the application domain, we can only be confident in those answers that can be obtained from *every* model of the KB.

Definition 3 (Certain answers). *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a DL KB, and let q be an n-ary query. The set* $\mathsf{cert}(q, \mathcal{K})$ *of* certain answers *to q over $\mathcal{K}$ is defined as follows:*

$$\{(a_1, \ldots, a_n) \in \mathsf{Ind}(\mathcal{A})^n \mid (a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}) \in \mathsf{ans}(q, \mathcal{I}) \text{ for every } \mathcal{I} \in \mathsf{Mods}(\mathcal{K})\}$$

Remark 6. Observe that certain answers are defined as *tuples of individuals*, rather than tuples of domain elements. This distinction is important since we do not make the standard names assumption, and so an individual may be mapped to different elements in different models of the KB.

Remark 7. If we consider Boolean FO queries, then certain answer semantics corresponds to *logical entailment*. Indeed, the DL KB can be expressed as a FO theory, and the problem is to determine whether the query (an FO sentence) holds in every model of the KB. By contrast, the evaluation of Boolean FO queries over databases corresponds to *model checking*, since we need to check whether the query (FO sentence) holds w.r.t. to a given FO interpretation. It is well known that logical entailment is more difficult than model checking, and so it is no surprise that ontology-mediated query answering is a more challenging computational task than query answering in databases.

We illustrate the notion of certain answers on an example.

Example 7. We consider the KB $\mathcal{K}_1$ consisting of the 'dessert' ABox $\mathcal{A}_1$ from Example 6 and the following TBox $\mathcal{T}_1$:

$$\mathsf{Cake} \sqsubseteq \mathsf{Dessert} \quad \mathsf{IceCream} \sqsubseteq \mathsf{Dessert} \quad \mathsf{hasDessert} \sqsubseteq \mathsf{hasCourse}$$
$$\exists \mathsf{hasCourse} \sqsubseteq \mathsf{Menu} \quad \exists \mathsf{hasDessert}^- \sqsubseteq \mathsf{Dessert}$$

Suppose that we are interested in finding all desserts, i.e., we wish to find all certain answers to the query $q_1 = \mathsf{Dessert}(x)$. Since there are five individuals in the ABox, there are five potential certain answers. We argue that four of them are indeed certain answers:

[8] The certain answer semantics is also used in other contexts, such as incomplete databases [112], data integration [142], and data exchange [6].

– $d_1 \in \mathsf{cert}(q, \mathcal{K}_1)$, since $\mathsf{Cake}(d_1) \in \mathcal{A}_1$ and $\mathsf{Cake} \sqsubseteq \mathsf{Dessert} \in \mathcal{T}_1$, and so we must have $d_1^{\mathcal{I}} \in \mathsf{Dessert}^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{K}_1$
– $d_2 \in \mathsf{cert}(q_1, \mathcal{K}_1)$, since $\mathsf{IceCream}(d_2) \in \mathcal{A}_1$ and $\mathsf{IceCream} \sqsubseteq \mathsf{Dessert} \in \mathcal{T}_1$
– $d_3 \in \mathsf{cert}(q_1, \mathcal{K}_1)$, since $\mathsf{Dessert}(d_3)$ appears explicitly in $\mathcal{A}_1$
– $d_4 \in \mathsf{cert}(q_1, \mathcal{K}_1)$, since $\mathsf{hasDessert}(m, d_4) \in \mathcal{A}_1$ and $\mathsf{hasDessert}^- \sqsubseteq \mathsf{Dessert} \in \mathcal{T}_1$

The fifth individual, m, is not a certain answer to q_1 w.r.t. $\mathcal{K}_1$. To see why, let us extend the interpretation $\mathcal{J}$ from Example 6 by setting:

– $\mathsf{hasCourse}^{\mathcal{J}} = \{(m, d_4), (m, g), (m', d_3), (m', g)\}$
– $\mathsf{Menu}^{\mathcal{J}} = \{m, m'\}$

It can be verified that $\mathcal{J}$ is a model of $\mathcal{K}_1$, yet $m^{\mathcal{J}} = m$ does not belong to $\mathsf{Dessert}^{\mathcal{J}}$. ▲

In this chapter, our main focus will be on the problem of ontology-mediated query answering, which will consist in computing the certain answers of queries over a DL KB. We will be particularly interested in understanding how the complexity of this task varies depending on the query language and description logic considered. For the purposes of analyzing the complexity of ontology-mediated query answering, we will recast OMQA as a decision problem[9]:

PROBLEM:	$\mathcal{Q}$ answering in $\mathcal{L}$ (with $\mathcal{Q}$ a query language and $\mathcal{L}$ a DL)
INPUT:	An n-ary query $q \in \mathcal{Q}$, an ABox $\mathcal{A}$, a TBox $\mathcal{T}$ formulated in $\mathcal{L}$, and a tuple $\boldsymbol{a} \in \mathsf{Ind}(\mathcal{A})^n$
QUESTION:	Does $\boldsymbol{a}$ belong to $\mathsf{cert}(q, (\mathcal{T}, \mathcal{A}))$?

To solve the problem of $\mathcal{Q}$ answering in $\mathcal{L}$, we must devise a *decision procedure*, that is, an algorithm that satisfies the following three requirements:

– Termination: the procedure is guaranteed to halt on any input
– Soundness: if the procedure returns 'yes', then $\boldsymbol{a} \in \mathsf{cert}(q, \mathcal{K})$
– Completeness: if $\boldsymbol{a} \in \mathsf{cert}(q, \mathcal{K})$, then the procedure returns 'yes'

We remark that if we have such a decision procedure, then we can use it to solve the original task of computing all certain answers. Indeed, we can enumerate all tuples of the same arity as the query, and for each, we can use the decision procedure to check whether or not the tuple is a certain answer.

According to certain answer semantics, if we pose an n-ary query q to an unsatisfiable KB $\mathcal{K}$, then every n-tuples of individuals from $\mathcal{K}$ is a certain answer, and so we will answer 'yes' for every tuple. Thus, query answering is trivial when the KB is unsatisfiable. It follows that to obtain a decision procedure for $\mathcal{Q}$ answering in $\mathcal{L}$, it suffices to provide decision procedures for the following two problems: (i) satisfiability of $\mathcal{L}$ KBs, and (ii) $\mathcal{Q}$ answering over *satisfiable* $\mathcal{L}$ KBs.

[9] We recall that a *decision problem* (alternatively known as a recognition problem) is a problem with a yes-or-no answer.

3.4 Complexity of Query Answering

We have seen in the previous subsection how the problem of ontology-mediated query answering can be formulated as a decision problem. Database query evaluation can be similarly recast as a decision problem: we are given as input a query q, database $\mathcal{D}$, and tuple of constants $\boldsymbol{a}$, and the problem is to decide whether $\boldsymbol{a} \in \mathsf{cert}(q, \mathcal{I}_\mathcal{D})$. When we speak of the complexity of OMQA or database query evaluation, we will always mean the complexity of these decision problems. In what follows, we assume that we have a function $|\cdot|$ that assigns to each of the objects (queries, TBoxes, ABoxes, databases, tuples) appearing in the input a natural number corresponding to the *size* of the object, e.g. the length of its string representation according to some suitable encoding. For example, $|\mathcal{T}|$ will denote the size of TBox $\mathcal{T}$.

The complexity of decision problems can be measured in different ways, depending on which inputs we choose to count and which we treat as fixed. When analyzing the complexity of query answering, there are two commonly considered complexity measures [209]:

- *Combined complexity* is measured as a function of the *size of the whole input*, that is, $|q| + |\mathcal{T}| + |\mathcal{A}| + |\boldsymbol{a}|$ in the case of OMQA and $|q| + |\mathcal{D}| + |\boldsymbol{a}|$ in the case of database query evaluation[10].
- *Data complexity* is with respect to the *size of the data*, that is, $|\mathcal{A}|$ for OMQA and $|\mathcal{D}|$ in the database setting. The sizes of all other inputs are treated as fixed constants and so they do not contribute to the complexity.

Combined complexity corresponds to the 'classical' way of measuring complexity, in which we consider all of the inputs to the decision problem and treat them equally. If we show that a problem is in polynomial time for combined complexity, then this is a good indication that the problem can be efficiently solved in practice. However, a problem that is intractable in combined complexity may nonetheless be prove feasible on typical inputs. Indeed, database query evaluation has been proven intractable in combined complexity for all of the commonly considered query languages, yet modern database systems are able to answer most user queries instantaneously. The key observation is that the queries encountered in practice are typically quite small, and their size is negligible when compared to the size of the (typically very large) database, and so the real predictor of performance is how querying algorithms scale with respect to the size of the database. For this reason, data complexity is generally considered the more useful complexity measure for databases. In the setting of ontology-mediated query answering, in addition the query and ABox, we have a TBox whose size can vary widely, from a few dozen axioms up to tens (or even hundreds) of thousands. However, it seems reasonable to assume that in most OMQA applications the ABox will be significantly larger than the TBox, so data complexity is also very relevant in this setting.

Computational complexity [172, 9] provides a hierarchy of different complexity classes that can be used to classify problems according to the amount of

[10] We typically omit $|\boldsymbol{a}|$ since we have $|\boldsymbol{a}| \leq |q| \cdot |\mathcal{A}|$ (or $|\boldsymbol{a}| \leq |q| \cdot |\mathcal{D}|$).

resources (time, space) that are required in order to solve them. In this paper, we will make use of the following complexity classes, which are ordered according to inclusion with each class being included in those later in the list:

- AC_0: problems that can be solved by a uniform family of circuits of constant depth and polynomial size, with unlimited fan-in AND gates and OR gates
- NLogSpace: problems that can be solved in non-deterministic logarithmic space
- P: problems that can be solved in polynomial time
- NP (resp. coNP): problems that can be solved (resp. whose complement can be solved) in non-deterministic polynomial time
- PSpace: problems that can be solved in polynomial space
- Exp: problems that can be solved in single-exponential time

It is known that AC_0 is strictly contained in the class LogSpace of all problems solvable in deterministic logarithmic space, which in particular means that it is a proper subclass of P.

The following theorems summarize what is known about the complexity of evaluating first-order and Datalog queries over databases:

Theorem 1 ([209, 210]). *First-order query evaluation is in* AC_0 *in data complexity and* PSpace*-complete in combined complexity.*

Theorem 2 ([209, 113]). *Datalog query evaluation is* P*-complete in data complexity and* Exp*-complete in combined complexity.*

In later sections, we will investigate the complexity of answering different forms of queries over knowledge bases formulated using the Horn DLs introduced in Section 2. As we shall see, in contrast to expressive DLs, for which answering even the simplest queries is coNP-hard in data complexity, it is possible to design query answering algorithms for Horn DLs that scale polynomially in the size of the ABox. It is for this reason that we sometimes use the term 'data-tractable' when referring to Horn DLs.

3.5 Techniques for Ontology-Mediated Query Answering

We have seen that in general, ontology-mediated query answering is more complex than database query evaluation, since we must consider all models of a KB rather than the single interpretation associated with a database. Query rewriting and saturation are two techniques that can be used to bridge this gap and enable the use of existing database systems for OMQA. These two techniques underlie most of the OMQA algorithms that have been developed for Horn DLs, including those that will be presented in this chapter.

Query Rewriting The basic idea behind query rewriting is as follows. In a first step, we rewrite the input query into a new query that contains all the relevant information from the TBox. In a second step, we pass the rewritten query to

a database system for evaluation over the ABox, which is treated as a (closed-world) database instance. Query rewriting thus provides a means of reducing the OMQA problem to the more well-studied problem of database query evaluation.

We now formalize the notion of a rewriting of a query. Note that we will use $\mathcal{I}_\mathcal{A}$ to denote the finite interpretation obtained by viewing $\mathcal{A}$ as a database. More precisely: $\Delta^{\mathcal{I}_\mathcal{A}} = \mathsf{Ind}(\mathcal{A})$, $A^{\mathcal{I}_\mathcal{A}} = \{a \mid A(a) \in \mathcal{A}\}$ (for every $A \in \mathsf{N_C}$), $r^{\mathcal{I}_\mathcal{A}} = \{(a,b) \mid r(a,b) \in \mathcal{A}\}$ (for every $r \in \mathsf{N_R}$), and $a^{\mathcal{I}_\mathcal{A}} = a$ (for every $a \in \mathsf{Ind}(\mathcal{A})$).

Definition 4 (Rewriting of a Query). *Let $\mathcal{T}$ be a DL TBox, let Σ be a finite signature, and let q, q' be two queries. We say that q' is a* rewriting of q w.r.t. $\mathcal{T}, \Sigma$ *just in the case that* $\mathsf{cert}(q, (\mathcal{T}, \mathcal{A})) = \mathsf{ans}(q', \mathcal{I}_\mathcal{A})$ *for every Σ-ABox $\mathcal{A}$. We call q' a* rewriting of q w.r.t. $\mathcal{T}, \Sigma$ relative to consistent ABoxes *if* $\mathsf{cert}(q, (\mathcal{T}, \mathcal{A})) = \mathsf{ans}(q', \mathcal{I}_\mathcal{A})$ *for every Σ-ABox $\mathcal{A}$ such that $(\mathcal{T}, \mathcal{A})$ is satisfiable.*

Remark 8. The signature Σ specifies the concept and role names that can appear in ABoxes of the considered application. To simplify the presentation, we will sometimes omit mention of Σ, when it is unimportant to the discussion at hand.

Example 8. Reconsider the query $q_1 = \mathsf{Dessert}(x)$ and KB $\mathcal{K}_1 = (\mathcal{T}_1, \mathcal{A}_1)$ from Example 7. It can be verified that the query

$$q_1' = \mathsf{Dessert}(x) \vee \mathsf{Cake}(x) \vee \mathsf{IceCream}(x) \vee \exists y.\mathsf{hasDessert}(y, x)$$

is a rewriting of q_1 w.r.t. $\mathcal{T}_1$. Intuitively, this is because q_1' captures the four ways to infer, using the axioms in $\mathcal{T}_1$, that a given ABox individual is an instance of the concept $\mathsf{Dessert}$.

If we evaluate q_1' over $\mathcal{I}_{\mathcal{A}_1}$ (which is identical to $\mathcal{I}_\mathcal{D}$ from Example 6), we obtain $\mathsf{ans}(q_1', \mathcal{I}_{\mathcal{A}_1}) = \{d_1, d_2, d_3, d_4\}$. Indeed:

- d_1 is an answer to the disjunct $\mathsf{Cake}(x)$, due to the assertion $\mathsf{Cake}(d_1)$
- d_2 is an answer to the disjunct $\mathsf{IceCream}(x)$, due to the assertion $\mathsf{IceCream}(d_2)$
- d_3 is an answer to the disjunct $\mathsf{Dessert}(x)$, due to the assertion $\mathsf{Dessert}(d_3)$
- d_4 is an answer to the disjunct $\exists y.\mathsf{hasDessert}(y, x)$, due to the assertion $\mathsf{hasDessert}(m, d_4)$

We can therefore conclude that $\mathsf{cert}(q_1, \mathcal{K}_1) = \{d_1, d_2, d_3, d_4\}$. ▲

We can define an analogous notion of rewriting for testing KB satisfiability.

Definition 5 (Rewriting of Unsatisfiability). *Let $\mathcal{T}$ be a DL TBox, let Σ be a finite signature, and let $q_\bot$ be a Boolean query. We call $q_\bot$ a* rewriting of unsatisfiability w.r.t. $\mathcal{T}, \Sigma$ *if for every Σ-ABox $\mathcal{A}$, we have* $\mathsf{cert}(q, (\mathcal{T}, \mathcal{A})) = ()$ *iff $(\mathcal{T}, \mathcal{A})$ is unsatisfiable.*

The preceding notions of rewriting can be further specialized by adding to the above definitions the requirement that the query q' (resp. $q_\bot$) be an FO or Datalog query, yielding the notions of *FO rewritings* and *Datalog rewritings*.

In the same way as query answering can be decomposed into satisfiability checking and query answering w.r.t. satisfiable KBs, one can show that it is

sufficient to be able to construct rewritings of unsatisfiability and rewritings of queries relative to consistent ABoxes. Indeed, if we have an FO rewriting $q_\bot$ of unsatisfiability w.r.t. $\mathcal{T}, \Sigma$ and an FO rewriting q' of q w.r.t. $\mathcal{T}, \Sigma$ relative to consistent ABoxes, then these can be combined to obtain an FO rewriting of q w.r.t. $\mathcal{T}, \Sigma$ (over arbitrary Σ-ABoxes). Basically, if q has answer variables $x_1, \ldots, x_n$, then the desired rewriting takes the form $q' \vee (q_\bot \wedge q^\Sigma_{\mathsf{ind}}(x_1) \wedge \ldots \wedge q^\Sigma_{\mathsf{ind}}(x_n))$, where q^Σ_{ind} is a unary query that retrieves all of the individuals that occur in a given Σ-ABox. Using a similar construction, we can show that it is possible to construct a Datalog rewriting of a query by combining a Datalog rewriting of the query relative to consistent ABoxes with a Datalog rewriting of unsatisfiability.

It is important to keep in mind that the existence of a rewriting is not guaranteed: it is possible to find queries and TBoxes for which no FO (resp. Datalog) rewriting exists. Moreover, as the next example illustrates, the existence of a rewriting depends upon the type of rewriting we consider.

Example 9. The query $\mathsf{Spicy}(x)$ has no FO rewriting w.r.t. the TBox

$$\{\exists \mathsf{hasIngredient}.\mathsf{Spicy} \sqsubseteq \mathsf{Spicy}\}.$$

Intuitively, we would like to use the following (infinite) query, which looks for $\mathsf{hasIngredient}$ chains that start at x and end at an individual asserted to be Spicy:

$$\begin{aligned}\mathsf{Spicy}(x) \vee\ & \exists x'(\mathsf{hasIngredient}(x, x') \wedge \mathsf{Spicy}(x')) \\ \vee\ & \exists x' x''(\mathsf{hasIngredient}(x, x') \wedge \mathsf{hasIngredient}(x', x'') \wedge \mathsf{Spicy}(x'')) \vee \ldots\end{aligned}$$

It can be proven, using techniques from finite model theory (see the introductory texts [76, 145]), that no FO query that is equivalent to this infinite disjunction.

However, this same query and TBox possesses a Datalog rewriting. Indeed, it suffices to take the single-rule Datalog program

$$\Pi = \{\mathsf{Spicy}(x) \leftarrow \mathsf{hasIngredient}(x, y), \mathsf{Spicy}(y)\}$$

and use Spicy as the distinguished relation. In this particular case, the Datalog program is just a translation of the TBox, but this is not the case in general. ▲

Most of the work to date has focused on FO rewritings, since (domain-independent) FO queries can be translated into SQL statements and evaluated using highly optimized relational database management systems. However, Datalog rewritings, which can be passed to Datalog engines for evaluation, are also popular as they are applicable to a wider range of DLs.

Saturation We have seen that standard database querying algorithms are incomplete for OMQA since they do not take into account the information provided by the TBox. Query rewriting addresses this problem by rewriting the query so as to incorporate the relevant information from the TBox. By contrast, saturation-based approaches to OMQA work by *rendering explicit (some*

of) the implicit information contained in the KB, making it available for query evaluation. In simple cases, saturation involves completing the ABox by adding those assertions that are logically entailed from the KB, and then evaluating the query over the saturated ABox. In more complex cases, we might have to have to enrich the ABox in other ways (perhaps adding new ABox individuals to act as witnesses for the existential restrictions), or we may need to combine saturation with query rewriting. Indeed, unlike query rewriting, for which we could formulate precise definitions of what constitutes a rewriting, saturation is a more abstract concept that englobes a variety of different approaches whose commonality is that they enrich the KB with some additional information, which can then be exploited for various reasoning tasks (in our case, query answering). Saturation-based reasoning techniques have been employed in a variety of areas, sometimes under different names: forward chaining, materialization, deductive closure, and consequence-based reasoning.

Example 10. We return to our running example about desserts. By 'applying' the inclusions in the TBox $\mathcal{T}_1$ to the ABox $\mathcal{A}_1$, we obtain a new saturated KB $\mathcal{K}_1'$ with the following additional assertions:

- $\mathsf{Dessert}(d_1)$, using $\mathsf{Cake}(d_1)$ and $\mathsf{Cake} \sqsubseteq \mathsf{Dessert}$
- $\mathsf{Dessert}(d_2)$, using $\mathsf{IceCream}(d_2)$ and $\mathsf{IceCream} \sqsubseteq \mathsf{Dessert}$
- $\mathsf{hasCourse}(m, d_4)$, using $\mathsf{hasDessert}(m, d_4)$ and $\mathsf{hasDessert} \sqsubseteq \mathsf{hasCourse}$
- $\mathsf{Menu}(m)$, using $\mathsf{hasCourse}(m, d_4)$ and $\exists \mathsf{hasCourse} \sqsubseteq \mathsf{Menu}$
- $\mathsf{Dessert}(d_4)$, using $\mathsf{hasDessert}(m, d_4)$ and $\exists \mathsf{hasDessert}^- \sqsubseteq \mathsf{Dessert}$

Once we have computed $\mathcal{K}_1'$, answering our query $\mathsf{Dessert}(x)$ is as simple as reading off the individuals that appear in a $\mathsf{Dessert}$ assertion: d_1, d_2, d_3, d_4. We observe that this is the same result as was obtained in Example 8 by means of query rewriting. Note however that the saturation process is performed independently of the query, so we infer not only those assertions needed to answer the specific query at hand, but also those needed to answer future queries. For example, using the same saturated KB $\mathcal{K}_1'$, we find that m is the unique certain answer to the query $\mathsf{Menu}(x)$. ▲

4 Instance Queries

In this section, we begin our exploration of ontology-mediated query answering by considering a very simple type of query that can be used to find all individuals that belong to a given concept or role. Up until the mid-2000s, work on querying DL knowledge bases focused almost exclusively on such queries, which are commonly known as instance queries.

Definition 6 (Instance queries). *An* instance query (IQ) *takes one of the following two forms:*

- $A(x)$ *where* $A \in \mathsf{N_C}$ *(concept instance query)*
- $r(x, y)$ *where* $r \in \mathsf{N_R}$ *(role instance query)*

ALGORITHM ComputeSubsumees
INPUT: DL-Lite$_R$ TBox $\mathcal{T}$, concept $B \in \mathsf{N_C} \cup \{\exists R \mid R \in \mathsf{N_R^{\pm}}\}$

1. Initialize $\mathsf{Subsumees} = \{B\}$ and $\mathsf{Examined} = \emptyset$.
2. While $\mathsf{Subsumees} \setminus \mathsf{Examined} \neq \emptyset$
   (a) Select $D \in \mathsf{Subsumees} \setminus \mathsf{Examined}$ and add D to $\mathsf{Examined}$.
   (b) For every concept inclusion $C \sqsubseteq D \in \mathcal{T}$
      – If $C \notin \mathsf{Subsumees}$, add C to $\mathsf{Subsumees}$
   (c) For every role inclusion $R \sqsubseteq S \in \mathcal{T}$ such that $D = \exists S$.
      – If $\exists R \notin \mathsf{Subsumees}$, add $\exists R$ to $\mathsf{Subsumees}$
   (d) For every role inclusion $R \sqsubseteq S \in \mathcal{T}$ such that $D = \exists \mathsf{inv}(S)$.
      – If $\exists \mathsf{inv}(R) \notin \mathsf{Subsumees}$, add $\exists \mathsf{inv}(R)$ to $\mathsf{Subsumees}$.
3. Return $\mathsf{Subsumees}$.

Fig. 1: Algorithm for computing subsumees of a given concept in DL-Lite$_R$.

Remark 9. The restriction to concept names in the preceding definition is without loss of generality. Indeed, suppose that we want to find all individuals that belong to C, where C is an arbitrary concept formulated in the DL we are considering. This can be accomplished by taking a fresh concept name A_C, adding the inclusion $C \sqsubseteq A_C$ to the TBox, and using the instance query $A_C(x)$.

In the remainder of this section, we will see how the techniques of query rewriting and saturation introduced in Section 3.5 can be applied to the problem of IQ answering (which is more commonly referred to as *instance checking*). We will consider three representative Horn DLs: DL-Lite$_R$, $\mathcal{EL}$, and $\mathcal{ELHI}_\bot$.

4.1 Instance Checking in DL-Lite$_R$ via Query Rewriting

We begin by considering the problem of instance checking over DL-Lite$_R$ knowledge bases. Both query rewriting and saturation-based approaches have been proposed in the literature [52]. We present a procedure based upon rewriting IQs into first-order queries since this is the more commonly used approach for DLs in the DL-Lite family. Moreover, it provides us with a simple setting in which to demonstrate this technique.

As mentioned in Section 3.5, to construct an FO-rewriting of an instance query q w.r.t. $\mathcal{T}, \Sigma$, it suffices to construct

– an FO-rewriting of q w.r.t. $\mathcal{T}, \Sigma$ relative to consistent ABoxes, and
– an FO-rewriting of unsatisfiability w.r.t. $\mathcal{T}, \Sigma$.

Indeed, if we have these two rewritings, then they can be straightforwardly combined to obtain an FO-rewriting of q that works for all Σ-ABoxes.

As a first step, we present in Figure 1 a procedure ComputeSubsumees that takes as input a DL-Lite concept B (that is, either a concept name or an exisential concept $\exists R$ with $R \in \mathsf{N_R^{\pm}}$) and a DL-Lite$_R$ TBox $\mathcal{T}$ and outputs the set of

ALGORITHM ComputeSubroles
INPUT: DL-Lite$_R$ TBox $\mathcal{T}$, role $R \in \mathsf{N}_\mathsf{R}^\pm$

1. Initialize $\mathsf{Subroles} = \{R\}$ and $\mathsf{Examined} = \emptyset$.
2. While $\mathsf{Subroles} \setminus \mathsf{Examined} \neq \emptyset$
   (a) Select $S \in \mathsf{Subroles} \setminus \mathsf{Examined}$ and add S to $\mathsf{Examined}$.
   (b) For every role inclusion $U \sqsubseteq S$ or $\mathsf{inv}(U) \sqsubseteq \mathsf{inv}(S)$ in $\mathcal{T}$
      – If $U \notin \mathsf{Subsumees}$, add U to $\mathsf{Subsumees}$
3. Return $\mathsf{Subroles}$.

Fig. 2: Algorithm for computing subroles of a given role in DL-Lite$_R$.

all DL-Lite concepts C such that $\mathcal{T} \models C \sqsubseteq B$. Such concepts are called the *subsumees of* B *w.r.t.* $\mathcal{T}$, and intuitively they capture all of the different reasons for an individual to be counted as a member of B. The algorithm ComputeSubsumees uses a backward chaining mechanism to iteratively compute the subsumees of B. The set Subsumees is used to store the subsumees that have been identified so far, and Examined keeps track of which concepts in Subsumees have already been examined. When examining a concept D, we add to Subsumees all those concepts that are direct subsumees of D, i.e., those for which we can infer the subsumption relationship using a single inclusion from $\mathcal{T}$.

We illustrate the functioning of ComputeSubsumees on an example.

Example 11. Consider the DL-Lite$_R$ TBox $\mathcal{T}_2$ consisting of the following axioms:

$$\mathsf{ItalianDish} \sqsubseteq \mathsf{Dish} \quad \mathsf{VegDish} \sqsubseteq \mathsf{Dish} \quad \mathsf{Dish} \sqsubseteq \exists \mathsf{hasIngredient}$$
$$\exists \mathsf{hasCourse}^- \sqsubseteq \mathsf{Dish} \quad \mathsf{hasMain} \sqsubseteq \mathsf{hasCourse} \quad \mathsf{hasDessert} \sqsubseteq \mathsf{hasCourse}$$

We run ComputeSubsumees on the input $(\mathcal{T}_2, \mathsf{Dish})$ in order to compute all of the concepts that imply Dish. In Step 1, we initialize Subsumees to $\{\mathsf{Dish}\}$. In the first iteration of the while loop, we have no choice but to select Dish. In Step 2(b), we will add ItalianDish, VegDish, and $\exists \mathsf{hasCourse}^-$ to Subsumees due to the inclusions $\mathsf{ItalianDish} \sqsubseteq \mathsf{Dish}$, $\mathsf{VegDish} \sqsubseteq \mathsf{Dish}$, and $\exists \mathsf{hasCourse}^- \sqsubseteq \mathsf{Dish}$ respectively. Note that we cannot use the inclusion $\mathsf{Dish} \sqsubseteq \exists \mathsf{hasIngredient}$ to add $\exists \mathsf{hasIngredient}$, since Dish appears on the left-hand side of the inclusion. Steps 2(c) and 2(d) are inapplicable since Dish is not an existential restriction. We will therefore return to the start of the while loop and select a new unexamined concept from Subsumees. Nothing new will be added to Subsumees when examining ItalianDish and VegDish, since these concepts do not appear on the right-hand side of any inclusions in $\mathcal{T}_2$. However, when we examine $\exists \mathsf{hasCourse}^-$, we will add both $\mathsf{hasMain}^-$ and $\mathsf{hasDessert}^-$, due to the role inclusions $\mathsf{hasMain} \sqsubseteq \mathsf{hasCourse}$ and $\mathsf{hasDessert} \sqsubseteq \mathsf{hasCourse}$ (this occurs in Step 2(c)). It can be verified that no further concepts will be added, and so the output of ComputeSubsumees will be

$$\{\mathsf{Dish}, \mathsf{ItalianDish}, \mathsf{VegDish}, \exists \mathsf{hasCourse}^-, \exists \mathsf{hasMain}^-, \exists \mathsf{hasDessert}^-\}.$$

We remark that these concepts capture all of the different ways of inferring that an individual is a member of Dish using the knowledge expressed in $\mathcal{T}_2$. ▲

Observe that there are at most $3|\mathcal{T}|$ concepts that can be added to $\mathsf{Subsumees}$, since there are at most $|\mathcal{T}|$ concept names appearing in $\mathcal{T}$ and at most $2|\mathcal{T}|$ concepts of the forms $\exists r^{(-)}$ with r a role name occurring in $\mathcal{T}$. As concepts are never removed from $\mathsf{Subsumees}$, and each concept in $\mathsf{Subsumees}$ is examined at most once, it follows that $\mathsf{ComputeSubsumees}$ runs in polynomial time in $|\mathcal{T}|$. One can further show that on input $(A, \mathcal{T})$ the algorithm outputs exactly the set of subsumees of A w.r.t. $\mathcal{T}$.

In Figure 2, we introduce an analogous procedure $\mathsf{ComputeSubroles}$ for roles. One can show that the procedure runs in polynomial time in $|\mathcal{T}|$ and a role S belongs to $\mathsf{ComputeSubroles}(R, \mathcal{T})$ just in the case that $\mathcal{T} \models S \sqsubseteq R$.

Next, we introduce a function ρ_x that translates concepts into FO queries. The variable x in the subscript of ρ_x indicates that the query should use x as the answer variable. The definition of ρ_x is what one would expect:

- $\rho_x(A) = A(x)$ for $A \in \mathsf{N_C}$
- $\rho_x(\exists r) = \exists y.r(x, y)$ for $r \in \mathsf{N_R}$
- $\rho_x(\exists r^-) = \exists y.r(y, x)$ for $r \in \mathsf{N_R}$

We can similarly introduce a function ρ_{xy} that maps roles to FO queries, using x, y as the first and second distinguished variables:

- $\rho_{xy}(r) = r(x, y)$ for $r \in \mathsf{N_R}$
- $\rho_{xy}(r^-) = r(y, x)$ for $r \in \mathsf{N_R}$

We now have the necessary machinery to construct the desired FO-rewritings. To obtain a rewriting of $A(x)$ w.r.t. $\mathcal{T}$ relative to consistent ABoxes, we simply take the disjunction of the queries obtained by applying ρ_x to the concepts in $\mathsf{ComputeSubsumees}(A, \mathcal{T})$:

$$\mathsf{RewriteIQ}(A, \mathcal{T}) = \bigvee_{C \in \mathsf{ComputeSubsumees}(A, \mathcal{T})} \rho_x(C)$$

The construction is similar if we have a role instance query $r(x, y)$:

$$\mathsf{RewriteIQ}(r, \mathcal{T}) = \bigvee_{S \in \mathsf{ComputeSubsumees}(r, \mathcal{T})} \rho_{x,y}(S)$$

Observe that an individual a belongs to the answer of $\mathsf{RewriteIQ}(A, \mathcal{T})$ on $\mathcal{I}_\mathcal{A}$ just in the case that there is an assertion in $\mathcal{A}$ that asserts the membership of a in one of the subsumees of A w.r.t. $\mathcal{T}$. Under the assumption that the KB $(\mathcal{T}, \mathcal{A})$ is satisfiable, we can show that the latter statement holds iff a is a certain answer to $A(x)$ over $(\mathcal{T}, \mathcal{A})$. Similar considerations apply to role instance queries.

Theorem 3. *For every finite signature Σ, concept name A (resp. role name r) and DL-Lite$_R$ TBox $\mathcal{T}$, the query* $\mathsf{RewriteIQ}(A, \mathcal{T})$ *(resp.* $\mathsf{RewriteIQ}(r, \mathcal{T})$*) is an FO-rewriting of $A(x)$ (resp. $r(x, y)$) w.r.t. $\mathcal{T}, \Sigma$ relative to consistent ABoxes.*

Remark 10. We can sometimes use the ABox signature Σ to simplify rewritings. Indeed, it is easy to see that the preceding theorem continues to hold if we remove from $\mathsf{RewriteIQ}(A, \mathcal{T})$ and $\mathsf{RewriteIQ}(r, \mathcal{T})$ all disjuncts that contain a concept or role name that does not belong to Σ.

We continue our previous example to illustrate the rewriting construction.

Example 12. Consider the IQ $q_2 = \mathsf{Dish}(x)$ and the TBox $\mathcal{T}_2$ from Example 11. We have seen that $\mathsf{ComputeSubsumees}(\mathsf{Dish}, \mathcal{T}_2)$ contains the following concepts:

$$\mathsf{Dish}, \mathsf{ItalianDish}, \mathsf{VegDish}, \exists \mathsf{hasCourse}^-, \exists \mathsf{hasMain}^-, \exists \mathsf{hasDessert}^-$$

We will therefore obtain the following rewriting of q_2 w.r.t. $\mathcal{T}_2$ relative to consistent ABoxes:

$$\begin{aligned}\mathsf{RewriteIQ}(\mathsf{Dish}, \mathcal{T}_2) = \mathsf{Dish}(x) \lor \mathsf{ItalianDish}(x) \lor \mathsf{VegDish}(x) \lor \exists y.\mathsf{hasCourse}(y, x) \\ \lor \exists y.\mathsf{hasMain}(y, x) \lor \exists y.\mathsf{hasDessert}(y, x)\end{aligned}$$

In fact, because $\mathcal{T}_2$ does not contain any inclusions expressing disjointness, we know that every ABox is consistent with $\mathcal{T}_2$, and so the preceding rewriting will give the correct result for all ABoxes. If we evaluate the query $\mathsf{RewriteIQ}(\mathsf{Dish}, \mathcal{T}_2)$ over the ABox consisting of the assertions

$$\mathsf{hasMain}(m, d_1) \quad \mathsf{hasDessert}(m, d_2) \quad \mathsf{VegDish}(d_3)$$

then we will obtain the following certain answers:

- d_1, because of the disjunct $\exists y.\mathsf{hasMain}(y, x)$
- d_2, because of the disjunct $\exists y.\mathsf{hasDessert}(y, x)$
- d_3, because of the disjunct $\mathsf{VegDish}(x)$ ▲

For unsatisfiability, we proceed in two steps, first showing how to define an FO query that detects violation of a single disjointness constraint, and then showing how these rewritings can be combined to obtain a rewriting of unsatisfability. For negative concept inclusion $A \sqsubseteq \neg B$, we can use the following Boolean query that checks for the existence of an individual belonging to $A \sqcap B$ by considering all possible ways of choosing a subsumee of A and a subsumee of B:

$$\mathsf{RewriteDisjoint}(A, B, \mathcal{T}) = \bigvee_{\substack{C \in \mathsf{ComputeSubsumees}(A, \mathcal{T}) \\ D \in \mathsf{ComputeSubsumees}(B, \mathcal{T})}} \exists x.(\rho_x(C) \land \rho_x(D))$$

For a negative role inclusion $R \sqsubseteq \neg S$, we can define in a similar fashion a Boolean query that checks if there exists a pair of individuals that belongs to both of the roles R and S:

$$\mathsf{RewriteDisjoint}(R, S, \mathcal{T}) = \bigvee_{\substack{U \in \mathsf{ComputeSubroles}(R, \mathcal{T}) \\ V \in \mathsf{ComputeSubroles}(S, \mathcal{T})}} \exists x, y.(\rho_{x,y}(U) \land \rho_{x,y}(V))$$

To obtain a rewriting of unsatisfiability w.r.t. $\mathcal{T}$, it then suffices to take the disjunction of the FO queries associated with the negative inclusions in $\mathcal{T}$:

$$\mathsf{RewriteUnsat}(\mathcal{T}) = \bigvee_{G \sqsubseteq \neg H \in \mathcal{T}} \mathsf{RewriteDisjoint}(G, H, \mathcal{T})$$

Indeed, it can be shown that a DL-Lite$_R$ KB $(\mathcal{T}, \mathcal{A})$ is unsatisfiable if and only if one of the negative inclusions in $\mathcal{T}$ is violated.

Theorem 4. *For every finite signature Σ and DL-Lite$_R$ TBox $\mathcal{T}$, the query* $\mathsf{RewriteUnsat}(\mathcal{T})$ *is an FO-rewriting of unsatisfiability w.r.t. $\mathcal{T}, \Sigma$.*

The following example illustrates the construction of a rewriting of unsatisfiability and how such a rewriting can be combined with a rewriting of an IQ relative to consistent ABoxes to obtain a rewriting that works for all ABoxes.

Example 13. We consider a variant $\mathcal{T}_3$ of the preceding TBox that contains two disjointness constraints:

$$\exists \mathsf{hasCourse}^- \sqsubseteq \mathsf{Dish} \quad \mathsf{hasMain} \sqsubseteq \mathsf{hasCourse} \quad \mathsf{hasDessert} \sqsubseteq \mathsf{hasCourse}$$
$$\mathsf{hasMain} \sqsubseteq \neg \mathsf{hasDessert} \quad \mathsf{Dish} \sqsubseteq \neg \exists \mathsf{hasCourse}$$

For the first negative inclusion $\mathsf{hasMain} \sqsubseteq \neg \mathsf{hasDessert}$, we obtain the following FO query:

$$\mathsf{RewriteDisjoint}(\mathsf{hasMain}, \mathsf{hasDessert}, \mathcal{T}_3) = \exists x, y \mathsf{hasMain}(x, y) \wedge \mathsf{hasDessert}(x, y)$$

since $\mathsf{hasMain}$ and $\mathsf{hasDessert}$ do not have any subroles (aside from themselves). For the second negative inclusion $\mathsf{Dish} \sqsubseteq \neg \exists \mathsf{hasCourse}$, each of the concepts Dish and $\exists \mathsf{hasCourse}$ has multiple subsumees:

$$\mathsf{ComputeSubsumees}(\mathsf{Dish}, \mathcal{T}_3) = \{\mathsf{Dish}, \exists \mathsf{hasCourse}^-, \exists \mathsf{hasMain}^-, \exists \mathsf{hasDessert}^-\}$$
$$\mathsf{ComputeSubsumees}(\exists \mathsf{hasCourse}, \mathcal{T}_3) = \{\exists \mathsf{hasCourse}, \exists \mathsf{hasMain}, \exists \mathsf{hasDessert}\}$$

The FO query $\mathsf{RewriteDisjoint}(\mathsf{Dish}, \exists \mathsf{hasCourse}, \mathcal{T}_3)$ expressing the violation of $\mathsf{Dish} \sqsubseteq \neg \exists \mathsf{hasCourse}$ will contain 12 disjuncts, corresponding to the 4 choices of a subsumee of Dish and the 3 choices for $\exists \mathsf{hasCourse}$:

$$\bigvee_{r \in \{\mathsf{hasCourse}, \mathsf{hasMain}, \mathsf{hasDessert}\}} \exists x.(\mathsf{Dish}(x) \vee \exists y.r(x, y)) \vee \bigvee_{r,s \in \{\mathsf{hasCourse}, \mathsf{hasMain}, \mathsf{hasDessert}\}} \exists x.(\exists y.r(y, x) \wedge \exists y.s(x, y))$$

Combining the preceding FO queries yields the following rewriting of unsatisfiability w.r.t. $\mathcal{T}_3$:

$$\begin{aligned}\mathsf{RewriteUnsat}(\mathcal{T}) = {} & \mathsf{RewriteDisjoint}(\mathsf{hasMain}, \mathsf{hasDessert}, \mathcal{T}) \\ & \vee \mathsf{RewriteDisjoint}(\mathsf{Dish}, \exists \mathsf{hasCourse}, \mathcal{T})\end{aligned}$$

In order to construct an FO-rewriting of $\mathsf{Dish}(x)$ w.r.t. $\mathcal{T}_3$, we will need to compute $\mathsf{RewriteIQ}(\mathsf{Dish}, \mathcal{T}_3)$, which can be done as in Example 12. We will also need to construct a query that returns all individuals that appear in some ABox assertion. The definition of this query will depend on the ABox signature Σ. If we take $\Sigma = \mathsf{sig}(\mathcal{T}_3)$, then we could use the following query:

$$q_{\mathsf{ind}}^{\Sigma}(x) = \mathsf{Dish}(x) \vee \exists y.\mathsf{hasCourse}(x,y) \vee \exists y.\mathsf{hasMain}(x,y) \vee \exists y.\mathsf{hasDessert}(x,y) \\ \vee \exists y.\mathsf{hasCourse}(y,x) \vee \exists y.\mathsf{hasMain}(y,x) \vee \exists y.\mathsf{hasDessert}(y,x)$$

By combining these queries, we obtain a rewriting of $q_2 = \mathsf{Dish}(x)$ w.r.t. $\mathcal{T}_3, \mathsf{sig}(\mathcal{T}_3)$:

$$\mathsf{RewriteIQ}(\mathsf{Dish}, \mathcal{T}_3) \vee (\mathsf{RewriteUnsat}(\mathcal{T}_3) \wedge q_{\mathsf{ind}}^{\Sigma})$$

Now let us consider what happens when we evaluate the preceding query over a $\mathsf{sig}(\mathcal{T}_3)$-ABox $\mathcal{A}$. When $(\mathcal{T}_3, \mathcal{A})$ is satisfiable, $\mathsf{RewriteUnsat}(\mathcal{T}_3)$ evaluates to false over $\mathcal{I}_{\mathcal{A}}$, and so we must satisfy the first disjunct $\mathsf{RewriteIQ}(\mathsf{Dish}, \mathcal{T}_3)$. If $(\mathcal{T}_3, \mathcal{A})$ is unsatisfiable, then $\mathsf{RewriteUnsat}(\mathcal{T}_3)$ evaluates to true and q_{ind} will retrieve all of the individuals that appear in $\mathcal{A}$. ▲

We have shown that for every IQ q, DL-Lite$_R$ TBox $\mathcal{T}$, and ABox signature Σ, it is possible to construct an FO rewriting of q w.r.t. $\mathcal{T}, \Sigma$, and thus we can reduce instance checking to FO query evaluation over databases. Importantly, this reduction is independent of the ABox, which means that the instance checking problem has the same low data complexity as the evaluation of FO queries over databases.

Theorem 5 (follows from results in [52], see also [10]). *In DL-Lite$_R$, satisfiability and instance checking are in AC_0 for data complexity.*

Regarding combined complexity, it is possible to obtain a P upper bound by observing that the rewriting procedure runs in polynomial time in $|\mathcal{T}|$ and produces an FO query that, because of its restricted syntax, can be answered in polynomial time in $|\mathcal{A}|$ (recall that when we analyze the complexity of query answering, we consider the decision problem of testing whether a given tuple is a (certain) answer). This upper bound can be improved to NLogSpace by employing a non-deterministic logarithmic space procedure that guesses a single disjunct in the rewriting of the IQ and verifies that the input tuple satisfies this disjunct. An alternative proof of NLogSpace membership proceeds by reducing instance checking in DL-Lite$_R$ to the satisfiablity problem of first-order Krom formulas, which is known to be complete for NLogSpace [10].

Theorem 6 ([10]). *In DL-Lite$_R$, satisfiability and instance checking are both* NLogSpace*-complete for combined complexity.*

We remark that the preceding complexity results hold not only for DL-Lite$_R$ but also for several other DL-Lite dialects (like DL-Lite$_F$ and DL-Lite$_A$) and can be shown using similar techniques (see [52, 10] for more details).

4.2 Saturation-based Procedure for Instance Checking in $\mathcal{EL}$

We will next consider the problem of instance checking in $\mathcal{EL}$, which is the basic member of the $\mathcal{EL}$ family of Horn DLs. Unlike DL-Lite$_R$, the first-order query rewriting approach cannot be used in general to handle $\mathcal{EL}$ KBs, since there exist pairs of IQs and TBoxes for which no FO rewriting exists (a concrete example was provided in Example 9). Instead, we will show how instance checking can be performed using a simple saturation-based approach. As we shall in the next subsection, this approach can be extended to handle more expressive Horn DLs.

To simplify the presentation of the saturation procedure, we will assume that the considered $\mathcal{EL}$ KBs have been normalized, that is, they only contain TBox inclusions of the following forms:

$$A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B \quad A \sqsubseteq \exists r.B \quad \exists r.A \sqsubseteq B$$

where $A_{(i)}, B \in \mathsf{N_C} \cup \{\top\}$, $r \in \mathsf{N_R}$, and $n \geq 1$. As mentioned in Section 2, this assumption is without loss of generality since every $\mathcal{EL}$ KB $\mathcal{K}$ can be transformed into a normalized KB $\mathcal{K}'$ that has the same logical consequences as $\mathcal{K}$ over the signature of $\mathcal{K}$. This transformation can be performed in polynomial time and will not impact the complexity results obtained in this subsection.

In Table 3, we present a set of five *saturation (or inference) rules* that can be used to infer new inclusions and assertions from a given $\mathcal{EL}$ knowledge base. These rules essentially correspond to a subset of the rules proposed in [19] for reasoning in an extension of $\mathcal{EL}$ called $\mathcal{EL}^{++}$, but use a syntax that is closer to that found in recent works on consequence-based reasoning in DLs. Each of the rules in Table 3 acts as a template that can be instantiated using different concept and role names to obtain a *rule instantiation* of the form

$$\frac{\alpha_1, \ldots, \alpha_n}{\beta}$$

where $\alpha_1, \ldots, \alpha_n, \beta$ are TBox inclusions or ABox assertions. We call $\alpha_1 \ldots \alpha_n$ the *premises* of the rule instantiation and β its *conclusion*. A rule instantiation ρ is said to be *applicable* to a KB $\mathcal{K}$ if $\mathcal{K}$ contains all of the the premises of ρ but does not contain ρ's conclusion. If ρ is applicable to $\mathcal{K}$, then applying it means adding the conclusion of ρ to $\mathcal{K}$.

Example 14. Consider the knowledge base $\mathcal{K}_2$ that comprises the inclusions:

$$\mathsf{PenneArrabiata} \sqsubseteq \mathsf{Dish} \quad \mathsf{PenneArrabiata} \sqsubseteq \mathsf{Spicy} \quad \mathsf{Spicy} \sqcap \mathsf{Dish} \sqsubseteq \mathsf{SpicyDish}$$

The following instantiation of rule **T1** is applicable to $\mathcal{K}_2$:

$$\frac{\mathsf{PenneArrabiata} \sqsubseteq \mathsf{Spicy} \quad \mathsf{PenneArrabiata} \sqsubseteq \mathsf{Dish} \quad \mathsf{Spicy} \sqcap \mathsf{Dish} \sqsubseteq \mathsf{SpicyDish}}{\mathsf{PenneArrabiata} \sqsubseteq \mathsf{SpicyDish}}$$

To apply this rule instantiation, we add $\mathsf{PenneArrabiata} \sqsubseteq \mathsf{SpicyDish}$ to $\mathcal{K}_2$. ▲

$$\frac{A \sqsubseteq B_i \ (1 \leq i \leq n) \quad B_1 \sqcap \ldots \sqcap B_n \sqsubseteq D}{A \sqsubseteq D} \ \textsf{T1} \qquad \frac{A \sqsubseteq B \quad B \sqsubseteq \exists r.D}{A \sqsubseteq \exists r.D} \ \textsf{T2}$$

$$\frac{A \sqsubseteq \exists r.B \quad B \sqsubseteq D \quad \exists r.D \sqsubseteq E}{A \sqsubseteq E} \ \textsf{T3}$$

$$\frac{A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B \quad A_i(a) \ (1 \leq i \leq n)}{B(a)} \ \textsf{A1} \qquad \frac{\exists r.B \sqsubseteq A \quad r(a,b) \quad B(b)}{A(a)} \ \textsf{A2}$$

Table 3: Saturation rules for $\mathcal{EL}$. Here $r \in \mathsf{N_R}$ and $A, B, D, E \in \mathsf{N_C} \cup \{\top\}$.

Note that in what follows, we will slightly abuse terminology and speak simply of rules and rule applications, rather than (applications of) rule instantiations.

By inspecting the rules in Table 3, we immediately observe that because of the syntactic restrictions on the conclusions of the saturation rules, there are only finitely many axioms and assertions that can be produced over a given finite signature. It follows that an exhaustive application of the saturation rules to a KB is guaranteed to terminate and produce a finite (saturated) KB. Moreover, the result of the saturation process does not depend on the order in which the rules are applied, so we make speak of *the* saturation of a KB.

The following example illustrates the computation of the saturation of a KB.

Example 15. Consider the $\mathcal{EL}$ KB $\mathcal{K}_3$ whose TBox contains the inclusions

$$\mathsf{PenneArrabiata} \sqsubseteq \exists \mathsf{hasIngredient}.\mathsf{ArrabiataSauce} \tag{13}$$
$$\mathsf{PenneArrabiata} \sqsubseteq \mathsf{PastaDish} \tag{14}$$
$$\mathsf{PastaDish} \sqsubseteq \mathsf{Dish} \tag{15}$$
$$\mathsf{PastaDish} \sqsubseteq \exists \mathsf{hasIngredient}.\mathsf{Pasta} \tag{16}$$
$$\mathsf{ArrabiataSauce} \sqsubseteq \exists \mathsf{hasIngredient}.\mathsf{Peperoncino} \tag{17}$$
$$\mathsf{Peperoncino} \sqsubseteq \mathsf{Spicy} \tag{18}$$
$$\exists \mathsf{hasIngredient}.\mathsf{Spicy} \sqsubseteq \mathsf{Spicy} \tag{19}$$
$$\mathsf{Spicy} \sqcap \mathsf{Dish} \sqsubseteq \mathsf{SpicyDish} \tag{20}$$

and whose ABox consists of the single assertion

$$\mathsf{PenneArrabiata}(p). \tag{21}$$

If we apply the saturation rules from Table 3, then we obtain the new axioms and assertions listed below. Note that we indicate on the right the rule that was

applied, followed by the axioms and/or assertions used as premises.

$\mathsf{ArrabiataSauce} \sqsubseteq \mathsf{Spicy}$	**T3** : (17), (18), (19)	(22)
$\mathsf{PenneArrabiata} \sqsubseteq \mathsf{Spicy}$	**T3** : (13), (22), (19)	(23)
$\mathsf{PenneArrabiata} \sqsubseteq \mathsf{Dish}$	**T1** : (14), (15)	(24)
$\mathsf{PenneArrabiata} \sqsubseteq \exists \mathsf{hasIngredient.Pasta}$	**T2** : (14), (16)	(25)
$\mathsf{PenneArrabiata} \sqsubseteq \mathsf{SpicyDish}$	**T1** : (23), (24), (20)	(26)
$\mathsf{Spicy}(p)$	**A1** : (23), (21)	(27)
$\mathsf{Dish}(p)$	**A1** : (24), (21)	(28)
$\mathsf{SpicyDish}(p)$	**A1** : (28), (27)	(29)

It can be verified that nothing further can be inferred using the rules. ▲

In addition to ensuring finite termination, the saturation rules from Table 3 possess two other important properties. First, they are *sound*, that is, they only allow us to derive axioms and assertions that are logical consequences. Secondly, they are *complete for instance checking*, by which we mean that allow us to derive all entailed ABox assertions. These properties can be established by adapting proofs of similar results in [19].

Theorem 7. *Let $\mathcal{K}$ be an $\mathcal{EL}$ knowledge base, and let $\mathcal{K}'$ be obtained by exhaustively applying the rules in Table 3 to $\mathcal{K} \cup \{A \sqsubseteq A, A \sqsubseteq \top \mid A \in \mathsf{N_C} \cap \mathsf{sig}(\mathcal{K})\} \cup \{\top \sqsubseteq \top\} \cup \{\top(a) \mid a \in \mathsf{Ind}(\mathcal{A})\}$ up to saturation. Then for every ABox assertion α, we have $\mathcal{K} \models \alpha$ iff $\alpha \in \mathcal{K}'$.*

Remark 11. To ensure completeness, before running the saturation rules, we first add to the KB some trivially entailed inclusions (of the forms $A \sqsubseteq A, A \sqsubseteq \top$) and assertions[11] (of the form $\top(a)$). Alternatively, we could introduce saturation rules with empty premises that generate these inclusions and assertions. In practice, one could simply allow these inclusions and assertions to be used as premises during the saturation process, without adding them.

Remark 12. The rules in Table 3 do not allow us to generate *all* entailed concept inclusions, and indeed, this is a good thing since there can be infinitely many (non-equivalent) concept inclusions that are entailed from a given KB. However, we can show that these rules are sufficient to obtain all entailed concept inclusions between concept names (we say that the rule calculus is *complete for classification*).

By Theorem 7, we can perform instance checking by exhaustively applying the saturation rules to the KB, and then checking if the resulting saturated KB contains the desired assertion.

[11] In this section, it will prove convenient to allow ABox assertions using the atomic concepts $\top$ and $\bot$, in addition to concept names. Refer to Section 2 for discussion.

Example 16. Reconsider the KB $\mathcal{K}_3$ and its saturation from Example 15. The individual p is a certain answer to the IQ $\mathsf{SpicyDish}(x)$ w.r.t. $\mathcal{K}_3$ since the assertion $\mathsf{SpicyDish}(p)$ is present in the saturation of $\mathcal{K}_3$. However, p is not a certain answer to $\mathsf{Peperoncino}(x)$ w.r.t. $\mathcal{K}_3$, since $\mathsf{Peperoncino}(p)$ was not derived. ▲

A closer inspection reveals that the saturation procedure runs in polynomial time in $|\mathcal{K}|$. Indeed, at each iteration, we must produce at least one new concept inclusion or ABox assertion of one of the following forms:

$$A \sqsubseteq B \qquad A \sqsubseteq \exists r.B \qquad A(a) \qquad r(a,b)$$

which is built using only the individuals, concept names and role names from $\mathcal{K}$, and there are only polynomially many such axioms and assertions. We therefore obtain a P upper bound on the combined complexity of instance checking in $\mathcal{EL}$. This result was first established in [19], and a matching P lower bound for data complexity was provided in [59].

Theorem 8 ([19, 59]). *Instance checking in $\mathcal{EL}$ is* P*-complete for both the data and combined complexity measures.*

4.3 Instance Checking in $\mathcal{ELHI}_\bot$

In this final subsection on instance checking, we move to a richer Horn DL, $\mathcal{ELHI}_\bot$, which integrates constructors from DL-Lite$_R$ and $\mathcal{EL}$. We will present a saturation procedure for $\mathcal{ELHI}_\bot$ that is similar in spirit to our $\mathcal{EL}$ saturation procedure but contains additional rules to handle the new constructors. This saturation procedure will provide a method of performing satisfiability and instance checking over $\mathcal{ELHI}_\bot$ KBs. Moreover, we shall see that inclusions obtained by saturating the TBox can be used to built Datalog rewritings of satisfiability and instance queries. The $\mathcal{ELHI}_\bot$ saturation rules will resurface again in Section 5, where they will be used to construct universal models.

As in the preceding subsection, we will assume that the input $\mathcal{ELHI}_\bot$ KB is in normal form, i.e., it contains only TBox inclusions of the forms

$$A_1 \sqcap \ldots \sqcap A_n \sqsubseteq D \quad A \sqsubseteq \exists R.B \quad \exists R.A \sqsubseteq B \quad R \sqsubseteq S$$

where $A_{(i)}, B \in \mathsf{N_C} \cup \{\top\}$, $D \in \mathsf{N_C} \cup \{\top, \bot\}$, $R, S \in \mathsf{N_R^{\pm}}$, and $n \geq 1$.

The saturation rules for $\mathcal{ELHI}_\bot$ are displayed in Table 4. They have been adapted from the saturation calculus Horn-$\mathcal{SHIQ}$ from [81] (itself adapted from an earlier calculus from [117]). Rules **T4** and rule **T5** handle role inclusions and the $\bot$ concept respectively. Rules **T7** is a more elaborate version of the $\mathcal{EL}$ rule **T3**, adapted to handle the presence of role inclusions and inverse roles. Note that due to the presence of inverse roles, it is necessary to allow conjunctions in existential concepts. Rule **T6** provides a means of introducing new concepts into such conjunctions. To understand rule **T8**, it is helpful to recall that $\exists \mathsf{inv}(S).A \sqsubseteq B$ can be equivalently expressed as $A \sqsubseteq \forall S.B$. The rule adds A as a condition on the right-hand side, which allows B to be added to the existential on the left-hand

$$\frac{\{A \sqsubseteq B_i\}_{i=1}^n \quad B_1 \sqcap \ldots \sqcap B_n \sqsubseteq D}{A \sqsubseteq D}\ \textsf{T1} \qquad \frac{R \sqsubseteq S \quad S \sqsubseteq T}{R \sqsubseteq T}\ \textsf{T4} \qquad \frac{M \sqsubseteq \exists R.(N \sqcap \bot)}{M \sqsubseteq \bot}\ \textsf{T5}$$

$$\frac{M \sqsubseteq \exists R.(N \sqcap N') \quad N \sqsubseteq A}{M \sqsubseteq \exists R.(N \sqcap N' \sqcap A)}\ \textsf{T6} \qquad \frac{M \sqsubseteq \exists R.(N \sqcap A) \quad \exists S.A \sqsubseteq B \quad R \sqsubseteq S}{M \sqsubseteq B}\ \textsf{T7}$$

$$\frac{M \sqsubseteq \exists R.N \quad \exists \mathsf{inv}(S).A \sqsubseteq B \quad R \sqsubseteq S}{M \sqcap A \sqsubseteq \exists R.(N \sqcap B)}\ \textsf{T8}$$

$$\frac{A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B \quad A_i(a)\ (1 \leq i \leq n)}{B(a)}\ \textsf{A1} \qquad \frac{\exists r.B \sqsubseteq A \quad r(a,b) \quad B(b)}{A(a)}\ \textsf{A2}$$

$$\frac{\exists r^-.B \sqsubseteq A \quad r(b,a) \quad B(b)}{A(a)}\ \textsf{A3} \qquad \frac{r \sqsubseteq s \quad r(a,b)}{s(a,b)}\ \textsf{A4} \qquad \frac{r \sqsubseteq s^- \quad r(a,b)}{s(b,a)}\ \textsf{A5}$$

Table 4: Saturation rules for $\mathcal{ELHI}_\bot$. Here $r \in \mathsf{N_R}$, $R, S \in \mathsf{N_R^\pm}$, $A, B \in \mathsf{N_C} \cup \{\top, \bot\}$, and M and $N^{(\prime)}$ are conjunctions of concepts from $\mathsf{N_C} \cup \{\top, \bot\}$.

side of the inclusion. Finally, we introduce three additional ABox saturation rules. Rule **A3** is like **A2** except that is concerns inverse roles, and rules **A4**-**A5** are used to infer new role assertions using derived role inclusions.

The notions of rule instantiation, applicable rule, and rule application are essentially the same as for $\mathcal{EL}$. Note however that when working with axioms containing conjunctions, we will *treat conjunctions as sets*. That is, we will assume that there are no repeated conjuncts, and we will not pay attention to the order of conjuncts. Thus, if a rule instantiation contains a premise $A \sqsubseteq \exists R.(B \sqcap C \sqcap D)$, and the KB contains the equivalent (but syntactically distinct) inclusion $A \sqsubseteq \exists R.(D \sqcap B \sqcap C)$, then we will consider that the premise is present in the KB when deciding whether the rule instantiation is applicable. Likewise, if a rule instantiation has $D \sqsubseteq \exists R.(A \sqcap B)$ as a conclusion, and the KB already contains $D \sqsubseteq \exists R.(B \sqcap A)$, then the rule instantiation will not be considered applicable.

We are now ready to describe the saturation procedure. Given an $\mathcal{ELHI}_\bot$ knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, we first enrich the KB with inclusions and assertions that are trivially entailed:

$$\begin{aligned}\mathcal{T}' = \mathcal{T} \cup \{&A \sqsubseteq A, A \sqsubseteq \top \mid A \in (\mathsf{N_C} \cap \mathsf{sig}(\mathcal{K})) \cup \{\top\}\} \\ &\cup \{\mathsf{inv}(R) \sqsubseteq \mathsf{inv}(S) \mid S \sqsubseteq R \in \mathcal{T}\} \\ \mathcal{A}' = \mathcal{A} \cup \{&\top(a) \mid a \in \mathsf{Ind}(\mathcal{A})\}\end{aligned}$$

We then exhaustively apply the rules in Table 4 to $(\mathcal{T}', \mathcal{A}')$ until nothing new can be derived (finite termination is guaranteed due to the restricted syntax of the inclusions in rule conclusions). We will denote the resulting KB by $\mathsf{saturate}(\mathcal{K})$. If we instead apply the rules only to $\mathcal{T}'$, then we will use $\mathsf{saturate}(()\mathcal{T})$ to denote the resulting TBox.

The next theorem resumes the key properties of $\mathsf{saturate}(\mathcal{K})$. It can be proven by adapting the proofs of similar results for Horn-$\mathcal{SHIQ}$ [81].

Theorem 9. *For every $\mathcal{ELHI}_\bot$ knowledge base $\mathcal{K}$, we have:*

1. $\mathcal{K} \models \alpha$ *for every* $\alpha \in \mathsf{saturate}(\mathcal{K})$.
2. *If* $\mathcal{K}$ *is unsatisfiable, then* $\bot(a) \in \mathsf{saturate}(\mathcal{K})$ *for some* $a \in \mathsf{Ind}(\mathcal{K})$.
3. *If* $\mathcal{K}$ *is satisfiable and* $\mathcal{K} \models \alpha$ *with* α *an ABox assertion, then* $\alpha \in \mathsf{saturate}(\mathcal{K})$.

By the preceding theorem, to determine whether a given KB is satisfiable, it suffices to compute $\mathsf{saturate}(\mathcal{K})$ and check whether it contains an assertion of the form $\bot(a)$. Instance checking is also trivial once $\mathsf{saturate}(\mathcal{K})$ has been computed: to test whether $\mathcal{K} \models \alpha$, with $\mathcal{K}$ a satisfiable KB and α an ABox assertion, we merely need to check whether α appears in $\mathsf{saturate}(\mathcal{K})$.

Example 17. Let $\mathcal{K}_4$ be the $\mathcal{ELHI}_\bot$ KB consisting of the TBox $\mathcal{T}_4$:

$$\exists \mathsf{contains}^-.\mathsf{VegFriendly} \sqsubseteq \mathsf{VegFriendly} \tag{30}$$
$$\mathsf{hasIngredient} \sqsubseteq \mathsf{contains} \tag{31}$$
$$\mathsf{Meat} \sqcap \mathsf{VegFriendly} \sqsubseteq \bot \tag{32}$$
$$\mathsf{Bolognese Sauce} \sqsubseteq \exists \mathsf{hasIngredient}.\mathsf{Meat} \tag{33}$$

and the ABox whose assertions are:

$$\mathsf{VegFriendly}(d) \tag{34}$$
$$\mathsf{hasIngredient}(d, b) \tag{35}$$
$$\mathsf{Bolognese Sauce}(b) \tag{36}$$

We observe that $\mathcal{K}_4$ is unsatisfiable. Indeed, the inclusion $\exists \mathsf{contains}^-.\mathsf{VegFriendly} \sqsubseteq \mathsf{VegFriendly}$ and assertions $\mathsf{VegFriendly}(d)$ and $\mathsf{hasIngredient}(d, b)$ together imply that b is VegFriendly. We also know that b has an ingredient of type Meat, due to $\mathsf{Bolognese Sauce}(b)$ and $\mathsf{Bolognese Sauce} \sqsubseteq \exists \mathsf{hasIngredient}.\mathsf{Meat}$. Since hasIngredient is a subrole of contains, it follows that b contains this unnamed ingredient. We can therefore use inclusion (30) to conclude that this ingredient is VegFriendly. This contradicts the disjointness constraint $\mathsf{Meat} \sqcap \mathsf{VegFriendly} \sqsubseteq \bot$ which states that it is not possible to belong to both Meat and VegFriendly.

We now show how the unsatisfiability of $\mathcal{K}_4$ can be discovered by means of the saturation rules from Table 4. To begin, we use rule **T8** and the inclusions (33), (30), and (31) to derive

$$\mathsf{Bolognese Sauce} \sqcap \mathsf{VegFriendly} \sqsubseteq \exists \mathsf{hasIngredient}.(\mathsf{Meat} \sqcap \mathsf{VegFriendly}) \tag{37}$$

Next, we can apply rule **T6** to the preceding inclusion and (32) to infer

$$\mathsf{Bolognese Sauce} \sqcap \mathsf{VegFriendly} \sqsubseteq \exists \mathsf{hasIngredient}.(\mathsf{Meat} \sqcap \mathsf{VegFriendly} \sqcap \bot) \tag{38}$$

Then, using the preceding inclusion and rule **T5**, we obtain

$$\mathsf{Bolognese Sauce} \sqcap \mathsf{VegFriendly} \sqsubseteq \bot \tag{39}$$

Finally, by applying the ABox saturation rules, we reach a contradiction:

$$\mathsf{contains}(d, b) \qquad \textbf{A4} : (31), (35) \tag{40}$$
$$\mathsf{VegFriendly}(b) \qquad \textbf{A3} : (31), (40) \tag{41}$$
$$\bot(b) \qquad \textbf{A1} : (15), (39) \tag{42}$$

Since $\bot(b)$ has been derived, we can conclude that $\mathcal{K}_4$ is unsatisfiable. ▲

A simple examination of the rules in Table 4 reveals that all derived axioms and assertions take one of the following forms:

$$R \sqsubseteq S \qquad M \sqsubseteq A \qquad M \sqsubseteq \exists R.N \qquad A(a) \qquad r(a, b)$$

where $r \in \mathsf{N_R}$, $R, S \in \mathsf{N_R^{\pm}}$, $A \in \mathsf{N_C} \cup \{\top, \bot\}$, M, N are conjunctions of concepts from $\mathsf{N_C} \cup \{\top, \bot\}$, and all individual names, concept names, and role names appear in $\mathcal{K}$. As the number of distinct (non-equivalent) axioms and assertions of these forms is at most single-exponential in the size of $\mathcal{K}$, it follows that $\mathsf{saturate}(\mathcal{K})$ can be computed in single-exponential time in $|\mathcal{K}|$, which yields an Exp upper bound for satisfiability and instance checking. This upper bound, which can be derived from Exp upper bounds for non-Horn DLs (see e.g., [71]), cannot be further improved. Indeed, matching Exp lower bounds follow from the Exp-hardness of subsumption in $\mathcal{ELI}$ [20].

Theorem 10. *In $\mathcal{ELHI}_\bot$, satisfiability and instance checking are* Exp*-complete in combined complexity.*

Observe that the TBox saturation rules do not depend on the ABox saturation rules, so it is possible to first fully saturate the TBox, and then in a second step, apply the ABox rules. We further remark that the ABox saturation rules can be viewed as Datalog rules which use concept names, role names, and the special concepts $\top$ and $\bot$ as predicate symbols. Formally, we can associate with each $\mathcal{ELHI}_\bot$ TBox $\mathcal{T}$ and ABox signature Σ the Datalog program $\Pi(\mathcal{T}, \Sigma)$ defined as follows:

$$\begin{aligned}
\Pi(\mathcal{T}, \Sigma) = &\{B(x) \leftarrow A_1(x), \ldots, A_n(x) \mid A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B \in \mathsf{saturate}(\mathcal{T})\} \cup \\
&\{B(x) \leftarrow A(y), r(x, y) \mid \exists r.A \sqsubseteq B \in \mathcal{T}\} \cup \\
&\{B(y) \leftarrow A(x), r(x, y) \mid \exists r^-.A \sqsubseteq B \in \mathcal{T}\} \cup \\
&\{s(x, y) \leftarrow r(x, y) \mid r \sqsubseteq s \in \mathsf{saturate}(\mathcal{T}), s \in \mathsf{N_R}\} \cup \\
&\{s(y, x) \leftarrow r(x, y) \mid r \sqsubseteq s^- \in \mathsf{saturate}(\mathcal{T}), s \in \mathsf{N_R}\} \cup \\
&\{\top(x) \leftarrow A(x) \mid A \in \mathsf{N_C} \cap \Sigma\} \cup \\
&\{\top(x) \leftarrow r(x, y) \mid r \in \mathsf{N_R} \cap \Sigma\} \cup \\
&\{\top(x) \leftarrow r(y, x) \mid r \in \mathsf{N_R} \cap \Sigma\}
\end{aligned}$$

Note that first five sets of Datalog rules making up $\Pi(\mathcal{T}, \Sigma)$ are in one-to-one correspondence with the five ABox saturation rules **A1**-**A5**, with rules in the i-th line of the definition of $\Pi(\mathcal{T}, \Sigma)$ corresponding the ABox saturation rule **A**i (for $1 \leq i \leq 5$). The last three sets of Datalog rules merely serve to populate $\top$ with all of the individuals in the ABox.

Example 18. Consider again the TBox $\mathcal{T}_4$ from Example 17, and let $\Sigma = \mathsf{sig}(\mathcal{T}_4)$. The Datalog program $\Pi(\mathcal{T}, \Sigma)$ associated with $\mathcal{T}_4$ and Σ contains the rules:

$$\begin{aligned}
\bot(x) &\leftarrow \mathsf{Meat}(x), \mathsf{VegFriendly}(x) \\
\bot(x) &\leftarrow \mathsf{BolognesseSauce}(x), \mathsf{VegFriendly}(x) \\
\mathsf{VegFriendly}(y) &\leftarrow \mathsf{VegFriendly}(x), \mathsf{hasIngredient}(x, y) \\
\mathsf{contains}(x, y) &\leftarrow \mathsf{hasIngredient}(x, y)
\end{aligned}$$

each corresponding to an inclusion from $\mathsf{saturate}(\mathcal{T}_4)$. The program additionally contains rules for populating $\top$ (one rule for each concept name in $\mathcal{T}_4$, and two for each role name) and rules corresponding to the translations of trivial inclusions like $\mathsf{Meat} \sqsubseteq \mathsf{Meat}$ and $\mathsf{Meat} \sqsubseteq \top$ (these latter rules could simply be omitted). ▲

The following theorem, which is a consequence of Theorem 9, resumes the important properties of the Datalog program $\Pi(\mathcal{T}, \Sigma)$.

Theorem 11. *For every finite signature Σ and $\mathcal{ELHI}_\bot$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ with $\mathsf{sig}(\mathcal{A}) \subseteq \Sigma$:*

1. *$\mathcal{K}$ is unsatisfiable iff $\mathsf{ans}((\Pi(\mathcal{T}, \Sigma), \bot), \mathcal{I}_\mathcal{A}) \neq \emptyset$;*
2. *If $\mathcal{K}$ is satisfiable, then for all $A \in \mathsf{N_C}$, $r \in \mathsf{N_R}$, and $a, b \in \mathsf{Ind}(\mathcal{A})$:*
   - *$\mathcal{K} \models A(a)$ iff $a \in \mathsf{ans}((\Pi(\mathcal{T}, \Sigma), A), \mathcal{I}_\mathcal{A})$;*
   - *$\mathcal{K} \models r(a, b)$ iff $(a, b) \in \mathsf{ans}((\Pi(\mathcal{T}, \Sigma), r), \mathcal{I}_\mathcal{A})$.*

It follows from the first statement of the preceding theorem that the Datalog query $(\Pi(\mathcal{T}, \Sigma) \cup \{Q_\bot \leftarrow \bot(x)\}, Q_\bot)$ is a rewriting of unsatisfiability w.r.t. $\mathcal{T}, \Sigma$. The second statement asserts that $(\Pi(\mathcal{T}, \Sigma), A)$ (resp. $(\Pi(\mathcal{T}, \Sigma), r)$) is a Datalog rewriting of the IQ $A(x)$ (resp. $r(x, y)$) w.r.t. $\mathcal{T}, \Sigma$ relative to consistent ABoxes. As discussed in Section 3.5, these rewritings can be combined together in order to obtain Datalog rewritings that hold for all ABoxes.

Since the construction of the Datalog program $\Pi(\mathcal{T}, \Sigma)$ is independent of the ABox (and polynomial w.r.t. $|\Sigma|$), and Datalog query evaluation is P-complete in data complexity, we obtain a P upper bound on the data complexity of instance checking and satisfiability in $\mathcal{ELHI}_\bot$. This positive result, which was first established in [111] for Horn-$\mathcal{SHIQ}$, is the best that we could hope for given that instance checking (resp. satisfiability) is already P-hard in the sublogic $\mathcal{EL}$ (resp. $\mathcal{EL}_\bot$).

Theorem 12. *[111] In $\mathcal{ELHI}_\bot$, satisfiability and instance checking are* P*-complete in data complexity.*

5 (Unions of) Conjunctive Queries

Instance queries are rather limited as a query language, as they do not allow us to express the natural selections and joins over relations that are common in standard database query languages. For this reason, the majority of works on

OMQA in the last decade have adopted *conjunctive queries (CQs)* as the basic query language.

CQs are a special class of first-order queries which allow only for conjunctions of positive atoms and existential quantification. CQs capture the plain *select-project-join* fragment of relational algebra and SQL, as well as the *basic graph patterns* that lie at the heart of SPARQL [106], which is the standard query language for OWL and RDF. It has been documented that a large percentage of queries posed to industrial databases systems fall into this fragment. By taking disjunctions of such queries, sharing the same free variables, we obtain *unions of CQs*, another prominent query language. CQs and UCQs play a central role in traditional databases, and they are the query languages of choice in areas like data integration and data exchange [142, 6].

Definition 7 ((Unions of) conjunctive queries). *A* conjunctive query (CQ) *is a first-order query $q(\boldsymbol{x})$ of the form*

$$\exists \boldsymbol{y}.P_1(\boldsymbol{t_1}) \wedge \cdots \wedge P_n(\boldsymbol{t_n})$$

where every variable contained in some $\boldsymbol{t_i}$ is contained in either $\boldsymbol{x}$ or $\boldsymbol{y}$. Recall that the free variables $\boldsymbol{x}$ are called the answer variables of q, and that the arity of the query is the length of the tuple $\boldsymbol{x}$.

A union of CQs (UCQ) *is a first-order query $q(\boldsymbol{x})$ of the form*

$$q_1(\boldsymbol{x}) \vee \cdots \vee q_n(\boldsymbol{x})$$

where all the $q_i(\boldsymbol{x})$ are CQs with the same tuple $\boldsymbol{x}$ of answer variables.

Let $q(\boldsymbol{x}) = \exists \boldsymbol{y}.\varphi(\boldsymbol{x}, \boldsymbol{y})$ be a CQ. Recall that $\mathsf{ans}(q, \mathcal{I}) = \{\boldsymbol{e} = (e_1, \ldots, e_n) \in (\Delta^{\mathcal{I}})^n \mid \mathcal{I} \models \exists \boldsymbol{y}.\varphi[\boldsymbol{x} \mapsto \boldsymbol{e}]\}$. Since the variables in $\boldsymbol{y}$ are existentially quantified, we have that $\mathcal{I} \models \exists \boldsymbol{y}.\varphi[\boldsymbol{x} \mapsto \boldsymbol{e}]$ just in the case that there exists a variable assignment π that extends $(x_1, \ldots, x_n) \mapsto (e_1, \ldots, e_n)$ by additionally mapping the variables in $\boldsymbol{y}$ to objects in $\Delta^{\mathcal{I}}$ in such a way that $\mathcal{I} \models \varphi[\pi]$. We call such a mapping π a *match for q in $\mathcal{I}$*.

Remark 13. A popular alternative syntax for CQs and UCQs is to write them as Datalog rules. A CQ corresponds to a single Datalog rule

$$q(\boldsymbol{x}) = \exists \boldsymbol{y}.P_1(\boldsymbol{t_1}) \wedge \cdots \wedge P_n(\boldsymbol{t_n}) \quad \rightsquigarrow \quad q(\boldsymbol{x}) \leftarrow P_1(\boldsymbol{t_1}), \ldots, P_n(\boldsymbol{t_n})$$

while a UCQ is written as a set of rules with the same head predicate:

$$\begin{array}{rcl} q(\boldsymbol{x}) = & \exists \boldsymbol{y_1}.P_1^1(\boldsymbol{t_1^1}) \wedge \cdots \wedge P_{n_1}^1(\boldsymbol{t_{n_1}^1}) & q(\boldsymbol{x}) \leftarrow P_1^1(\boldsymbol{t_1^1}), \ldots, P_{n_1}^1(\boldsymbol{t_{n_1}^1}) \\ \vee & \exists \boldsymbol{y_2}.P_1^2(\boldsymbol{t_1^2}) \wedge \cdots \wedge P_{n_2}^2(\boldsymbol{t_{n_2}^2}) & q(\boldsymbol{x}) \leftarrow P_1^2(\boldsymbol{t_1^2}), \ldots, P_n^2(\boldsymbol{t_n^2}) \\ & \vdots & \rightsquigarrow \quad \vdots \\ \vee & \exists \boldsymbol{y_\ell}.P_1^\ell(\boldsymbol{t_1^\ell}) \wedge \cdots \wedge P_{n_\ell}^\ell(\boldsymbol{t_{n_\ell}^\ell}) & q(\boldsymbol{x}) \leftarrow P_1^\ell(\boldsymbol{t_1^\ell}), \ldots, P_{n_\ell}^\ell(\boldsymbol{t_{n_\ell}^\ell}) \end{array}$$

We now illustrate some queries that can be expressed as CQs or UCQs.

Example 19. Consider the following queries:

$$q_3(y,x) = \exists z.\mathsf{serves}(x,y) \land \mathsf{hasIngredient}(y,z) \land \mathsf{Spicy}(z)$$
$$q_4(y,x) = q_1(x,y) \lor \exists z, z'.\mathsf{serves}(x,y) \land \mathsf{hasIngredient}(y,z) \land \mathsf{hasIngredient}(z,z'), \mathsf{Spicy}(z')$$

The first query is a CQ that retrieves dishes y that contain a spicy ingredient, together with the establishment x where they are served. The query q_4 is a UCQ that finds pairs of y and x as in q_3, but it also retrieves the pair y, x if y has an ingredient that in turn contains a spicy ingredient. ▲

While some very restricted forms of (U)CQs can be expressed as instance queries by defining the query as a concept in the TBox, the arbitrary use of variables in CQs makes them a strict generalization of IQs.

We discuss in this section how to answer UCQs over $\mathcal{ELHI}_\bot$ knowledge bases. We assume in what follows that we are always given a satisfiable $\mathcal{K}$ as an input, since query answering over unsatisfiable knowledge bases is trivial, and we can test for satisfiability in advance using the procedure discussed in Section 4.3 (which has no higher complexity than any of the procedures described below).

5.1 Canonical Model Construction

As a preliminary step, we will show how to define a universal model of a given satisfiable $\mathcal{ELHI}_\bot$ KB using the saturated TBox obtained by applying the rules in Section 4.3. This universal model will play a central role in the query answering techniques developed in this and the following section.

Given a satisfiable $\mathcal{ELHI}_\bot$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, we consider the interpretation $\mathcal{I}_{\mathcal{T},\mathcal{A}}$ (alternatively denoted $\mathcal{I}_\mathcal{K}$) defined as follows. The domain $\Delta^{\mathcal{I}_{\mathcal{T},\mathcal{A}}}$ consists of sequences of the form $aR_1M_1 \dots R_nM_n$ ($n \geq 0$), where $a \in \mathsf{Ind}(\mathcal{A})$, and for every $i \geq 1$, $R_i \in \mathsf{N}_\mathsf{R}^\pm$ and M_i is a conjunction of concepts from $\mathsf{N}_\mathsf{C} \cup \{\top\}$. More precisely, $\Delta^{\mathcal{I}_{\mathcal{T},\mathcal{A}}}$ consists of all sequences $aR_1M_1 \dots R_nM_n$ that satisfy:

- If $n \geq 1$, then there exists $B_1 \sqcap \dots \sqcap B_m \sqsubseteq \exists R_1.M_1 \in \mathsf{saturate}(\mathcal{T})$ such that $B_j(a) \in \mathsf{saturate}(\mathcal{K})$ for every $1 \leq j \leq m$.
- For every $1 \leq i < n$, $M_i \sqsubseteq \exists R_{i+1}.M_{i+1} \in \mathsf{saturate}(\mathcal{T})$.

To complete the definition of $\mathcal{I}_{\mathcal{T},\mathcal{A}}$, we must fix the interpretation of the individual names, concept names, and role names from $\mathcal{K}$. This is done as follows[12]:

$$\begin{aligned}
a^{\mathcal{I}_{\mathcal{T},\mathcal{A}}} &= a \\
A^{\mathcal{I}_{\mathcal{T},\mathcal{A}}} &= \{a \in \mathsf{Ind}(\mathcal{A}) \mid A(a) \in \mathsf{saturate}(\mathcal{K})\} \cup \\
&\quad \{e \in \Delta^{\mathcal{I}_{\mathcal{T},\mathcal{A}}} \setminus \mathsf{Ind}(\mathcal{A}) \mid e = e'RM \text{ and } A \in M\} \\
r^{\mathcal{I}_{\mathcal{T},\mathcal{A}}} &= \{(a,b) \mid r(a,b) \in \mathsf{saturate}(\mathcal{K})\} \cup \\
&\quad \{(e_1,e_2) \mid e_2 = e_1 S\, M \text{ and } S \sqsubseteq r \in \mathsf{saturate}(\mathcal{T})\} \cup \\
&\quad \{(e_2,e_1) \mid e_2 = e_1 S\, M \text{ and } S \sqsubseteq r^- \in \mathsf{saturate}(\mathcal{T})\}
\end{aligned}$$

[12] Recall that we treat conjunctions of concepts as sets. Abusing notation, we use $A \in M$ to mean that A is a conjunct of M.

It is easy to show that $\mathcal{I}_\mathcal{K}$ is a model of $\mathcal{K}$, and we will henceforth refer to it as the *canonical model* of $\mathcal{K}$.

Note that the domain of $\mathcal{I}_\mathcal{K}$ contains the individuals in $\mathcal{A}$, and additional objects whose existence if implied by the axioms in $\mathsf{saturate}(\mathcal{T})$ of the form $M \sqsubseteq \exists R.N$. The latter objects are called *anonymous* and defined in such a way that if $aR_1M_1 \dots R_nM_n$ is in $\Delta^{\mathcal{I}_\mathcal{K}}$, then so is $aR_1M_1 \dots R_{n-1}M_{n-1}$. Hence these objects naturally form tree-like structures rooted at the individuals. Moreover, if we take the undirected graph that has the domain of $\mathcal{I}_\mathcal{K}$ as nodes and an (undirected) edge between two objects e, e' whenever $(e, e) \in r^{\mathcal{I}_\mathcal{K}}$ for some $r \in \mathsf{N_R}$, then we obtain a structure that can be viewed as comprising different parts:

- the restriction to the individuals, which is an arbitrary graph sometimes called the *core* of $\mathcal{I}_\mathcal{K}$
- a set of potentially infinite trees of anonymous objects, each of which is rooted at one of the individuals in the core, as we have discussed

For this reason, $\mathcal{I}_\mathcal{K}$ is often associated with a forest and it is given names such as a *(pseudo-) forest model*, see e.g. [90, 89, 58, 57]. This forest-like structure is a useful property that is exploited by many algorithms, and in particular by the ones we discuss in this chapter.

We now illustrate the construction of canonical models on a simple example:

Example 20. Consider the KB $\mathcal{K}_5$ whose TBox $\mathcal{T}_5$ contains the following axioms:

$$\begin{aligned}
\mathsf{PenneArrabiata} &\sqsubseteq \exists \mathsf{hasIngredient}.\mathsf{Penne} \\
\mathsf{Penne} &\sqsubseteq \mathsf{Pasta} \\
\mathsf{PenneArrabiata} &\sqsubseteq \exists \mathsf{hasIngredient}.\mathsf{ArrabiataSauce} \\
\mathsf{ArrabiataSauce} &\sqsubseteq \exists \mathsf{hasIngredient}.\mathsf{Peperoncino} \\
\mathsf{Peperoncino} &\sqsubseteq \mathsf{Spicy} \\
\mathsf{PizzaCalabrese} &\sqsubseteq \exists \mathsf{hasIngredient}.\mathsf{Nduja} \\
\mathsf{Nduja} &\sqsubseteq \mathsf{Spicy}
\end{aligned}$$

and whose ABox is as follows:

$$\mathsf{serves}(r, b) \qquad \mathsf{serves}(r, p) \qquad \mathsf{PenneArrabiata}(b) \qquad \mathsf{PizzaCalabrese}(p)$$

Note that $\mathsf{saturate}(\mathcal{T}_5)$ contains, additionally to the axioms above, the following axioms that result from applications of **T6**:

$$\begin{aligned}
\mathsf{PenneArrabiata} &\sqsubseteq \exists \mathsf{hasIngredient}.(\mathsf{Penne} \sqcap \mathsf{Pasta}) \\
\mathsf{ArrabiataSauce} &\sqsubseteq \exists \mathsf{hasIngredient}.(\mathsf{Peperoncino} \sqcap \mathsf{Spicy}) \\
\mathsf{PizzaCalabrese} &\sqsubseteq \exists \mathsf{hasIngredient}.(\mathsf{Nduja} \sqcap \mathsf{Spicy})
\end{aligned}$$

The canonical model $\mathcal{I}_{\mathcal{K}_5}$ of this knowledge base is depicted in Figure 3. For readability, we use the following abbreviations for the anonymous objects:

$e_1 = p\ \mathsf{hasIngredient}\ (\mathsf{Nduja} \sqcap \mathsf{Spicy})$

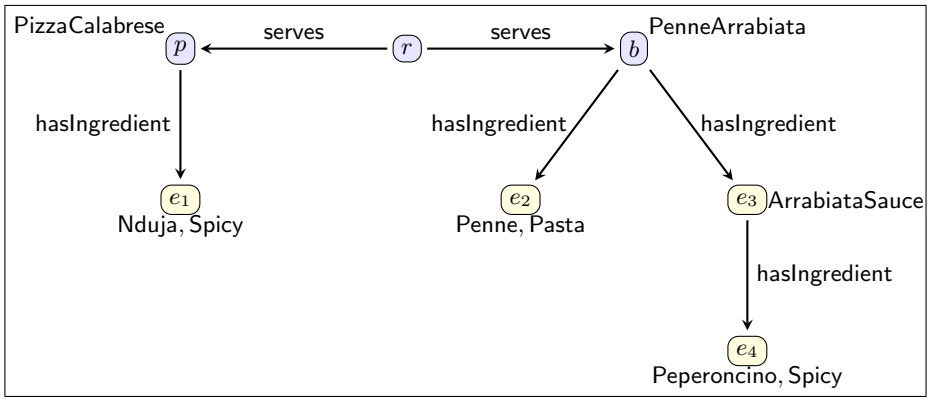

Fig. 3: Canonical model $\mathcal{I}_{\mathcal{K}_5}$ of the knowledge base $\mathcal{K}_5$ in Example 20. We use blue for ABox individuals and yellow for anonymous objects.

$e_2 = b$ hasIngredient (Penne $\sqcap$ Pasta)
$e_3 = b$ hasIngredient ArrabiataSauce
$e_4 = b$ hasIngredient ArrabiataSauce hasIngredient (Peperoncino $\sqcap$ Spicy) ▲

The crucial property of $\mathcal{I}_\mathcal{K}$ is that it is a *universal* model of $\mathcal{K}$, that is, it is 'contained' in every model of $\mathcal{K}$. For each model $\mathcal{I}$ of $\mathcal{K}$, we can define a *homomorphism* from $\mathcal{I}_\mathcal{K}$ to $\mathcal{I}$, which is a function $h : \Delta^{\mathcal{I}_\mathcal{K}} \to \Delta^\mathcal{I}$ such that

- $h(a^{\mathcal{I}_\mathcal{K}}) = a^\mathcal{I}$ for each individual a in $\mathcal{K}$,
- $e \in A^{\mathcal{I}_\mathcal{K}}$ implies $h(e) \in A^\mathcal{I}$ for every concept name A, and
- $(e, e') \in r^{\mathcal{I}_\mathcal{K}}$ implies $(h(e), h(e')) \in r^\mathcal{I}$ for every role name r.

It is not hard to see that matches of CQs are preserved under homomorphisms:

Fact 1 *Let q be a CQ, and h a homomorphism from an interpretation $\mathcal{I}$ to an interpretation $\mathcal{J}$. If π is a match for q in $\mathcal{I}$, then the mapping $h \circ \pi$ obtained by composing π with h is a match for q in $\mathcal{J}$.*

The importance of this property lies in the fact that, for an arbitrary CQ or UCQ q, every answer to q in $\mathcal{I}_\mathcal{K}$ is an answer to q in every model of $\mathcal{K}$. Since the converse also holds (an answer to q in every model is clearly also an answer to q in the particular model $\mathcal{I}_\mathcal{K}$), the certain answers to q over $\mathcal{K}$ coincide with the answers to q over $\mathcal{I}_\mathcal{K}$.

Theorem 13. *Let $\mathcal{K}$ be a satisfiable $\mathcal{ELHI}_\bot$ knowledge base, let $q(\boldsymbol{x}) = q_1(\boldsymbol{x}) \vee \ldots \vee q_n(\boldsymbol{x})$ be a UCQ, and let $\boldsymbol{a}$ be a tuple of individuals of the same arity as $\boldsymbol{x}$. Then $\boldsymbol{a} \in \mathsf{cert}(q, \mathcal{K})$ iff $\boldsymbol{a} \in \mathsf{ans}(q, \mathcal{I}_\mathcal{K})$ iff $\boldsymbol{a} \in \mathsf{ans}(q_i, \mathcal{I}_\mathcal{K})$ for some $1 \leq i \leq n$.*

This result has been shown even for significantly more expressive query languages than UCQs (for example, positive first-order queries and Datalog queries),

and it allows us to focus on the simpler problem of evaluating a CQ over the canonical model only, instead of considering the possibly infinitely many models that a knowledge base may possess.

Example 21. Let $\mathcal{K}_5$ and $\mathcal{T}_5$ be the knowledge base and TBox from Example 20, and recall the queries q_3 and q_4 from Example 19. The match with $x \mapsto r$, $y \mapsto p$ and $z \mapsto e_1$ shows that $(p, r) \in \mathsf{ans}(q_3, \mathcal{I}_{\mathcal{K}_5})$, and there are no further answers, hence $\mathsf{ans}(q_1, \mathcal{I}_{\mathcal{K}_5}) = \{(p, r)\}$. For q_4, we also have $(p, r) \in \mathsf{ans}(q_4, \mathcal{I}_{\mathcal{K}_5})$, but in this case, we additionally get $(b, r) \in \mathsf{ans}(q_4, \mathcal{I}_{\mathcal{K}_5})$, as witnessed by the assignment $x \mapsto r$, $y \mapsto b$, $z \mapsto e_3$ and $z' \mapsto e_4$, which is a match for the second disjunct. Hence $\mathsf{ans}(q_4, \mathcal{I}_{\mathcal{K}_5}) = \{(p, r), (b, r)\}$. We therefore obtain the following certain answers over $\mathcal{K}_5$:

$$\mathsf{cert}(q_3, \mathcal{K}_5) = \{(p, r)\} \qquad \mathsf{cert}(q_4, \mathcal{K}_5) = \{(p, r), (b, r)\}$$ ▲

5.2 Conjunctive Query Answering in $\mathcal{ELHI}_\bot$

By Theorem 13, to design a procedure for answering UCQs over $\mathcal{ELHI}_\bot$ KBs, it is sufficient to focus on the problem of testing whether $\boldsymbol{a} \in \mathsf{ans}(q, \mathcal{I}_\mathcal{K})$ for a given tuple $\boldsymbol{a}$, KB $\mathcal{K}$ and CQ $q(\boldsymbol{x})$. By definition, $\boldsymbol{a} \in \mathsf{ans}(q, \mathcal{I}_\mathcal{K})$ iff $\boldsymbol{a}$ is the image of $\boldsymbol{x}$ under some match π. In relational databases, a standard way to determine the existence of such a match is take an assignment that maps $\boldsymbol{x}$ to $\boldsymbol{a}$, extend it non-deterministically by assigning an individual in the database to each existentially quantified variable, and finally to test whether the guessed assignment π is a match, that is, whether $A(\pi(x))$ and $R(\pi(x), \pi(y))$ are present in the database for every query atom $A(x)$ or $R(x, y)$.

In the presence of ontologies, the situation is more complicated. First, we have seen that an assertion may hold because it is implied by $\mathcal{K}$, without syntactically occurring in the ABox $\mathcal{A}$. Hence, in the simple algorithm outlined in the preceding paragraph, we would need to replace the syntactic containment of assertions in the database by an instance check $\mathcal{K} \models A(\pi(x))$ or $\mathcal{K} \models A(\pi(x))$, which can be carried out using the procedure described in Section 4.3. However, even with this adaptation, the resulting procedure would not be complete. A second and more challenging problem is that when guessing a variable assignment, it is not enough to map every existentially quantified variable to an individual, as we may need to consider mappings to anonymous objects in $\Delta^{\mathcal{I}_\mathcal{K}}$ in order to find the desired match. For instance, in the previous example, we saw that we needed to map $z \mapsto e_1$ to obtain $(p, r) \in \mathsf{ans}(q_3, \mathcal{I}_{\mathcal{K}_5})$ and that we have to map $z \mapsto e_3$ and $z \mapsto e_4$ to get $(b, r) \in \mathsf{ans}(q_4, \mathcal{I}_{\mathcal{K}_5})$. There can be infinitely many anonymous objects in $\mathcal{I}_\mathcal{K}$, and we do not know in advance which of them may occur in the image of the match. Hence there are infinitely many different matches that may need to be considered, and it is not apparent how to devise an effective procedure over this infinite search space.

We tackle this problem next. One possible solution would be to characterize a finite set O of anonymous objects from $\Delta^{\mathcal{I}_\mathcal{K}}$ and show that whenever there exists a match for a CQ in $\mathcal{I}_\mathcal{K}$, there exists a match that ranges over the named

individuals and O only. Some existing techniques implicitly rely on a characterization of this set O, but only for specific combinations of a query q and a TBox $\mathcal{T}$, see e.g., [80, 164, 148].

Here we take a different approach and present an algorithm [81] that rewrites the query in such a way that we do not need to consider mappings to the anonymous objects, but we can instead restrict our attention to matches to named individuals. More specifically, given a CQ $q(\boldsymbol{x})$, we construct a UCQ $\mathsf{rew}_{\mathcal{T}}(q)$ (with the same answer variables $\boldsymbol{x}$) with the property that $\boldsymbol{a} \in \mathsf{ans}(q, \mathcal{I}_{\mathcal{K}})$ iff there is a disjunct q' in $\mathsf{rew}_{\mathcal{T}}(q)$ and a match π for q' in $\mathcal{I}_{\mathcal{K}}$ such that $\pi(\boldsymbol{x}) = \boldsymbol{a}$ and the range of π contains only named individuals.

The intuition underlying the rewriting procedure is as follows. Suppose q has an existential variable x, and there is a match π for q in $\mathcal{I}_{\mathcal{K}}$ such that $\pi(x)$ is an anonymous object in the tree part of $\mathcal{I}_{\mathcal{K}}$, and it has no descendant in the image of π. Then for all atoms $R(y, x)$ or $R(x, y)$ of q, the 'neighbor' variable y must be mapped to the parent p of $\pi(x)$ in $\mathcal{I}_{\mathcal{K}}$. A rewriting step chooses such a variable x, together with an existential axiom $M \sqsubseteq \exists S.N$ from $\mathsf{saturate}(\mathcal{T})$ such that all atoms of q involving x are satisfied provided the parent p is an instance of M. Then the algorithm can 'clip off' x, eliminating all query atoms involving it, and adding instead fresh atoms to ensure that the parent p satisfies M. The resulting query q' has a match π' that is similar to π, but crucially, the length of the longest path occurring in the image of π' is strictly shorter than for π. By repeating this procedure, we can clip off all variables matched in the tree part to obtain shorter and shorter matches, until we end up with a set of queries such that, if they have a match in $\mathcal{I}_{\mathcal{K}}$, then they have a match whose range contains only ABox individuals.

Definition 8. *For a CQ q and a $\mathcal{ELHI}_{\bot}$ TBox $\mathcal{T}$, we write $q \to_{\mathcal{T}} q'$ if q' can be obtained from q by applying the following steps:*

(S1) Select in q an arbitrary existentially quantified variable x such that there are no atoms of the form $R(x, x)$ in q.

(S2) Replace each role atom of the form $R(x, y)$ in q, where y and R are arbitrary, by the atom $\mathsf{inv}(R)(y, x)$.

(S3) Let $V_p = \{y \mid R(y, x) \in q \text{ for some } R\}$, and select some $M \sqsubseteq \exists S.N \in \mathsf{saturate}(\mathcal{T})$ such that
   (a) $S \sqsubseteq R \in \mathsf{saturate}(\mathcal{T})$ for every $R(y, x) \in q$, and
   (b) $\{A \mid A(x) \in q\} \subseteq N$.

(S4) Drop from q every atom that contains x.

(S5) Select a variable $y \in V_p$ and replace every occurrence of $y' \in V_p$ in q by y.

(S6) Add the atoms $\{A(y) \mid A \in M\}$ to q.

We write $q \to_{\mathcal{T}}^{*} q'$ if $q = q_0$ and $q' = q_n$ for some finite rewrite sequence $q_0 \to_{\mathcal{T}} q_1 \cdots \to_{\mathcal{T}} q_n$, $n \geq 0$. Furthermore, we let $\mathsf{rew}_{\mathcal{T}}(q) = \{q' \mid q \to_{\mathcal{T}}^{*} q'\}$.

In (S1) we guess an existentially quantified variable x (we exclude variables appearing in self-loops as such variables cannot be mapped to anonymous objects). For convenience, in (S2), we invert all atoms of the form $R(x, y)$, so that

x always appears in the second position of role atoms. In (S3), we let V_p be the set of all 'neighbor' variables y of x for which there is an atom $R(y, x)$ in q; intuitively, every such variable y must be mapped to the parent p of $\pi(x)$. We also select a TBox inclusion that ensures the existence of a suitable child $\pi(x)$, under the assumption that the left-hand side of the inclusion is satisfied at p. Then we can clip off x in (S4), merge all variables of V_p in (S5), and add to q new atoms that enforce satisfaction of the concepts appearing on the left-hand side of the selected axiom in (S6).

Example 22. Consider the second disjunct of q_4, that is, the following CQ:

$$q_5(y, x) = \exists z, z'.\mathsf{serves}(x, y) \wedge \mathsf{hasIngredient}(y, z) \wedge \mathsf{hasIngredient}(z, z') \wedge \mathsf{Spicy}(z')$$

Recall that $(b, r) \in \mathsf{ans}(q_5, \mathcal{I}_\mathcal{K})$, as witnessed by the match $\pi(x) = r$, $\pi(y) = b$, $\pi(z) = e_3$, and $\pi(z') = e_4$. In our running example, a possible rewriting step for q_5 could select in (S1) the variable z' (which intuitively means that we guess that $\pi(z')$ is a leaf in the image of q_5 under some match, as is the case for the match π). Then there is nothing to do in (S2), and in (S3) we see that $V_p = \{z\}$ is the only variable that has to be mapped to the parent of $\pi(z')$. We need to select some axiom $M \sqsubseteq \exists S.N \in \mathsf{saturate}(\mathcal{T}_5)$ that ensures the satisfaction of all atoms involving z', that is, of $\mathsf{hasIngredient}(z, z')$ and $\mathsf{Spicy}(z')$. We see that $\mathsf{ArrabiataSauce} \sqsubseteq \exists \mathsf{hasIngredient}.\mathsf{Spicy}$ is such an axiom, so in (S4) and (S6), we drop the atoms $\mathsf{hasIngredient}(z, z')$ and $\mathsf{Spicy}(z')$ and replace them by $\mathsf{ArrabiataSauce}(z)$ (we may skip (S5) since $|V_p| = 1$). Summing up, after one rewriting step we get:

$$q_5'(y, x) = \exists z.\mathsf{serves}(x, y) \wedge \mathsf{hasIngredient}(y, z) \wedge \mathsf{ArrabiataSauce}(z)$$

Note that every match of q_5' can be extended to a match of q_5, hence the rewriting procedure does not introduce any incorrect answers. The motivation for introducing q_5' is that the image of a match of q_5' goes one step less deep into the anonymous part of the canonical model than the corresponding match for q_5. In a second rewriting step, we again choose to eliminate z, and we get that $V_p = \{y\}$. We select in (S3) the axiom $\mathsf{PenneArrabiata} \sqsubseteq \exists \mathsf{hasIngredient}.\mathsf{ArrabiataSauce}$, since mapping y to an instance of $\mathsf{PenneArrabiata}$ suffices to make the atoms that involve z (namely, $\mathsf{hasIngredient}(y, z)$ and $\mathsf{ArrabiataSauce}(z)$) true. We can then replace these atoms by $\mathsf{PenneArrabiata}(y)$ to obtain

$$q_5''(y, x) = \mathsf{serves}(x, y) \wedge \mathsf{PenneArrabiata}(y)$$

Now we have obtained a query q_5'' that has a match ($\pi(x) = r$, $\pi(y) = b$) where all variables are mapped to individuals. Moreover, by virtue of the rewriting process, we know that the match for q_5'' implies the existence of a corresponding match for q_5, and so we have $(b, r) \in \mathsf{ans}(q_5, \mathcal{I}_{\mathcal{K}_5})$. ▲

The rewriting procedure that we have just presented is a slightly simplified version of the one defined in [81] for Horn-$\mathcal{SHIQ}$, and the results in that paper imply the following theorem.

Theorem 14. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a satisfiable $\mathcal{ELHI}_\bot$ knowledge base, and let $q(\boldsymbol{x})$ be a CQ. Then $\boldsymbol{a} \in \mathsf{cert}(q, \mathcal{K})$ iff there is some $q' \in \mathsf{rew}_\mathcal{T}(q)$ and an assignment π from the variables in q' to $\mathsf{Ind}(\mathcal{A})$ such that $\pi(\boldsymbol{x}) = \boldsymbol{a}$ and π is a match for q in $\mathcal{I}_\mathcal{K}$.*

Importantly, since $\mathsf{Ind}(\mathcal{A})$ is bounded, for each $q' \in \mathsf{rew}_\mathcal{T}(q)$, there are only a bounded number of candidate assignments π (in fact, single-exponentially many). We need to check whether one of these candidate matches π is indeed a match, that is, all of its atoms are satisfied under π. We can test each candidate assignment in turn, using the instance checking algorithms from Section 4.3 to decide whether a given query atom is satisfied under a given assignment. By Theorem 14, the approach we have just described yields a terminating, sound, and complete decision procedure for CQ answering in $\mathcal{ELHI}_\bot$.

The procedure also has an optimal worst-case combined and data complexity. For combined complexity, we know that computing $\mathsf{saturate}(\mathcal{T})$ is feasible in single exponential time. The cardinality of the set of rewritten queries $\mathsf{rew}_\mathcal{T}(q)$ is single exponential in $|\mathcal{T}|$ and $|q|$, since it only contains queries whose variables are a subset of the variables in q, and concept and role names that appear in $\mathcal{T}$. Moreover, the set $\mathsf{rew}_\mathcal{T}(q)$ can be computed in single exponential time. Once $\mathsf{rew}_\mathcal{T}(q)$ has been computed, we can consider the single-exponentially many candidate assignments of the variables for each $q' \in \mathsf{rew}_\mathcal{T}(q)$, and for each such assignment, we need to do a number of instance checks that is linear in the size of q'; by Theorem 12, instance checking can be performed in single exponential time. Since already consistency and instance checking in $\mathcal{ELI}$ are hard for single exponential time [19], the resulting EXP bound is optimal.

As for data complexity, we note that $\mathsf{rew}_\mathcal{T}(q)$ can be computed in constant time for a fixed $\mathcal{T}$ and q, and the number of candidate assignments π is polynomial in the size of the ABox. Since instance checking is in P regarding data complexity, we obtain:

Theorem 15. *CQ answering in $\mathcal{ELHI}_\bot$ and Horn-$\mathcal{SHIQ}$ is* EXP*-complete in combined complexity and* P*-complete in data complexity.*

We remark that the same bounds hold for Horn-$\mathcal{SHIQ}$, which is in fact the logic for which this rewriting procedure was developed.

With minor adaptations, we can use the same technique to obtain optimal bounds for $\mathcal{ELH}$ and DL-Lite$_\mathcal{R}$. For combined complexity, we can devise an algorithm that runs in non-deterministic polynomial time. First, we compute $\mathsf{saturate}(\mathcal{T})$ in polynomial time. Then we can non-deterministically guess and build the right $q' \in \mathsf{rew}_\mathcal{T}(q)$, guess a candidate assignment π, and check in polynomial time if it is a match. Indeed, to determine whether an assignment is a match, we must perform a polynomial number of instance checks, and each check can be done in polynomial time for $\mathcal{ELH}$ and DL-Lite$_\mathcal{R}$. This NP bound is optimal, since CQ answering is already NP-hard over an ABox alone, seen as a database, and with no TBox. For data complexity, instance checking in $\mathcal{EL}$ is hard for P, and we can easily obtain a matching upper bound for $\mathcal{ELH}$: the

set $\mathsf{rew}_{\mathcal{T}}(q)$ can be obtained in polynomial time, there are only polynomially candidate assignments for the rewritten queries, and testing for a match only needs a polynomial number of polynomial-time instance checks. In DL-Lite$_{\mathcal{R}}$, we will see that our techniques can be used to obtain an FO rewriting, yielding membership in AC_0. These bounds are summarized in the next theorem.

Theorem 16. *CQ answering in $\mathcal{ELH}$ and DL-Lite$_{\mathcal{R}}$ is* NP*-complete in combined complexity. For $\mathcal{ELH}$ the data complexity is* P*-complete, and for DL-Lite$_{\mathcal{R}}$ the data complexity is in AC_0.*

We note that the results in this theorem are anterior to [81]. The upper bounds for DL-Lite$_{\mathcal{R}}$ follow from the seminal papers on the DL-Lite family and the original PerfectRef query rewriting algorithm [50, 52], and the upper bounds for CQ answering in $\mathcal{EL}$ were first established in [186, 134, 133]. In Section 8, we will give a brief overview of subsequent work aimed at developing, optimizing, and implementing efficient CQ answering algorithms for these and related logics.

We close this subsection by considering different possible ways of translating Theorem 14 into a concrete query answering algorithm.

A Datalog rewriting approach for CQs in $\mathcal{ELHI}_{\bot}$ A first option is to define a Datalog rewriting. Indeed, since $\mathsf{rew}_{\mathcal{T}}(q)$ is a UCQ, it can be viewed as a set Π_{rew} of Datalog rules that all use the same head predicate Q. By Theorem 14, (Π_{rew}, Q) will give the correct answer to q if evaluated over the enriched ABox consisting of all assertions that are entailed from the KB. To obtain this completed ABox, we may exploit the Datalog program $\Pi(\mathcal{T}, \Sigma)$ from Section 4.3, which has the property that for every Σ-ABox $\mathcal{A}$ and every concept or role assertion $P(\boldsymbol{t})$, $\mathcal{T}, \mathcal{A} \models P(\boldsymbol{t})$ iff $\boldsymbol{t} \in \mathsf{ans}(\mathcal{I}_{\mathcal{A}}, (\Pi(\mathcal{T}, \Sigma), P))$. It follows that the query $(\Pi(\mathcal{T}, \Sigma) \cup \Pi_{\mathsf{rew}}, Q)$ is a Datalog rewriting of q w.r.t. $\mathcal{T}, \Sigma$ relative to consistent ABoxes.

A combined approach for CQs in $\mathcal{ELHI}_{\bot}$ Another possibility is to use Theorem 14 as the basis of a *combined* approach in the spirit of [152] that uses both saturation and rewriting. As $\mathsf{saturate}(\mathcal{K})$ contains all the assertions entailed by $\mathcal{T}$ and $\mathcal{A}$, it suffices to pose the UCQ $\mathsf{rew}_{\mathcal{T}}(q)$ over (the interpretation corresponding to) the assertions in $\mathsf{saturate}(\mathcal{K})$ viewed as a database. Compared to the pure Datalog rewriting approach, the combined approach has the advantage that we can use standard relational database systems, which are more mature than Datalog engines. Its main drawback is that the saturated version of the ABox needs to be recomputed whenever the KB is modified (see Section 8 for further discussion of combined approaches to OMQA).

An FO rewriting approach for CQs in DL-Lite$_{\mathcal{R}}$ We can also use the rewritten set of queries as the basis of an FO rewriting approach for DL-Lite$_{\mathcal{R}}$. We know that to determine whether $\boldsymbol{a} \in \mathsf{cert}(q, \mathcal{K})$, we only need to decide

whether there is some some $q' \in \mathsf{rew}_{\mathcal{T}}(q)$ and an assignment π from the variables in q' to $\mathsf{Ind}(\mathcal{A})$ such that $\mathcal{T}, \mathcal{A} \models P(\pi(\boldsymbol{t}))$ for each atom $P(\boldsymbol{t})$ in q. We can exploit our rewriting algorithm for instance checking in DL-Lite$_{\mathcal{R}}$ (Section 4.1), and replace in the queries $q' \in \mathsf{rew}_{\mathcal{T}}(q)$ each atom $A(t)$ by $\mathsf{RewriteIQ}(A, \mathcal{T})$ and each atom $r(t, t')$ by $\mathsf{RewriteIQ}(r, \mathcal{T})$. The result of this replacement is a (positive) FO query[13] q_{rew} that is a rewriting of q w.r.t. $\mathcal{T}$ relative to consistent ABoxes. As discussed in Sections 3 and 4, the latter rewriting can be combined with an FO rewriting of unsatisfiability to obtain an FO rewriting of q w.r.t. $\mathcal{T}$ that works for all ABoxes over the given signature. Since the data complexity of answering FO queries over relational databases is in AC_0, this yields the remaining upper bound in Theorem 16.

5.3 Related results and discussion

We close this section with a short discussion of other results that are related to answering CQs and other positive fragments of FO queries in the presence of (both Horn and non-Horn) DL ontologies.

Other results on CQ answering in Horn DLs For members of the $\mathcal{EL}$ family, including $\mathcal{EL}^{++}$ and some other fragments not contained in $\mathcal{ELHI}_{\bot}$, the first complexity results for CQ answering were established in [186, 133, 134]. An important result common to the three works was that CQ answering is undecidable for $\mathcal{EL}^{++}$. Both [186] and [134] present fragments that are NP-complete for combined complexity, while [133] focused on data complexity. A PSPACE upper bound for a fragment of regular $\mathcal{EL}^{++}$ was obtained in [135]. More recently, a tight NP upper bound was shown for the fragment of regular $\mathcal{ELRO}^{+}$ that restricts complex role assumptions to transitivity axioms [202], and a tight PSPACE upper bound was obtained for (full) regular $\mathcal{ELRO}^{+}$ that corresponds to OWL 2 EL [203, 201].

For more expressive Horn DLs, like Horn-$\mathcal{SHIQ}$, there are fewer results. The query rewriting technique we have discussed in this section was proposed in [81] for Horn-$\mathcal{SHIQ}$, and it shares core ideas with previous algorithms for $\mathcal{EL}$ [186] and DL-Lite [188]. The complexity bounds for Horn-$\mathcal{SHIQ}$ had been obtained already in [78], but with a different algorithm less suited for implementation. Recent work has considered the more expressive Horn logics Horn-$\mathcal{SHOIQ}$ and Horn-$\mathcal{SROIQ}$ and generalizations of CQs with a limited form of recursion [166]; we will discuss this in Section 6.4.

Unions of Conjunctive Queries and Positive Existential Queries In this section, we have mainly focused on CQs, but by Theorem 13, we know that the obtained results extend to UCQs. In fact, the universal model property also applies to *positive existential queries (PEQs)*, a class of FO queries that generalize CQs by allowing arbitrary combinations of conjunctions and disjunctions of

[13] If desired, we could use standard equivalence-preserving transformations to turn q_{rew} into an equivalent UCQ.

atoms. Since every positive FO formula can be put into disjunctive normal form (i.e., rewritten as a disjunction of conjunctions), PEQs have the same expressive power as UCQs, although they can be exponentially more succinct. The complexity results for (U)CQ answering in Horn DLs can be easily transferred to PEQs. The core idea is that to when checking whether a candidate assignment is a match for the query, one also considers a choice of a subset of the query atoms whose satisfaction leads to the PEQ being satisfied. If we consider data complexity, enumerating all possible choices of subsets of query atoms requires only constant time. If we consider combined complexity, then CQ answering is already NP-hard, so an additional step that non-deterministically guesses a suitable 'good' subset of query atoms causes no further increase in complexity.

Results for Non-Horn DLs Recall that for DLs that are capable of expressing disjunction, a universal model does not exist in general, so to decide whether a given tuple is an answer to a UCQ $\bigvee_i q_i$, we need to verify that in every model $\mathcal{I}$ of the KB $\mathcal{K}$, we have $\boldsymbol{a} \in \mathsf{ans}(q_i, \mathcal{I})$ for some q_i. The loss of a universal model has a major impact on the complexity of query answering. Data complexity becomes coNP-complete-hard [193, 164], and for combined complexity, query answering typically becomes harder by one exponential. Over the last decade, 2Exp upper bounds for answering CQs or extensions thereof have been obtained for many expressive DLs for which satisfiability and entailment are Exp-complete, such as $\mathcal{SHIQ}$ [89, 56], $\mathcal{SHOQ}$ [90], $\mathcal{ZIQ}$, $\mathcal{ZOQ}$, and $\mathcal{ZOI}$ [57]. These bounds apply for PEQs as well, and they are tight for CQs in every DL that contains $\mathcal{ALCI}$ [147] or $\mathcal{SH}$ [80]. For all DLs between $\mathcal{ALC}$ and $\mathcal{ALCHQ}$, the succinctness gap between PEQs and UCQs makes a difference: UCQs can be answered in single-exponential time [167, 147], but PEQs need double-exponential time in the worst case [169].

The results above have been obtained using a variety of techniques, such as automata [58, 57], resolution [158, 176], or modified tableaux [144, 164]. In most other cases, query answering algorithms can be viewed as comprising two main steps. In the first step, partial assignments from the query variables to the individuals occurring in the ABox are used to generate an exponential number of new query answering problems that can be answered over restricted tree-like interpretations. In a second stage, queries over tree-shaped interpretations are answered using techniques like *rolling-up* [54, 110, 89] that encodes the queries into concepts, or *knot* [82, 79] and *domino* [78] techniques that break all possible interpretations into small structures.

The loss of the universal model property is particularly problematic for DLs that simultaneously support inverse roles, nominals, and number restrictions, like $\mathcal{ALCOIQ}$ and its extensions, since these logics also lack the forest-like models that other logics enjoy. This makes the query answering problem so challenging that it has still not been successfully solved. Answering CQs is known to be hard for N2Exp for $\mathcal{ALCOIQ}$ (in fact, for the slightly weaker $\mathcal{ALCOIF}$) [91], and decidable for $\mathcal{ALCHOIQ}$ [191], but no elementary upper bounds on its complexity have been established. For $\mathcal{SHOIQ}$, decidability of CQs remains

open, although UCQs are known to be undecidable in the closely related logic that extends $\mathcal{ALCOIQ}$ with a transitive closure operator on roles [165].

6 Navigational Queries

The last decade has seen a huge growth of applications that store and query data that has a relatively simple structure, but that is highly connected and does not comply to a fixed, rigid relational schema. This includes, for example, applications in which the data stems from the so-called web of linked data, or from social, biological, and chemical networks. While CQs and UCQs are the predominant query languages for relational databases, they are widely considered to be insufficient for this kind of applications, since they cannot express even basic reachability queries or retrieve pairs of objects that are connected by a path with certain features. Instead, for querying this kind of data, one is interested in so-called 'navigational' query languages.

The most basic navigational query language is *regular path queries (RPQs)*, which allow one to find all pairs of objects that are connected by a chain of roles (binary relations, in the database setting) that comply with a given regular language. In two-way RPQs (2RPQs), the vocabulary of the regular language comprises both roles and their inverses. We note that 2RPQs lie at the heart of XPath [69], which is the standard query language for querying XML documents, and are also present in SPARQL 1.1 [106], where they go by the name of *property paths*. By combining (2)RPQs and CQs, we obtain *conjunctive (2)RPQs*, allowing one to search for patterns that conjunctively combine regular paths.

Within the database community, there have been considerable research efforts devoted to studying the properties of these navigational query languages, developing query answering algorithms for them, and extending these languages with yet more features to meet the needs of applications. In the past few years, the DL research community has also begun to explore the use of navigational query languages for OMQA. This chapter provides an overview of this recent and ongoing line of research.

6.1 Regular path queries and their extensions

We start by formally defining the language of C2RPQs. They are syntactically very similar to CQs, but the atoms of the form $R(t, t')$ are generalized to $L(t, t')$, where L is a regular language over the alphabet of role names and their inverses. Intuitively, a pair of objects satisfies such an atom if they are connected via a chain of roles whose label belongs to L. The language L can be represented either by regular expressions or non-deterministic finite state automata (NFAs); the latter representation is known to be exponentially more succinct [77]. We note that the complexity results we mention in this chapter were shown for regular expressions in the case of lower bounds, and NFA for upper bounds, hence they all hold independently of the representation.

Definition 9. *Recall that* $\mathsf{N}_\mathsf{R}^\pm$ *contains all role names and their inverses. A* conjunctive two-way regular path query (C2RPQ) *has the form* $q(\boldsymbol{x}) = \exists \boldsymbol{y}.\varphi$ *where* $\boldsymbol{x}$ *and* $\boldsymbol{y}$ *are tuples of variables, and* φ *is a conjunction of atoms of the forms:*

(i) $A(t)$*, where* $A \in \mathsf{N}_\mathsf{C}$ *and* $t \in \mathsf{N}_\mathsf{I} \cup \boldsymbol{x} \cup \boldsymbol{y}$*, and*
(ii) $L(t, t')$*, where* L *is (an NFA or regular expression defining) a regular language over* $\mathsf{N}_\mathsf{R}^\pm \cup \{A? \mid A \in \mathsf{N}_\mathsf{C}\}$*, and* $t, t' \in \mathsf{N}_\mathsf{I} \cup \boldsymbol{x} \cup \boldsymbol{y}$*.*

Conjunctive (one-way) regular path queries (CRPQs) *are obtained by disallowing symbols from* $\mathsf{N}_\mathsf{R}^\pm \setminus \mathsf{N}_\mathsf{R}$ *in atoms of type (ii).* Two-way regular path queries (2RPQs) *consist of a single atom of type (ii) such that* t *and* t' *are both answer variables.* Regular path queries (RPQs) *are 2RPQs that do not use any roles from* $\mathsf{N}_\mathsf{R}^\pm \setminus \mathsf{N}_\mathsf{R}$*.*

To define the semantics of C2RPQs, we proceed as for CQs by defining a notion of match. As a first step, we must specify how the atoms of the form $L(t, t')$ should be interpreted.

A *path* from e_0 to e_n in interpretation $\mathcal{I}$ is a sequence $e_0 u_1 e_1 u_2 \dots u_n e_n$ with $n \geq 0$ such that every e_i is an element from $\Delta^\mathcal{I}$, every u_i is a symbol from $\mathsf{N}_\mathsf{R}^\pm \cup \{A? \mid A \in \mathsf{N}_\mathsf{C}\}$, and for every $1 \leq i \leq n$:

- If $u_i = A?$, then $e_{i-1} = e_i \in A^\mathcal{I}$;
- If $u_i = R \in \mathsf{N}_\mathsf{R}^\pm$, then $(e_{i-1}, e_i) \in R^\mathcal{I}$.

The *label* $\lambda(p)$ of path $p = e_0 u_1 e_1 u_2 \dots u_n e_n$ is the word $u_1 u_2 \dots u_n$. Note that if $p = e_0$, then we define $\lambda(p)$ to be ε. For every language L over $\mathsf{N}_\mathsf{R}^\pm \cup \{A? \mid A \in \mathsf{N}_\mathsf{C}\}$, the semantics of L w.r.t. interpretation $\mathcal{I}$ is defined as follows:

$$L^\mathcal{I} = \{(e_0, e_n) \mid \text{there is some path } p \text{ from } e_0 \text{ to } e_n \text{ with } \lambda(p) \in L\}$$

A *match* for a C2RPQ q in an interpretation $\mathcal{I}$ is a mapping π from the terms in q to elements in $\Delta^\mathcal{I}$ such that

- $\pi(c) = c^\mathcal{I}$ for each $c \in \mathsf{N}_\mathsf{I}$,
- $\pi(t) \in A^\mathcal{I}$ for each atom $A(t)$ in q, and
- $(\pi(t), \pi(t')) \in L^\mathcal{I}$ for each $L(t, t')$ in q.

Finally, using this notion of match, we can define what it means for a tuple to be an *answer* to a C2RPQ $q(\boldsymbol{x})$ in interpretation $\mathcal{I}$:

$$\mathsf{ans}(q, \mathcal{I}) = \{\boldsymbol{e} \mid \boldsymbol{e} = \pi(\boldsymbol{x}) \text{ for some match } \pi \text{ for } q \text{ in } \mathcal{I}\}$$

We will again be interested in the certain answers, that is, the tuples of individuals $\boldsymbol{a}$ for which $\boldsymbol{a}^\mathcal{I} \in \mathsf{ans}(q, \mathcal{I})$ for every model $\mathcal{I}$ of the KB. Importantly, C2RPQs share with CQs the property of being preserved under homomorphisms, the certain answers to a C2RPQ q over an $\mathcal{ELHI}_\bot$ KB $\mathcal{K}$ coincide with the answers to q in the canonical model $\mathcal{I}_\mathcal{K}$ of $\mathcal{K}$.

Example 23. The following CRPQ is similar to the CQ q_3 and the UCQ q_4 in Example 19: it retrieves dishes y that contain a spicy ingredient, together with the location x where they are served. However, unlike q_3 and q_4, this query can find the spicy component no matter how many $\mathsf{hasIngredient}$ steps away it is.

$$q_6(y, x) = \exists z.\mathsf{serves}(x, y) \wedge \mathsf{hasIngredient}^*\mathsf{Spicy}?(y, z)$$

The following one-atom CRPQ can express the infinite FO query of Example 9:

$$q_7(x) = \exists y.\mathsf{hasIngredient}^*\mathsf{Spicy}?(x, y)$$

In fact, it is also possible to compute the answers to this query using a (non-conjunctive) 2RPQ. Since 2RPQs do not support existential variables, we cannot use $\mathsf{hasIngredient}^*\mathsf{Spicy}?(x, y)$ since it will only allow us to retrieve the values of x whose spicy component y happens to be an ABox individual, which need not be the case in general. To make sure that all x with an spicy component are found, we can use the following slightly modified query q_8:

$$q_8(x, y) = \mathsf{hasIngredient}^*\mathsf{Spicy}?\Sigma^*(x, y)$$

where $\Sigma = \mathsf{N}_\mathsf{R}^\pm \cap \mathsf{sig}(\mathcal{K})$. Observe that the language Σ^* allows us to reach some ABox individual starting from any element in $\mathcal{I}_\mathcal{K}$. It follows that whenever there is a match π for q_7, possibly mapping y to an anonymous object, there will be a match π' for q_8 such that $\pi(x) = \pi'(x)$ and $\pi'(x) \in \mathsf{Ind}(\mathcal{A})$. Thus, the query q_7 is equivalent to the projection of q_8 onto its first component, or more formally, $\mathsf{cert}(q_7, \mathcal{K}) = \{a \mid (a, b) \in \mathsf{cert}(q_8, \mathcal{K}) \text{ for some } b\}$, for every knowledge base $\mathcal{K}$. ▲

We note that C(2)RPQs are strictly more expressive than CQs. Indeed, every CQ is is also a C2RPQ, but there exist RPQs (like $\mathsf{Spicy}^*(x, y)$) that cannot be expressed as a CQ (nor as an FO query). The language of Datalog queries is in turn strictly more expressive than C2RPQs. It is not hard to show that every C2RPQ can equivalently expressed as a Datalog query. In fact, C2RPQs fall inside the *linear* fragment of Datalog that restricts the use of recursion by allowing at most one recursive predicate in each rule body[14].

C2RPQs are in general better behaved computationally than full Datalog. Take for example the fundamental analysis task of query containment, which consists in deciding whether the answer to one query is always contained in the answer to another query over every possible database. Query containment of Datalog queries is known to be undecidable [198], while this problem has been shown decidable even for extensions of C2RPQs [55, 84, 180, 44] and in the presence of constraints [99, 61]. Moreover, as we shall see in Section 7, Datalog query answering over DL knowledge bases is undecidable even for very simple DLs [144]. In contrast, C2RPQs are decidable even for very expressive DLs [57].

[14] A predicate is called *recursive* if it occurs in a cycle in the dependency graph of the Datalog program, whose nodes are the program's predicates and which contains an edge between two predicates whenever there is a rule that contains one of the predicates in the body and the other in the head.

The increased expressiveness of C2RPQs, and in particular their ability to express simple recursive queries like reachability, together with their good computational properties, make them a very appealing query language for OMQA. They are especially relevant when querying KBs formulated in lightweight DLs, since the query language can compensate for limited expressivity of the DL (e.g., inability to propagate information over roles in DL-Lite$_R$, lack of inverses in $\mathcal{EL}$).

Example 24. The query $q_7 = \exists y.\mathsf{Spicy}^*(x, y)$ can be seen as a C2RPQ rewriting of the query $\mathsf{Spicy}(x)$ w.r.t. the TBox $\mathcal{T} = \{\exists \mathsf{hasIngredient}.\mathsf{Spicy} \sqsubseteq \mathsf{Spicy}\}$. There is no DL-Lite$_\mathcal{R}$ TBox equivalent to this TBox, and the desired meaning of $\mathsf{Spicy}(x)$ is not captured by any FO query over DL-Lite. ▲

6.2 Answering 2RPQs

In this section, we present an algorithm for answering 2RPQs in $\mathcal{ELHI}_\bot$. In 2RPQs, there are no existentially qualified variables, hence it is enough to consider matches to the named individuals. A straightforward algorithm for evaluating a 2RPQ $L(x, y)$ would first guess a pair of individuals (a, b), and then check whether there is a path between a and b whose label complies with L. However, checking the existence of such a path is still challenging: although this chain starts and ends in the 'core' of the canonical model, it need not be fully contained in it. Indeed, a path between two individuals in $\mathcal{I}_\mathcal{K}$ may still need to go (possibly quite deep) into the anonymous part and come back out in order to satisfy the regular expressions in the query, as we illustrate next.

Example 25. Reconsider KB $\mathcal{K}_5$ from Example 20. The mapping $\pi(x) = \pi(y) = p$ is a match for q_8, hence $(p, p) \in \mathsf{ans}(q_8, \mathcal{I}_{\mathcal{K}_5})$. However, the fact that there is a $\mathsf{hasIngredient}^*\mathsf{Spicy}?\Sigma^*$-labelled path from p to p is only witnessed by the path of the form

$$p\ \mathsf{hasIngredient}\ e_1\ \mathsf{Spicy}?\ e_1\ \mathsf{hasIngredient}^-\ p$$

or by longer paths that have this one as a suffix. Similarly, all paths witnessing that $\pi(x) = \pi(y) = b$ is a match for q_8 (and hence $(b, b) \in \mathsf{ans}(q_8, \mathcal{I}_{\mathcal{K}_5})$) need to go two steps into the anonymous part and pass by e_4. That is, we need to navigate the path

$$p\ \mathsf{hasIngredient}\ e_3\ \mathsf{hasIngredient}\ e_4\ \mathsf{Spicy}?\ e_4\ \mathsf{hasIngredient}^-e_3\ \mathsf{hasIngredient}^-\ p$$

in order to satisfy the regular expression. ▲

To describe how to address this problem, let us first suppose that the regular language in the 2RPQ is given by an NFA α (as mentioned earlier, this is without loss of generality, since regular expressions are easily translated into NFAs). Our strategy for deciding the existence of a suitable path is to define a relation Loop_α that stores all possible 'loops' through the anonymous part of $\mathcal{I}_\mathcal{K}$ that can be used to partially satisfy α. That is, we store every path that starts and ends at a given individual a and takes the query automaton α from state s to state s'.

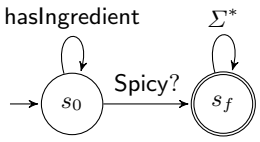

Fig. 4: NFA α_8 for the regular expression $\mathsf{hasIngredient}^*\mathsf{Spicy}?\Sigma^*$ from query q_8.

Intuitively, if such a loop exists at the individual a, then we may 'jump' directly from (a, s) to (a, s') when looking for a path between two individuals, and in this way we can avoid navigating the anonymous part in our algorithm.

An important observation that we can draw from the construction of the canonical model is that the conjunction of concept names that an object satisfies uniquely determines everything that occurs 'below it' in the anonymous part. Hence, the relevant loops from a state s to a state s' can be characterized in terms of the conjunctions of concepts that enforce them. We thus define Loop_α as a relation that associates with each pair of states s, s' from α a set of conjunctions M of concept names, in such a way that the following holds:

(†) $M \in \mathsf{Loop}_\alpha[s, s']$ iff for every individual a, we have that $a \in M^{\mathcal{I}_\mathcal{K}}$ implies that there exists a path $p = e_0 u_1 e_1 u_2 \dots u_n e_n$ in $\mathcal{I}_\mathcal{K}$ such that $e_0 = e_n = a$, e_i is of the form $a \cdot w$ for $0 < i < n$, and $\lambda(p) \in L(\alpha_{s,s'})$, where $\alpha_{s,s'}$ obtained from α by making s the starting state and s' the unique final state.

Example 26. In Figure 4, we display an NFA α_8 representing the (regular language corresponding to the) regular expression $\mathsf{hasIngredient}^*\mathsf{Spicy}?\Sigma^*$ from the query q_8. Apart from the trivial loops (that is, paths of length 0 from an individual and state to itself), we have that:

- $\mathsf{PizzaCalabrese} \in \mathsf{Loop}_{\alpha_8}(s_0, s_f)$, since from any object e that belongs to $\mathsf{PizzaCalabrese}$ we can walk one $\mathsf{hasIngredient}$ step to an instance e' of Nduja, which is also Spicy (such an object e' exists in the canonical model due to inclusion $\mathsf{PizzaCalabrese} \sqsubseteq \exists \mathsf{hasIngredient}.(\mathsf{Nduja} \sqcap \mathsf{Spicy})$ in $\mathsf{saturate}(\mathcal{T}_5)$). Since the object e' satisfies Spicy, we can move to s_f while staying at e', and then we can take a $\mathsf{hasIngredient}^-$ step back to the original e, while staying in the final state s_f.
- $\mathsf{PenneArrabiata} \in \mathsf{Loop}_{\alpha_8}(s_0, s_f)$, since from any object e that is of type $\mathsf{PenneArrabiata}$, we can walk one $\mathsf{hasIngredient}$ step to an instance e' of $\mathsf{ArrabiataSauce}$, and then take a second $\mathsf{hasIngredient}$ step to an instance e'' of $\mathsf{Peperoncino}$, which is also Spicy (again, the existence of e' and e'' is ensured by corresponding axioms in $\mathsf{saturate}(\mathcal{T}_5)$). Since e'' satisfies Spicy, we can move to s_f while staying at e', and then we can take a $\mathsf{hasIngredient}^-$ step back to e', and another one to e, while remaining in final state s_f. ▲

A possible way to test whether $M \in \mathsf{Loop}_\alpha[s, s']$ is to explicitly compute the full table Loop_α. This can be done inductively using the following rules[15], obtained by adapting existing constructions[16] for $\mathcal{ELH}$ and DL-Lite$_\mathcal{R}$ to $\mathcal{ELHI}_\bot$.

(L1) For every $s \in S$: $\mathsf{Loop}_\alpha[s, s] = \mathsf{N_C}$.

(L2) If $M_1 \in \mathsf{Loop}_\alpha[s_1, s_2]$ and $M_2 \in \mathsf{Loop}_\alpha[s_2, s_3]$, then $M_1 \sqcap M_2 \in \mathsf{Loop}_\alpha[s_1, s_3]$.

(L3) If $\{C_1, \ldots, C_n\} \subseteq \mathsf{N_C}$, $\mathcal{T} \models C_1 \sqcap \cdots \sqcap C_n \sqsubseteq A$, and $(s_1, A?, s_2) \in \delta$, then $C_1 \sqcap \cdots \sqcap C_n \in \mathsf{Loop}_\alpha[s_1, s_2]$.

(L4) If $\{C_1, \ldots, C_n\} \subseteq \mathsf{N_C}$, $\mathcal{T} \models C_1 \sqcap \cdots \sqcap C_n \sqsubseteq \exists R.D$, $\mathcal{T} \models R \sqsubseteq R'$, $\mathcal{T} \models R \sqsubseteq R''$, $(s_1, R', s_2) \in \delta$, $D \in \mathsf{Loop}_\alpha[s_2, s_3]$, and $(s_3, R''^-, s_4) \in \delta$, then $C_1 \sqcap \cdots \sqcap C_n \in \mathsf{Loop}_\alpha[s_1, s_4]$.

The resulting table is exactly the desired relation described by (†), see [32] for a formal proof of the analogous results for $\mathcal{ELH}$ and DL-Lite$_\mathcal{R}$. Instead of building the full table, another possibility is to add a set of axioms that reduce testing $M \in \mathsf{Loop}_\alpha[s, s']$ to an entailment test in $\mathcal{ELHI}_\bot$. The latter alternative has been used for $\mathcal{ELHI}_\bot$ [27], but for a more involved notion of loop designed for an extension of C2RPQs.

Now that we have a means of determining which loops through the anonymous part are available from a given ABox individual, we are ready to present the evaluation algorithm EvalAtom in Figure 5. It takes as input an NFA $\alpha = (S, \Sigma, \delta, s_0, F)$, an $\mathcal{ELHI}_\bot$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, and a pair of individuals (a, b) from $\mathcal{A}$, and it decides whether $(a, b) \in \mathsf{cert}(\alpha(x, y), \mathcal{K})$. First, there is an initial consistency check in Step 1 to determine whether the input KB is satisfiable (this step can be skipped for $\mathcal{ELHI}$ KBs, which are always satisfiable). If the KB is shown to be unsatisfiable, then the query trivially holds, so the algorithm outputs `yes`. Otherwise, we initialize current with the pair (a, s_0) and count to 0. We also compute the maximum value max of the counter, which corresponds to the longest length of path that needs to be considered. At every iteration of the while loop (Step 3), we start with a single pair (c, s) stored in current and then proceed to guess a new pair (d, s') together with either a transition of the form (s, σ, s'), or a conjunction of concept names $M \in \mathsf{Loop}_\alpha[s, s']$. The first option corresponds to taking a step in the ABox, whereas the second corresponds to a shortcut through the anonymous part. In the first case, the idea is that we would like to append σd to the path guessed so far, but to do so, we must ensure that the conditions of paths are satisfied. This is the purpose of the entailment checks in Step 2(c). If we choose the second option, then we must check that the concept names in the selected conjunction M are entailed at the current individual. In both cases, if the applicable check succeeds, then we place (d, s') in current and increment count. We exit the while loop once we have reached the maximum counter value or the pair in count takes the form (b, s_f) with s_f a final state. In the latter case, we have managed to guess a path with the required properties, and so the algorithm returns `yes`.

[15] As earlier, we treat conjunctions of concepts as sets, ignoring order and repetitions.

[16] We note that for $\mathcal{ELH}$ and DL-Lite$_\mathcal{R}$, the construction is simpler as we only need to store concept names, rather than conjunctions of concept names.

ALGORITHM EvalAtom
INPUT: NFA $\alpha = (S, \Sigma, \delta, s_0, F)$ with $\Sigma \subseteq \mathsf{N}_\mathsf{R}^\pm \cup \{A? \mid A \in \mathsf{N}_\mathsf{C}\}$, $\mathcal{ELHI}_\bot$ KB $(\mathcal{T}, \mathcal{A})$, $(a, b) \in \mathsf{Ind}(\mathcal{A}) \times \mathsf{Ind}(\mathcal{A})$

1. Test whether $(\mathcal{T}, \mathcal{A})$ is satisfiable, output `yes` if not.
2. Initialize $\mathsf{current} = (a, s_0)$ and $\mathsf{count} = 0$. Set $\mathsf{max} = |\mathcal{A}| \cdot |S| + 1$.
3. While $\mathsf{count} < \mathsf{max}$ and $\mathsf{current} \notin \{(b, s_f) \mid s_f \in F\}$
   (a) Let $\mathsf{current} = (c, s)$.
   (b) Guess a pair $(d, s') \in \mathsf{Ind}(\mathcal{A}) \times S$ and either $(s, \sigma, s') \in \delta$ or $M \in \mathsf{Loop}_\alpha[s, s']$.
   (c) If (s, σ, s') was guessed
      – If $\sigma \in \mathsf{N}_\mathsf{R}^\pm$, then verify that $\mathcal{T}, \mathcal{A} \models \sigma(c, d)$, and return `no` if not.
      – If $\sigma = A?$, then verify that $c = d$ and $\mathcal{T}, \mathcal{A} \models A(c)$, and return `no` if not.
   (d) If M was guessed, then verify that $c = d$ and that $\mathcal{T}, \mathcal{A} \models B(c)$ for every concept name $B \in M$, and return `no` if not.
   (e) Set $\mathsf{current} = (d, s')$ and increment count.
4. If $\mathsf{current} = (b, s_f)$ for some $s_f \in F$, return `yes`. Else return `no`.

Fig. 5: Non-deterministic algorithm for 2RPQ answering in $\mathcal{ELHI}_\bot$.

Example 27. We describe a successful run of the algorithm EvalAtom on input α_8 from q_8, the KB $\mathcal{K}_5$ from Example 20, and the pair (p, p). We start with $\mathsf{current} = (p, s_0)$. In the first iteration, in Step 3(b) we guess (p, s_f) and $\mathsf{PizzaCalabrese} \in \mathsf{Loop}_\alpha[s_0, s_f]$. Then in Step 3(d) we verify that $p = p$ and $\mathcal{K}_5 \models \mathsf{PizzaCalabrese}(p)$. Since we now have a pair (p, s_f) and s_f is a final state of α_8, we exit the while loop and in Step 4, we return `yes`. This is correct, since $(p, p) \in \mathsf{cert}(q_8, \mathcal{K}_5)$. ▲

The correctness of this algorithm was proved for DL-Lite$_\mathcal{R}$ and $\mathcal{ELH}$ in [32], and the result can also be extended to $\mathcal{ELHI}_\bot$:

Proposition 1. *For every 2RPQ $q = \alpha(x, y)$, $\mathcal{ELHI}_\bot$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, and pair of individuals (a, b) from $\mathsf{Ind}(\mathcal{A})$: $(a, b) \in \mathsf{cert}(q, \mathcal{K})$ if and only if there is some execution of $\mathsf{EvalAtom}(\alpha, \mathcal{K}, (a, b))$ that returns* `yes`.

The algorithm EvalAtom needs to make calls to procedures that decide satisfiability, instance checking, and membership of a conjunction of concept names in the Loop_α table. We have discussed that for $\mathcal{ELHI}_\bot$, satisfiability and instance checking are feasible in EXP in combined complexity. Computing the Loop_α table takes polynomially many iterations in the size of α and $\mathcal{T}$, and each iteration may need to do some subsumption tests (for (L3) and (L4)), which require at most single exponential time. Hence, testing for loops can also be achieved using no more than single exponential time. Thus, we can view EvalAtom as a non-deterministic polynomial-time procedure that makes external calls to EXP procedures. Since $\mathrm{NP}^{\mathrm{EXP}} = \mathrm{EXP}$, we obtain an EXP upper bound for 2RPQ answering in $\mathcal{ELHI}_\bot$. To obtain a matching hardness result, we recall that instance checking is EXP-hard for $\mathcal{ELHI}_\bot$, and observe that instance checking can be reduced to answering a simple 2RPQ given by a regular expression of the form R or $A?$. Thus, 2RPQ answering is EXP-complete in combined complexity. For data complexity, we observe that computing the Loop_α table can be

done independently of the ABox $\mathcal{A}$, hence it takes only constant time in $|\mathcal{A}|$ to test whether $B \in \mathsf{Loop}_\alpha[s, s']$. Since in data complexity, satisfiability and entailment in $\mathcal{ELHI}_\bot$ are P-complete, we obtain a tight P upper bound for answering 2RPQs.

Since the complexity of $\mathsf{EvalAtom}$ is dominated by the complexity of entailment and instance checking, we can obtain better upper bounds for 2RPQ answering in sublogics of $\mathcal{ELHI}_\bot$ that are not hard for Exp. Indeed, both combined and data complexity drop to P-complete for $\mathcal{ELH}$ [19], and thus 2RPQ answering has the same complexity as subsumption and instance checking in $\mathcal{ELH}$. For DL-Lite$_\mathcal{R}$, satisfiability, subsumption and instance checking are known to feasible in NLogSpace in combined complexity and in $\mathsf{AC}_0 \subsetneq$ LogSpace in data complexity [52]. We do not obtain the same upper bounds for 2RPQ answering, but we can use these facts to argue that $\mathsf{EvalAtom}$ gives us P and NLogSpace upper bounds in combined in data complexity, respectively; both bounds are known to be tight [32].

Theorem 17 ([32]). *For $\mathcal{ELHI}_\bot$, 2RPQ answering is* Exp*-complete in combined complexity and* P*-complete in data complexity. For DL-Lite$_\mathcal{R}$ and $\mathcal{ELH}$, the combined complexity drops to* P*-complete. In data complexity, the problem is* NLogSpace*-complete for DL-Lite$_\mathcal{R}$, and* P *complete for $\mathcal{ELH}$.*

6.3 Extending the Approach to C2RPQs

Recall that 2RPQs are single-atom queries that do not contain quantified variables. The latter restriction turns out to be inessential, as the complexity results for 2RPQs hold also for single-atom queries with quantified variables. This can be easily shown for queries of the form $\exists x.\, \alpha(x, t)$ (with t a term), $\exists x.\, \alpha(t, x)$ (with t a term), and $\exists x, y.\, \alpha(x, y)$ (with $x \neq y$). Indeed, $q(x) = \exists y.\alpha(t, y)$ can be replaced by the 2RPQ $q'(t, y) = \alpha'(t, y)$, where $L(\alpha') = L(\alpha) \cdot \Gamma^*$, where $\Gamma = \mathsf{N}_\mathsf{R}^\pm \cap \mathsf{sig}(\mathcal{K})$, and then the answers to q are obtained by projecting the answers of q' to the first position. Likewise, $\exists x.\alpha(x, t)$ can be answered by taking a 2RPQ with regular language $\Gamma^* \cdot L(\alpha)$ and $\exists x, y.\alpha(x, y)$ by using $\Gamma^* \cdot L(\alpha) \cdot \Gamma^*$. Single-atom queries of the form $\exists x.\alpha(x, x)$ also have the same complexity, but showing this requires a more intricate proof [32].

Using a similar approach, one can show that the upper bounds in Theorem 17 hold even for the more general class of C2RPQs where every existentially quantified variable occurs in at most one role atom, that is, C2RPQs with no existential join variables, since they can be answered by combining the answers to a linear number of one-atom C2RPQs.

For arbitrary C2RPQs, which may have existential join variables, we require a more complex algorithm that combines the ideas discussed in Section 5.2 with the loop computation from the present section. Such an algorithm has been proposed in [32] for DL-Lite$_R$ and $\mathcal{ELH}$, and in [27] for handling an extension of C2RPQs over $\mathcal{ELHI}_\bot$ KBs. The basic idea is to rewrite the input C2RPQ into a set of C2RPQs for which we only need to consider matches that map all variables to ABox individuals. This may be accomplished using rewriting procedure that

is quite similar in spirit to the one we described for CQs, but considerably more involved since we need to take into account possible 'loops' that go deeper into the anonymous part than the image of the query variables. By combining this extended rewriting procedure with the 2RPQ answering algorithm described in this section, the following complexity results can be shown:

Theorem 18 ([27, 32]). *C2RPQ answering is*

1. NLogSpace*-complete in data complexity for DL-Lite*$_{\mathcal{R}}$*,*
2. P*-complete in data complexity for* $\mathcal{ELH}$ *and* $\mathcal{ELHI}_{\bot}$*,*
3. PSpace*-complete in combined complexity for DL-Lite*$_{\mathcal{R}}$ *and* $\mathcal{ELH}$*, and*
4. Exp*-complete in combined complexity for* $\mathcal{ELHI}_{\bot}$*.*

6.4 Results for Other DLs

The Exp upper bound in combined complexity for C2RPQs has been shown even for the significantly more expressive Horn-$\mathcal{SHOIQ}$ [166], which is in close correspondence with the Horn fragment of the expressive profiles of OWL. However, the technique employed there is quite different. In a nutshell, it considers all the (exponentially many) different ways of breaking the query into parts that are matched at the core, and parts that are matched in the trees. Answering the latter reduces to C2RPQ answering in the simpler Horn-$\mathcal{SHIQ}$. The authors use automata on infinite trees for this purpose, but the technique we have described could also be used: although presented for lightweight DLs [32], it is based on the earlier algorithm for Horn-$\mathcal{SHIQ}$ and extends easily to the latter. For the parts that are matched at the core, the authors use an explicit, step-by-step computation of all the possible relevant paths between ABox individuals, possibly passing by the anonymous part. This is somehow similar in spirit to the loop computation we have discussed, but necessarily more involved, since the relevant paths in Horn-$\mathcal{SHOIQ}$ are significantly more complex than simple loops (mainly due to the almost complete loss of the forest-like structure of the canonical models) and uses a Horn-$\mathcal{SHIQ}$ C2RPQ answering algorithm as an oracle. The results of [166] can be lifted to P2RPQs (defined analogously to PEQs) and cover also Horn-$\mathcal{SROIQ}$, which is even more expressive than Horn-$\mathcal{SHOIQ}$ and underlies OWL 2, the newest version of OWL [170]. However, for the latter DL, C2RPQ and P2RPQ answering are 2Exp-complete in combined complexity. If we consider data complexity, P2RPQ answering in all Horn logics between $\mathcal{EL}$ and Horn-$\mathcal{SROIQ}$ is complete for P, and all the algorithms we have mentioned run in polynomial time in the size of the ABox.

In non-Horn DLs, the complexity picture is very similar to CQs: the lack of universal models raises the data complexity to coNP-hard [193], and the combined complexity to 2Exp-complete, for every DL between $\mathcal{ALC}$ and the highly expressive $\mathcal{ZIQ}$, $\mathcal{ZOQ}$ and $\mathcal{ZOI}$ [58, 57]. The main difference with the (U)CQ setting is that even restricted classes of C2RPQs are 2Exp-hard for $\mathcal{ALC}$ [169] (by contrast, CQ answering in $\mathcal{ALCHQ}$ is in Exp), and that C2RPQs are undecidable already for $\mathcal{ALCOIF}$.

6.5 Navigational Queries Beyond (C)(2)RPQs

There has been considerable interest in recent years in extending (C)(2)RPQs with additional features that are considered important for applications. In particular, a useful XPath construct that is missing in C2RPQs is the possibility of using *test operators*, also known as *nesting*, to express sophisticated conditions along navigation paths. One simple way to introduce nesting into (C)(2)RPQs is to replace regular expression by so-called *nested regular expressions (NREs)*, in which one can use $\langle\rho\rangle$ to enforce the existence of an outgoing path that satisfies ρ, where ρ may itself be an NRE. For example, one could use $\langle\mathsf{awarded}\ \mathsf{MichelinStar}?\rangle$ to test whether a restaurant has been awarded a Michelin star and the NRE $(\mathsf{hasWorkedAt}\ \mathsf{Restaurant}?\ \langle\mathsf{awarded}\ \mathsf{MichelinStar}?\rangle\ \mathsf{hasWorkedAt}^-)^*$ to find chefs that are connected via a sequence of chefs such that every pair of adjacent chefs has worked at the same Michelin-starred restaurant. NREs were initially introduced for the purpose of defining nSPARQL, a navigational extension of SPARQL [174]. Subsequent investigations into the use of NREs for querying graph databases revealed them to have desirable computational properties [22, 23].

The query answering problem for (C)N2RPQs (defined using NREs) in the presence of DL ontologies was recently investigated in [27]. In that work, the authors show that, for a wide range of DLs, adding nesting to (C)2RPQs does not increase the worst-case data complexity of query answering. For expressive DLs, this can be shown by reducing CN2RPQs to plain C2RPQs, by introducing new concepts in the TBox that capture the nested expressions. For $\mathcal{ELHI}_\bot$ and its sublogics, one can use a more sophisticated version of the rewriting and loop procedures mentioned in Section 6.3. However, the news is not all positive as it was further shown that adding nesting leads to Exp-hardness in combined complexity, even for (non-conjunctive) 2NRPQs and the lightweight DLs DL-Lite and $\mathcal{EL}$. This negative result contrasts sharply with the tractable data complexity for the same setting but without nesting (cf. Theorem 17).

The preceding results have been complemented by three other recent works [203, 132, 43]. In [203], the authors consider the problem of answering (a slight extension[17] of) CNRPQs over OWL 2 EL knowledge bases. With regards to combined complexity, they establish a PSpace upper bound for CNRPQs and a P upper bound for NRPQs, thereby demonstrating that it is the combination of of nesting and inverses that leads to Exp-hardness. In [132], the authors investigate a variety of different XPath-inspired query languages, whose most expressive member essentially corresponds to N2RPQs extended with negation over unary and binary expressions. It is shown that negation over binary expressions immediately leads to undecidability, and the query answering problem for the path-positive fragment (allowing only unary negation) is coNP-complete in data complexity and Exp-complete in combined complexity for both DL-Lite$_R$ and $\mathcal{EL}$ (the Exp upper bound is shown for $\mathcal{ELHI}_\bot$). Finally, in [43], the authors compare three different approaches to extending C2RPQs with nesting,

[17] The query languages considered in [203, 132] also allow unary tests to be combined using conjunctive and disjunction. A similar construct was considered in [31].

with PFO+TC1 (positive FO queries with transitive closure on binary predicates) being the most expressive language. They establish a general decidability result for PFO+TC1 queries that holds for all DLs satisfying a quasi-forest model property, and for the DL $\mathcal{S}$, they show $(k+2)$-EXPTIME-hardness for queries with k levels of nesting of the transitive closure operator.

The issue of defining interesting path query languages that support nesting remains an active area of research in the database community, and there have been several recent proposals, including: regular queries [180], guarded regular queries [33], nested monadically defined queries [192], and the more general family of nested flag-and-check queries [44]. Beyond nesting and negation, (C)2RPQs have also been extended with path variables and regular relations [21].

7 Undecidability of Answering FO and Datalog Queries

We have seen in the preceding sections how various restricted forms of first-order and Datalog queries can be answered over Horn DL knowledge bases. Moreover, the procedures that we have devised run in polynomial time in the size of the ABox, making them suitable for applications involving large amounts of data. It is natural to wonder whether these nice computational properties extend to more expressive query languages, and in particular, to the classes of (full) first-order and Datalog queries defined in Section 3.2. Unfortunately, we will shall see that the answer is negative: not only do we lose tractability, but we even *lose decidability*. It is for this reason that full first-order and Datalog queries are not considered suitable query languages for OMQA.

7.1 First-order Queries

Because of the open-world semantics of DL knowledge bases, it is possible to reduce the validity problem for first-order sentences to the problem of answering Boolean FO queries over empty DL KBs. As the FO validity problem is undecidable, we obtain the following result.

Theorem 19. *First-order query answering is undecidable in every DL.*

The preceding theorem is quite discouraging, but it still leaves open the possibility that there may exist other natural classes of FO queries that are more expressive than (U)CQs, yet remain decidable in the presence of DL ontologies. Of particular interest are the *extensions of (U)CQs with negation or inequalities*, which have been extensively studied in the database setting. These query languages are formally defined as follows.

Definition 10 (Conjunctive Queries with Safe Negation). *A* conjunctive query with safe negation *($CQ^{\neg s}$) is an FO query of the form $q(\boldsymbol{x}) = \exists \boldsymbol{y}\ \varphi$ where φ is a conjunction of (positive) atoms and negated atoms using variables in $\boldsymbol{x} \cup \boldsymbol{y}$ and such that every variable occurs in at least one positive atom.*

Remark 14. The requirement that every variable occurs in some positive atom is made to ensure domain independence. Dropping this condition would mean allowing queries like $\neg\mathsf{Spicy}(x)$ that are not domain independent.

Example 28. The following CQ$^{\neg s}$ finds menus whose main course is not spicy.

$$\exists y\, \mathsf{Menu}(x) \wedge \mathsf{hasMain}(x, y) \wedge \neg\mathsf{Spicy}(y)$$

This query will return all individuals m such that in every model of the KB, m belongs to Menu and has an hasMain-successor that does *not* belong to Spicy. ▲

Definition 11 (Conjunctive Queries with Inequalities). *A* conjunctive query with inequalities *(CQ$^{\neq}$) is an FO query $q(\boldsymbol{x}) = \exists \boldsymbol{y}\ \varphi$ where φ is a conjunction of atoms and inequalities $t_1 \neq t_2$ whose variables are contained in $\boldsymbol{x} \cup \boldsymbol{y}$.*

Example 29. The following CQ$^{\neq}$ could be used to to find menus that contain at least three courses:

$$\begin{aligned}\exists y_1 y_2 y_3\ & \mathsf{Menu}(x) \wedge \mathsf{hasCourse}(x, y_1) \wedge \mathsf{hasCourse}(x, y_2) \wedge \mathsf{hasCourse}(x, y_3) \\ & \wedge y_1 \neq y_2 \wedge y_1 \neq y_3 \wedge y_2 \neq y_3\end{aligned}$$

Observe that this query could be captured using the concept $\mathsf{Menu} \sqcap \geq 3\mathsf{hasCourse}$ in DLs that allow for conjunction and unqualified number restrictions. In effect, by allowing inequalities in the language, we are able to express some limited form of number restrictions.

We could also use inequalities to find two menus offered by the same establishment that contain different dessert courses:

$$\begin{aligned}\exists y_1 y_2 z_1 z_2\ & \mathsf{offers}(x, y_1) \wedge \mathsf{Menu}(y_1) \wedge \mathsf{hasDessert}(y_1, z_1) \wedge \\ & \mathsf{offers}(x, y_2) \wedge \mathsf{Menu}(y_2) \wedge \mathsf{hasDessert}(y_2, z_2) \wedge z_1 \neq z_2\end{aligned}$$

This query is not expressible as a DL concept. ▲

Analogously to how we defined UCQs, we can define UCQ$^{\neg s}$s (resp. UCQ$^{\neq}$s) as disjunctions of CQ$^{\neg s}$s (resp. CQ$^{\neq}$s) that have the same answer variables.

The complexity and decidability of (unions of) CQ$^{\neg s}$ and CQ$^{\neq}$ was first investigated in [185], but it is only more recently that some key questions, such as the decidability of answering CQ$^{\neg s}$s and CQ$^{\neq}$s in DL-Lite$_R$, have been resolved [101]. While some open questions remain, the results obtained so far paint a decidedly negative picture:

Theorem 20. *The following problems are undecidable:*

- *CQ$^{\neg s}$ answering in DL-Lite$_R$[101]*
- *UCQ$^{\neq}$ answering in $\mathcal{EL}_\bot$ [185]*
- *CQ$^{\neq}$ answering in DL-Lite$_R$[101]*
- *CQ$^{\neq}$ answering in $\mathcal{EL}_\bot$ [122]*

We will not explain how the preceding decidability resuls are obtained, but merely note that in contrast to the query languages from the preceding sections, the answers to $\text{CQ}^{\neq}$s and $\text{CQ}^{\neg s}$s are not preserved under homomorphisms, and thus we are not able to use the universal model for query answering.

One solution that has been proposed in response to the undecidability of FO query answering is to adopt an alternative *epistemic semantics* [51]. The idea is to start with a standard DL query language $\mathcal{Q}$ (like IQs or CQs) and to introduce *epistemic atoms* of the form $\mathbf{K}q$ ($q \in \mathcal{Q}$), which are interpreted as the certain answers to q. These epistemic atoms can then be combined using the Boolean connectives and first-order logic quantifiers. To answer such a query, one may proceed by first computing the certain answers to the queries appearing in the epistemic atoms, storing the results in a database, and then evaluating a first-order query over this database. It follows that the query answering problem for the epistemic query language with embedded $\mathcal{Q}$-queries is decidable (resp. P in data complexity) in a DL $\mathcal{L}$ whenever $\mathcal{Q}$ answering in $\mathcal{L}$ is decidable (resp. P in data complexity).

Example 30. The epistemic query $\exists y\, \mathbf{K}\mathsf{Menu}(x) \wedge \mathbf{K}\mathsf{hasMain}(x, y) \wedge \neg\mathbf{K}\mathsf{Spicy}(y)$ returns all menus m that have a main dish d that is *not known to be spicy*, or more formally, d is not a certain answer to $\mathsf{Spicy}(x)$. Note that this is quite different from *knowing* that the main dish d is *not spicy* (and such a distinction may be relevant when choosing a menu!).

7.2 Datalog Queries

From the early days of description logic research, there has been significant interest in combining DLs with Datalog rules. Unfortunately, the combination of DLs and rules often leads to undecidability:

Theorem 21 ([144]). *Datalog query answering is undecidable in every DL that can express (directly or indirectly) an inclusion of the form $A \sqsubseteq \exists r.A$.*

Since $A \sqsubseteq \exists r.A$ is directly expressible in $\mathcal{EL}$ and can be simulated using the pair of DL-Lite inclusions $A \sqsubseteq \exists r$, $\exists r^- \sqsubseteq A$, we have the following:

Corollary 1. *Datalog query answering is undecidable in DL-Lite and $\mathcal{EL}$.*

It is worth noting that Datalog queries are preserved under homomorphisms, and thus, one can in principle evaluate a Datalog query over the (potentially infinite) universal model of a Horn DL knowledge base. However, running a Datalog program over the universal model leads to a new interpretation in which the domain elements may be arbitrarily connected, thus lacking the forest structure upon which many query answering techniques rely. For some restricted forms of Datalog queries, like the navigational queries from Section 6, it is still possible to develop techniques that exploit the forest structure of the universal model, but for general Datalog queries, the ability to arbitrarily connect unnamed objects leads to undecidability.

One simple way of regaining decidability is to enforce that Datalog rules be only applied to ABox individuals, rather than unnamed objects. This can be formalized using the notion of *(weak) DL-safety* [160, 184], which requires that every variable in a rule (head) occurs in a body atom whose relation does not appear in the TBox. DL-safe and weak DL-safe Datalog are considerably less expressive as query languages over DL KBs than unrestricted Datalog, but can nevertheless express some queries that are not captured by other decidable query languages we have considered. Using similar techniques to those presented in Section 5 for CQ answering in $\mathcal{ELHI}_\bot$, it was shown in [81] that the complexity of answering weak DL-safe Datalog queries over Horn-$\mathcal{SHIQ}$ KBs is EXP-complete in combined complexity and P-complete in data complexity.

8 Recent and Ongoing Research in OMQA

In this section, we provide an overview of recent work on OMQA and areas of ongoing research. Although we try to include many directions in which there are interesting developments, it is not a complete survey, and the discussion should not be considered exhaustive.

8.1 OMQA in DL-Lite

In the mid-2000's, Calvanese et al. [50, 52] introduced PerfectRef (for 'perfect reformulation'), the first query rewriting algorithm for DL-Lite, which was implemented in the QUONTO system [2]. The PerfectRef algorithm produces a UCQ-rewriting of the input CQ and TBox by interleaving *rewriting steps*, in which a query atom is rewritten by 'applying' a TBox inclusion in the backwards direction (e.g., rewriting $\mathsf{Menu}(x)$ into $\exists y.\mathsf{hasCourse}(x, y)$ using $\exists\mathsf{hasCourse} \sqsubseteq \mathsf{Menu}$), and *reduction steps*, in which unifiable atoms are merged (such unifications are essential to the completeness of the rewriting mechanism). The PerfectRef algorithm paved the way by showing how OMQA could be reduced to database query evaluation, but experiments showed that the UCQ-rewritings generated by PerfectRef were often extremely large (containing on the order of tens of thousands of CQs), making it very costly, and sometimes impossible, to compute and evaluate them. This spurred a whole line of research devoted to the design, implementation, and optimization of query rewriting algorithms. The RQR algorithm, proposed by Perez-Urbina, Motik and Horrocks and implemented in the REQUIEM system [175], achieved significantly better performance by transforming the input DL-Lite$_\mathcal{R}$ TBox and query into a set of first-order clauses and then applying a resolution procedure to compute a rewriting. The use of function symbols to handle existential axioms and the native support for axioms of the form $A \sqsubseteq \exists R.B$ (instead of having to simulate them via role inclusions, cf. Example 2) makes RQR more goal-oriented and allows it to avoid some unnecessary or redundant intermediate results. Further improvements were obtained by the RAPID system of Chortaras, Trivela and Stamou [67] which by virtue of

its more sophisticated resolution strategy and additional optimizations is able to substantially reduce the number of 'useless' inferences.

It should be noted that both REQUIEM and (the initial version of) RAPID generate UCQ-rewritings and thus are limited by the potentially huge size of the minimal UCQ-rewriting (the same holds for other UCQ-based rewriting approaches [95, 124, 211]). Indeed, it is not hard to see that the smallest UCQ-rewriting of a query may be exponentially large: take for instance the CQ $A_1(x) \wedge A_2(x) \wedge \ldots \wedge A_n(x)$ and the TBox $\{B_i^j \sqsubseteq A_i \mid 1 \leq i \leq n, 1 \leq i \leq m\}$, whose minimal UCQ-rewriting is a disjunction of $(m+1)^n$ CQs, corresponding to the $(m+1)^n$ different ways of choosing, for each $1 \leq i \leq n$, one of the concepts $A_i, B_i^1, \ldots, B_i^m$. This exponential blowup is commonly observed in practice due to the fact that real-world ontologies typically contain complex hierarchies of concepts, and thus there are often several choices of how to rewrite a given atom. Observe however that such choices can be compactly representing by adopting an alternative representation, e.g., the preceding CQ admits a short rewriting as a positive existential query $(\bigvee_{i=1}^{n}(A_i(x) \vee B_i^1(x) \vee \ldots \vee B_i^m(x)))$ or a *non-recursive Datalog* (NDL) program $(\{Q(x) \leftarrow Q_1(x), \ldots, Q_n(x)\} \cup \{Q_i(x) \leftarrow A_i(x), Q_i(x) \leftarrow B_i^j(x) \mid 1 \leq i \leq n, 1 \leq j \leq m\})$. This suggests that much more substantial gains in performance can be obtained by dropping the UCQ representation of rewritings in favour of more succinct query languages. This idea was first explored by Rosati and Almatelli whose PRESTO system [188] produces NDL-rewritings. An experimental evaluation showed it to significantly outperform the UCQ-based rewriting approaches; similar performances were obtained by RAPID$_d$, a variant of RAPID that outputs NDL-rewritings [68]. The *tree witness rewriting* of Kikot, Kontchakov, and Zacharyschev [119], which is utilized by the ONTOP system [181], provides another example of an NDL-rewriting approach. An experimental comparison showed it to be the most efficient among the considered NDL approaches and also confirmed the superiority of NDL-based rewriting algorithms over UCQ-based ones. We note in passing that in addition to NDL-rewritings, there have been some recent works producing different types of PE-rewritings, such as semi-conjunctive queries (SCQs) [206] and joins of unions of conjunctive queries (JUCQs) [46, 45]. In the latter work, different decompositions of the original query into subqueries give rise to a space of different JUCQ-rewritings, and a cost function is used to estimate the cost of executing a particular rewriting and to select the most efficient one.

Some further optimizations have been developed that are not applicable in every setting, but can lead to dramatic improvements in performance when they can be used. First, if one has control over the way that data is stored, then one may store concepts as integer values and assign these values in such a way that identifying the set of individuals satisfying a given concept can be achieved by posing simple range queries over the database. This technique, known as *semantic indexing*, has been shown to be very effective, and it is exploited by the UCQ-based QUEST rewriting engine of Rodriguez-Muro and Calvanese [182] and has been more recently used in combination with the aforementioned tree witness rewriting within the ONTOP system [181]. Another important type of

optimization involves the use of so-called *ABox dependencies* (also known as *extensional constraints*), which are TBox inclusions that hold in the interpretation (database) associated with the ABox. Intuitively, if we know that the ABox satisfies the TBox inclusion $A \sqsubseteq B$, then it is useless to rewrite the atom $B(x)$ into $A(x)$, since whenever the ABox contains $A(a)$, it must also contain $B(a)$. In the QUEST system, the TBox is simplified by removing inclusions that are made redundant by the constraints, and this simplified TBox is used during query rewriting. Further optimizations based upon exploiting extensional constraints, as well as disjointness and functionality axioms, were developed by Rosati and implemented in the PREXTO system [183]. We should mention that both QUEST and PREXTO produce UCQs, but the rewritings they generate can be significantly smaller than those of other UCQ-based systems, since they only need to work for ABoxes satisfying the constraints, rather than for arbitrary ABoxes.

The *combined approach* of Kontchakov et al. [126] represents an entirely different approach to achieving efficient answering in DL-Lite. The basic idea is to saturate the ABox using the TBox axioms, and then to evaluate the query over the saturated ABox. More precisely, one computes, in an offline phase, a finite first-order interpretation (i.e., a relational database) that corresponds to a compact representation of the canonical model of the KB (recall that we cannot in general compute the full canonical model, as it may be infinite). During the construction of this interpretation, new ABox individuals are introduced to serve as witnesses for the existential restrictions in the TBox axioms. However, to ensure finiteness, instead of generating several (possibly infinitely many) witnesses for the same inclusion (as in the canonical model construction), we will 'reuse' the same witnessing individual. If we now evaluate the input CQ over this new saturated interpretation, then we will be sure to obtain all of the certain answers, but we may also obtain some false answers due to the reuse of witnesses. There are two ways of addressing this issue. The first possibility (adopted in [126]) is to *rewrite* the CQ in order to block spurious answers and to evaluate the rewritten query over the saturated interpretation. If the TBox is formulated in the basic dialect DL-Lite$_{core}$, then the rewriting step results in a polynomial-size FO-query. However, for DL-Lite$_R$, the rewritten query may be exponentially large. Thus, an alternative approach, proposed in a subsequent work [149] and implemented in the COMBO system, consists in evaluating the original query over the saturated interpretation to get a superset of the certain answers, and then applying an external polynomial-time filtering procedure to weed out the spurious answers. An experimental evaluation comparing COMBO with RAPID and PRESTO showed it to be comparable to these systems in simpler settings, but much more robust to increases in the size of the data or the complexity of the concept hierarchy induced by the TBox.

We note that none of the preceding rewriting algorithms for DL-Lite$_R$ is guaranteed to terminate in polynomial time. While UCQ-based rewritings are necessarily exponential in the worst case, it is natural to wonder whether polynomial rewritings can be achieved by adopting the more succinct PE, NDL, and FO representations. A first negative result was obtained by Kikot, Kontchakov, and

Zacharyschev [118] who proved the impossibility of generating an FO-rewriting in polynomial time (unless P = NP) but left open the existence of polysize rewritings. In a series of subsequent works [120, 121, 29], the preceding authors, joined by Bienvenu and Podolskii, established tight connections between the size of rewritings of CQs w.r.t. DL-Lite$_R$ TBoxes and the circuit complexity of certain Boolean functions, which allowed them to pinpoint the worst-case size of PE-, NDL-, and FO-rewritings under various restrictions on the TBox and the input query. In general, the news is bad: even if we assume that the ABox has been saturated (i.e., we perform query answering over the core of the canonical model), PE- and NDL-rewritings can be exponentially large, and a superpolynomial lower bound on the size of FO-rewritings holds under the widely-held complexity-theoretic assumption that NP $\not\subseteq$ P/poly [120]. For PE-rewritings, this negative result cannot be easily escaped: the exponential lower bound applies even if the query is tree-shaped [120] or if the TBox has depth 2 (i.e., it can only produce canonical models whose elements are at most two 'steps' away from the ABox) [121], and a superpolynomial lower bound has recently been shown for the very restricted setting in which the input query is a linear CQ and the TBox has depth 2 [29]. In the case of NDL-rewritings, the picture is brighter: polysize NDL-rewritings always exist for tree-shaped queries with a bounded number of leaves (and arbitrary DL-Lite$_R$ TBoxes), and for bounded treewidth queries paired with bounded depth ontologies [29]. Moreover, an analysis of the combined complexity shows that CQ answering is tractable for these classes of queries and TBoxes, suggesting that it may be possible to define NDL-rewritings that can be both generated and evaluated in polynomial time (as was done in [34] for tree-shaped CQs in DL-Lite$_{\mathsf{core}}$). We should point out that the aforementioned negative results on the size of rewritings concern so-called *pure* rewritings, which do not use any constants other than those given in the query. Indeed, Gottlob and Schwentick [98] showed that if one admits existential quantification over two special constants (assumed to be present in every ABox), then polynomial-size NDL-rewritings exist for all CQs and DL-Lite$_R$ TBoxes, although it is unclear whether the obtained rewritings (which encode non-deterministic guesses using the special constants) can be successfully used in practice (see [93] for further discussion).

8.2 OMQA beyond DL-Lite

We next review the OMQA algorithms and systems that have been proposed for $\mathcal{EL}$ and its Horn extensions. A Datalog rewriting for $\mathcal{ELH}$ was defined by Rosati [186] and used to establish P data complexity of CQ answering in that logic. Pérez-Urbina, Motik and Horrocks [176] subsequently proposed a resolution-based Datalog rewriting algorithm for the much more expressive $\mathcal{ELHIO}^{\bot}$. The algorithm has been implemented in the previously mentioned Requiem system; it returns a UCQ-rewriting when the input ontology is in DL-Lite$_R$ and otherwise outputs a Datalog rewriting. The kyrie system [156] of Mora and Corcho is based upon the same resolution procedure as Requiem, but it includes several additional optimizations that significantly improve the running time. A new

version, KYRIE2, integrates optimizations based upon extensional constraints from the PREXTO system (see earlier). The first practical algorithm for CQ answering in Horn-$\mathcal{SHIQ}$, based upon Datalog rewriting, was proposed by Eiter et al. [81] and implemented in the CLIPPER system. We presented the main ideas underlying this algorithm in Section 5. The resolution-based RAPID system, first developed for DL-Lite$_R$, has been extended first to $\mathcal{ELHI}$ [207], and very recently to Horn-$\mathcal{SHIQ}$ [208]. It is highly optimized and outperforms CLIPPER, making it currently the most efficient approach available to handle all of Horn-$\mathcal{SHIQ}$. As noted in Section 5, CQ answering algorithms have been proposed for the even more expressive Horn-$\mathcal{SHOIQ}$ and Horn-$\mathcal{SROIQ}$ [166], but at the time of writing, there are no implemented systems targeting these DLs.

A highly influential line of work was initiated by Lutz, Toman, and Wolter [146] who introduced the *combined approach*, which we have already discussed for DL-Lite but was in fact first developed to handle $\mathcal{EL}$ and its extensions. In that work, they introduce the notion of *combined FO-rewritability*, which generalizes FO-rewritability by allowing a query-independent polynomial-time preprocessing step in which one builds an FO-interpretation from the ABox and TBox, followed by an ABox-independent query rewriting step that generates an FO-query whose evaluation of the interpretation yields the certain answers. As we have seen, the first step corresponds to compiling the TBox into the ABox and yields a finite representation of the canonical model, whereas the second step serves to block unsound answers that can result from approximating the possibly infinite canonical model with a finite interpretation (alternatively, one may replace the query rewriting step by a filtering step that identifies and discards the spurious answers). The interest of the combined approach is that it provides a means of exploiting relational database technology, while being applicable to a much wider range of DLs than (plain) FO-rewritability. In particular, the original paper by Lutz et al. showed that the approach could be applied to $\mathcal{ELH}_\bot^{dr}$ (which extends $\mathcal{ELH}_\bot$ with domain and range restrictions), for which CQ answering is P-hard in data complexity, thus preventing the use of plain FO-rewriting. For this logic, the query rewriting step involves only very simple modifications of the query and is guaranteed to terminate in polynomial time. Stefanoni, Motik, and Horrocks [202, 201] subsequently showed how the technique could be adapted to handle first nominals, then transitive roles. In both works, the saturation step is handled by means of a Datalog program, and following [149], filtering is used in place of rewriting to eliminate unsound answers. Very recently, Feier et al. [83] have further extended the combined approach to RSA, which is a fragment of Horn-$\mathcal{SHOIQ}$ that was introduced in [64] as a way of capturing the three OWL 2 profiles while retaining PTIME combined complexity for basic reasoning tasks (satisfiability and instance checking). Both the saturation and filtering steps are specified declaratively by means of a logic program with function symbols and stratified negation, and the answers are obtained by computing the minimal model of this program using a logic programming system (note however that one could equally well store the saturated interpretation as a database and leverage relational technology to perform the querying phase). Experiments conducted

on prototype implementations of the preceding algorithms show the combined approach to be highly effective. The principal drawback is that the saturated interpretation can be costly to compute, and it needs to be kept up to date, which may be problematic in applications in which the data changes frequently.

Another approach to using relational database systems to support OMQA with non-FO-rewritable ontology languages relies upon the observation is that while FO-rewritings need not exist for all CQs and all Horn DL ontologies, it is still possible that for particular query-ontology pairs, an FO-rewriting does exist (and hence relational technology can be used to answer such queries). Thus, an interesting and potentially quite useful research direction is to develop methods for identifying the cases where FO-rewriting is possible and to produce such rewritings when they exist. A first step in this direction was made by Bienvenu, Lutz, and Wolter [30], who established decidability and complexity results for FO-rewritability of IQs in Horn DLs, showing the problem to be PSPACE-complete for $\mathcal{EL}$ TBoxes and the full ABox signature and EXP-complete for $\mathcal{ELHI}$ and for $\mathcal{EL}$ if one may restrict the ABox signature (note that even if FO-rewritings do not exist for arbitrary ABoxes, they might exist for ABoxes formulated in a restricted signature). While these results were quite positive (similar problems in databases are known to be undecidable), the automata-based decision procedures used to show the upper bounds were ill-suited for implementation. Combining these theoretical results with an existing backward-chaining rewriting procedure [124], Hansen, Lutz, Seylan, and Wolter recently proposed a practical algorithm for testing FO-rewritability of IQs w.r.t. ontologies formulated in $\mathcal{ELH}^{dr}$. The algorithm has been implemented in the GRIND system, and experimental results on real-world ontologies are very encouraging: the vast majority of IQs do possess FO-rewritings, and the computed rewritings (represented as NDL programs) are typically quite small. The challenge in future work will be to see whether it is possible to extend this approach to more expressive Horn DLs and more expressive queries (in particular, CQs).

8.3 Querying Existing Relational Data Using Mappings

Throughout this chapter, we have assumed that the data is given as a set of ABox assertions, which may be stored as relational tables, Datalog facts, or RDF triples, but which only involve unary and binary relations (concepts and roles). However, in many applications, one is interested in using ontologies to query *existing* relational data, which typically involves relations of arity greater than two. In order to be able to apply the preceding techniques to arbitrary relational databases, it is necessary to provide a *mapping* that specifies the semantic relationship between the database relations and the concepts and roles in the considered DL vocabulary. Formally, a mapping is a finite set of *mapping assertions*, each taking the form $\varphi \rightarrow \psi$ where φ is an query formulated using the database relations and ψ is a query in the DL vocabulary. *Global-as-view (GLAV)* mappings, in which φ is a CQ and ψ is a single atom (without quantifiers), are the most commonly considered. Given a relational database and a GLAV mapping, we obtain the corresponding ABox by applying the mapping

assertions (viewed as rules) to the database, and the objective is to compute the certain answers over the KB consisting of this ABox and the TBox. The term *ontology-based data access* (OBDA for short) was originally coined to refer to this problem, but in recent years the term has taken on a more general meaning and is often used when speaking of the simpler OMQA setting without mappings.

Observe that by computing the ABox induced by the database and mappings, we end up with an OMQA problem, to which we can apply all of the techniques discussed in this chapter. However, it is often preferable to work with so-called *virtual ABoxes*, meaning that we use the mappings to define the ABox, but do not actually produce it. Indeed, if we work with DL-Lite ontologies and utilize an FO-rewriting approach, then we can proceed in three steps: (i) perform query rewriting as usual to obtain an FO-query that is guaranteed to give the right answers if it were evaluated over the (virtual) ABox, (ii) *unfold* the rewriting using the mapping assertions to obtain an FO-query over the database signature, and (iii) evaluate the resulting FO-query over the original relational database. This approach was first elaborated by Poggi et al. and implemented in the Mastro system [178]. Experience using this system in a real-world application with the Italian Ministry of Economy and Finance showed that the mapping unfolding phase yielded extremely large queries, which in many cases could not be handled by the database system. An analysis of the obtained queries revealed that they contained a lot of redundancies and could be significantly simplified by exploiting the containment relationships between the database queries appearing in the mapping assertions. This idea has been formalized in the PerfectMap algorithm [177], which has been incorporated into the Mastro system and experimentally validated on the aforementioned application. Mappings are also supported by the Ontop system [181]. In this approach, the TBox is integrated into the mapping in such a way that applying the mapping assertions directly generates all inferable assertions (i.e., the new mapping produces the core of the canonical model), and the tree witness rewriting is used to handle query matches that involve anonymous individuals. The FO-query obtained by unfolding the rewritten query using the modified mapping is simplified using *semantic query optimization*, which exploits the integrity constraints satisfied by the underlying database. Experiments with Ontop have shown that the resulting queries are typically of reasonable size and can be efficiently evaluated by relational database systems.

8.4 Inconsistency-tolerant Query Answering

While it may be reasonable to assume that the TBox has been properly debugged, the ABox is typically much larger and subject to more frequent modifications, making errors in the data almost inevitable. Such errors may render the KB inconsistent, making standard query algorithms next to useless (since when the KB is inconsistent, every tuple is trivially returned as an answer). Appropriate mechanisms for dealing with inconsistent data are thus crucial to the successful use of OMQA in practice. Ideally, one would restore consistency by identifying and correcting the errors, but when this is not possible, a sensible

strategy is to adopt an *inconsistency-tolerant semantics* which allows reasonable answers to be obtained despite the inconsistencies. The most well-known, and arguably the most natural, such semantics is the *AR semantics* [138], which was inspired by earlier work on consistent query answering in relational databases (see [24] for a survey). The semantics is based upon the notion of a *repair*, defined as an inclusion-maximal subset of the data that is consistent with the ontology. Repairs correspond to the different ways of achieving consistency while retaining as much of the original data as possible. Query answering under AR semantics amounts to computing those answers that hold in every repair. Two other natural repair-based semantics are the *brave semantics* [35], which only requires that an answer holds in *some* repair, and the more cautious *IAR semantics* [138], which corresponds to evaluating the query over the intersection of the repairs.

The complexity of answering queries under the AR semantics has been thoroughly investigated for a range of DLs [138, 187, 25, 35]. The results are rather discouraging: the problem is coNP-hard in data complexity already for instance queries in DL-Lite [138] and for conjunctive queries in any DL that can express disjointness of atomic concepts [25]. The IAR and brave semantics, which can be seen respectively as providing under- and over-approximations of the set of answers w.r.t. AR semantics, are more computationally well-behaved: for DL-Lite$_R$, both semantics can be computed using first-order query rewriting [139, 35], and thus has the same low complexity as CQ answering under classical semantics. Generalizing the IAR and brave semantics, Bienvenu and Rosati [35] introduced two parameterized families of inconsistency-tolerant semantics, called k-defeater and k-support semantics, that approximate the AR semantics from above and from below, respectively, and converge to the AR semantics in the limit. They established a general tractability result that applies to all known first-order rewritable languages, in particular many dialects of DL-Lite.

When information on the reliability of different facts is available, it is natural to use this information to identify *preferred repairs*, and to use the latter as the basis of inconsistency-tolerant query answering. A weight-based version of the AR semantics was first considered in the work of Du, Qi, and Shen [73]. More recently, Bienvenu, Bourgaux, and Goasdoué [26] studied the complexity of CQ answering in DL-Lite$_R$ under variants of the AR and IAR semantics based upon several different notions of preferred repairs, in which preferences are captured by cardinality, weights, or priority levels.

In terms of implementations, there are currently two systems for CQ answering over inconsistent DL-Lite KBs: the Quid system [189] implements the IAR semantics, using either query rewriting or ABox cleaning, and the CQAPri system [26] implements the AR, IAR and brave semantics, using tractable methods to obtain the answers under IAR and brave semantics and calls to a SAT solver to identify the answers holding under AR semantics (the system can also exploit preferences in the form of priority levels). For expressive DLs, Du et al. [73] have implemented a SAT-based algorithm for answering ground CQs (i.e., conjunctions of IQs) in $\mathcal{SHIQ}$ under weight-based AR semantics.

8.5 Temporal Query Answering

Time plays a central role in many application domains, and data is usually time-dependent: new contracts are signed, projects conclude, students graduate, menus change, etc. It is thus not surprising that the study of extensions of classical DLs that can model and reason about time is almost as old as DLs themselves, dating back to the early 1990s [194]. There are many different approaches to incorporating time into DLs, allowing for different design choices, which lead to a variety of temporal DLs with different computational properties. A prominent approach to construct temporal DLs is to combine DLs with dynamic formalisms, such as classical temporal logics like LTL and CTL, logics of time intervals [105], or action logics [11], and provide a two-dimensional semantics. For each such combination, there are other design choices to be made, like deciding to which components of the DL syntax (concepts, roles, ABoxes) temporal operators can be applied. There is a vast literature on temporal DLs; we refer to [13, 154, 12, 87] for surveys. Most work so far, however, focuses on so-called 'standard' reasoning tasks, like satisfiability testing and concept subsumption. Following in the steps of the research on classical DLs, the study of temporal DLs based on the lightweight DLs of the $\mathcal{EL}$ and DL-Lite families has become a very active area of research and with much progress in the last few years [16, 15, 102, 103].

Recently, the study of temporal OMQA is also receiving interest. A general framework for answering temporal queries over temporal data in the presence of classical ontologies was proposed in [104], considering queries with temporal operators over time-stamped databases, but with a *global* TBox (axioms hold at all moments of time) formulated in classical (non-temporal) DLs. For variations of this basic setting, decidability and tight complexity bounds for query answering have been obtained for expressive DLs between $\mathcal{ALC}$ and $\mathcal{SHQ}$ [18], and most recently for the $\mathcal{EL}$ family [40]. Borgwardt et al. have shown that in this setting query rewritability is preserved: the rewritability of the underlying (non-temporal) query language can be lifted to the temporal one [39], implying positive decidability results for OMQA in some Horn DLs.

A major limitation of these approaches is that they only consider global, atemporal TBoxes, and hence they do not allow for the highly desirable conceptual modeling of temporal properties needed, e.g., in applications related to data streams from sensor networks [14]. This can be supported by allowing for temporal operators to be used as regular concept constructors, which may also appear in TBoxes. Unfortunately, this extension makes query answering harder; in particular, the unrestricted use of temporal operators results in the loss of FO-rewritability for CQs over the DL-Lite family [16]. Positive results for FO-rewritability were obtained in [16] by restricting the set of available temporal operators and the occurrences of temporal concepts in the ontologies. Under these restrictions, rewriting into two-sorted FO with an order relation is indeed possible. The most recent work of these authors [14] carries out a detailed investigation of the limits of rewritability in the presence of more general forms of temporal TBoxes (e.g., more temporal operators are allowed) using as target

rewriting languages two-sorted FO with an order relation and addition, as well as monadic second order logic with an order relation.

8.6 Reasoning Support for Building and Maintaining OMQA systems

In order to use OMQA in a given application, one first requires an ontology that defines the terminology and the semantic relationships between the terms. As developing an ontology is difficult and time-consuming, it is important to provide tools to aid ontology engineers in this task. For ontology debugging, the key reasoning service is *axiom pinpointing* [195, 114], in which the problem is to generate minimal subsets of the KB that explain a given (surprising or undesirable) consequence; such subsets are often called *justifications*. For $\mathcal{ELH}$ TBoxes, justifications correspond to minimal models of propositional Horn formulas and can computed using SAT solvers [196]. In DL-Lite, the problem is simpler: all justifications of a TBox axiom can be enumerated in polynomial delay [173].

If suitable reference ontologies are available for the application area, then rather than starting from scratch, one may begin by extracting the relevant portions of existing ontologies. This is known as *module extraction* and has been the subject of a number of works in recent years, see e.g. [100, 204, 127, 129]. In the OMQA setting, one is typically interesting in modules that preserve answers to CQs. This can be formalized using the notion of *query inseparability* [123], in which two TBoxes $\mathcal{T}_1, \mathcal{T}_2$ are said to be Σ-query inseparable just in the case that they return the same answers to all queries formulated in the signature Σ for all Σ-ABoxes (a notion of query inseparability for KBs can be defined similarly [41]). Deciding query inseparability is a difficult task: the problem is EXP-complete for both $\mathcal{EL}$ [153] and DL-Lite$_R$ TBoxes [41]. Despite these discouraging results, Konev et al. [123] have shown that, by employing polynomial-time incomplete algorithms, it is possible to use query inseparability as the basis for practical module extraction in DL-Lite$_R$. Beyond module extraction, query inseparability can be used to analyze the effects of importing an ontology into another or of refining an ontology by adding additional axioms [153].

Another relevant reasoning service is *emptiness testing* [17], which comes in two flavours: *query emptiness* and *predicate emptiness*. The former is relevant when developing OMQA systems that propose a fixed set of predefined queries, as it allows one to detect whether a given query provides an empty answer over all ABoxes formulated in a given vocabulary, a serious modeling error. Predicate emptiness tests whether every query using a given predicate (concept or role) returns an empty answer (again for ABoxes over the specified signature). It can be used to identify the set of concept and role names that can be meaningfully used in queries (and thus should be included in the query formulation interface), and it can also serve as the basis for module extraction. The complexity of emptiness testing has been investigated for a range of DLs. In $\mathcal{EL}$, both forms of emptiness testing are tractable and amenable to efficient implementation.

If we are building a full-fledged OBDA system with mappings to link the ontology to a relational database (see Section 8.3), then it is also important to

provide support for constructing, debugging, and maintaining mappings. The problem of mapping debugging was first investigated by Lembo et al. [140, 141] who provided algorithms and complexity results for detecting inconsistencies and redundancies in mapping assertions. More recently, Bienvenu and Rosati [36] have initiated an investigation into query-based comparison of *OBDA specifications* (i.e., mapping-TBox pairs), in which two specifications are deemed equivalent if they give the same answers to the considered query or class of queries for all possible data sources. Such comparisons could be used, e.g., to simplify the specification or to determine whether changes to the ontology and/or mappings may impact query results.

8.7 Improving the Usability of OMQA Systems

In order for OMQA to be widely adopted in practice, it is essential that OMQA systems be easily usable by end users. In particular, it should be possible for users without any prior experience with ontologies to formulate queries that capture their information needs. This has motivated research into user-friendly interfaces that aid users in formulating their queries.

A pioneering project in this direction is Quelo [85], which provides a controlled natural language interface for users to interactively construct a query, starting from a very simple query (which is simply phrased as 'I am looking for something'), and adding additional constraints or modifying previously added ones. To support the edits of the user, the interface uses reasoning to retrieve, for example, which are the relevant constraints (concept and role names) that can be added or removed from the query at a given stage. The resulting query corresponds to a tree-shaped CQ, which can be written as a complex DL concept and answered using existing reasoning engines.

Later projects providing similar functionalities are the Faceted Search interface of Arenas et al. [8], and the Optique *virtual query formulation system (VQS)* [200]. Unlike Quelo, they do not aim at supporting natural language query formulation. Instead, the former provides faceted search facilities in which the user can interactively click and unclick several options to retrieve the desired information. Optique VQS is being developed within the Optique project[18], and it aims at providing an easy-to-use graphical interface that allows end users to easily build complex queries.

In addition to aiding users in formulating their queries, it is also important to help them understand the query results. As mentioned earlier, the problem of explanation has already been extensively studied in the DL community for the purposes of ontology debugging [155, 38, 195, 114, 196, 173, 107, 108]. These works have focused on explaining entailed TBox axioms (or possibly ABox assertions), but not answers to conjunctive queries. To the best of our knowledge, the first work to explicitly consider explanation in the OMQA setting was that of Borgida et al. [37], who proposed a proof-theoretic approach to explaining positive answers to CQs over DL-Lite$_\mathcal{A}$ KBs. The approach outputs a single

[18] `http://optique-project.eu/`

proof, involving both TBox axioms and ABox assertions, generated by 'tracing back' the relevant part of the rewritten query, with minimality criteria being used to select a 'simplest' proof. The problem of explaining negative query answers over DL-Lite$_\mathcal{A}$ KBs (that is, why is a given tuple *not* a certain answer?) has been investigated by Calvanese et al. ([62]). Formally, the explanations for $\mathsf{ans} \notin \mathsf{cert}(q, (\mathcal{T}, \mathcal{A}))$ correspond to (minimal) sets $\mathcal{A}'$ of ABox assertions such that $\mathsf{ans} \in \mathsf{cert}(q, (\mathcal{T}, \mathcal{A} \cup \mathcal{A}'))$. Practical algorithms for computing such explanations were proposed by Du, Wang, and Shen, first for consistent KBs [74] and then for inconsistent KBs [75]. Explanations of positive and negative query answers under the brave, AR, and IAR semantics (discussed in Section 8.4) have been explored by Bienvenu, Bourgaux, and Goasdoué [42].

8.8 OMQA with Closed Predicates

As discussed in Section 6, ABoxes are interpreted under the open world semantics, while databases are given a closed world semantics. However, there are many applications where the open world semantics of DLs is too weak and it does not allow us to obtain all the desired inferences. For example, suppose the data to be queried contains the students enrolled in a specific course, which are extracted from a database that is known to be complete. Then this information should be considered complete (even if other parts of the data are not), and query answering algorithms should exploit this to exclude irrelevant models and infer more query answers.

Combining open and closed world reasoning in DLs is not a new topic [47], but it has received renewed attention in recent years [150, 86, 197]. A way of achieving partial closed world reasoning is to consider DBoxes [197], which syntactically look just like ABox, but semantically, they are interpreted like a database: the instances of the concepts and roles in a DBox are given exactly by the assertions it contains, and the *unique name assumption* is made for the *active domain* of the individuals occurring in it. More recent approaches enrich the knowledge base by specifying a set of concepts and roles that are to be interpreted as *closed predicates* [150]. In this way, some ABox assertions are interpreted under closed semantics, as in DBoxes, while others are considered open, as in ABoxes.

Most works on reasoning with closed predicates focus on studying the data complexity of query answering. Unfortunately, the problem is NP-hard even for the core fragments of DL-Lite [86]. The authors of that work established a matching upper bound for DL-Lite with functionality (the interaction of the latter with the closed predicates and inverse roles makes the problem particularly challenging). An in-depth analysis of the reasons for NP-hardness, as well as criteria for showing tractability for specific TBoxes, is provided by Lutz et al. [150]). In more recent work [151], the same authors explore the problem of classifying specific TBox-query pairs according to their data complexity and, among other contributions, identify some FO-rewritable cases. The combined complexity of querying with closed predicates had not been studied until very recently [162], but it has now been shown that query answering is at least coNExp hard for any extension of $\mathcal{EL}$, and in most cases 2Exp complete. Some of these complexity bounds

are not hard to infer from the standard open world setting, using ideas that had already been exploited by Franconi et al. [86] to show that query answering in $\mathcal{ALCIF}$ with closed predicates is equivalent to standard query answering in $\mathcal{ALCOIF}$, whose complexity is a longstanding open problem.

8.9 Aggregates

Aggregate functions like *max, min, count, sum* and *avgr* are among the most frequently used features of popular query languages, including SQL. In the context of OMQA, they have received surprisingly little attention. This is mostly due to the fact that the certain answer semantics is not very suitable for aggregates, and it is not always clear what their expected meaning should be under the open-world semantics. For example, if a knowledge base only states that Mary teaches a course, then we can build models where she teaches n courses for every n, and there are no certain answers to the query 'How many courses does Mary teach?'. This is in fact the semantics given to these kind of queries in the first work on OMQA with aggregates [60], where the authors adopt an *epistemic* semantics where aggregation is only done over the data that coincides in all models. A more recent work revisiting this topic gives a stronger semantics for *count* and *count distinct* that in the example above would allow us to infer a lower bound of one course as an answer [131]. That work does not consider other aggregate functions. Moreover, the proposed semantics is rather costly: already for DL-Lite, deciding if a number is in the answer is hard for coNP in data complexity and for the second level of the polynomial hierarchy in combined complexity. In DL-Lite$_{\mathcal{R}}$, the combined complexity is even coNExp-hard.

8.10 Bridging the Gap with SPARQL

SPARQL is the standard language for querying RDF datasets [179]. The core of the SPARQL language are so-called *basic graph patterns (BGPs)*, which essentially correspond to CQs. SPARQL also provides additional constructs to build complex queries from BGPs. Some of these constructors, like union, have a natural counterpart in FO queries and the languages we have discussed in this chapter. Others do not directly correspond to FO connectives but are still expressible in FO. In fact, it has been shown that SPARQL, as a query language, is equivalent to relational algebra, and hence to domain-independent FO queries [4]. This implies, unfortunately, that full SPARQL is undecidable if we use it as query language in our OMQA setting. In practice, SPARQL is often used as a query language for query answering in the presence of ontologies, but with a somewhat different semantics defined in the so-called *entailment regimes*, see [88] and its references. We note that SPARQL supports aggregate functions in queries, which we have discussed above. There is a newer version of the SPARQL standard, SPARQL 1.1, [106], and one of its core features is to add the co-called *property paths*, that basically correspond to regular expressions as in C2RPQs.

Optional operator. A useful feature of SPARQL is the OPTIONAL operator. In the query languages we have discussed, query answers always take the form of a relation (that is, a set of tuples of individuals) of a fixed arity. Using OPTIONAL, one can define queries where binding some of the variables is optional and thereby obtain as answers tuples of different arities, where the optional variables are matched if possible, but left unassigned otherwise. For example, we can retrieve pairs of dishes and restaurants where they are served, and optionally retrieve also their price if it is available. This can be very useful in the presence of incomplete information, hence it would be a good feature to add to CQs, C2RPQs, or the other query languages for OMQA that we have discussed. Unfortunately, the presence of OPTIONAL makes queries non-monotonic. This means that, unlike CQs and other *positive* fragments of FO queries, they are not preserved under homomorphisms. Hence there is no analogous to Theorem 13, and we cannot rely on the existence of a universal model for answering these queries. The query answering algorithm that we discussed in Section 5 has been extended to a family of well-behaved CQs with OPTIONAL [3]. Other recent works also aim at giving a suitable semantics to fragments of SPARQL in the presence of ontologies, and devising query answering algorithms [7, 130].

Meta-modeling and meta-querying. Standard DLs and the query languages usually employed for OMQA do not have meta-modeling and meta-querying functionalities. Intuitively, in meta-querying, queries can ask for properties of concepts and roles using variables that are bound to such objects (instead of binding variables to individuals only). Meta-modeling can be seen as a generalization of this, where one can use concept and roles as predicate arguments already in the knowledge base. This allows one to assert properties of concept and roles and to ask for such properties in queries. Meta-modeling was considered in the early days of DLs, but nowadays it is not supported in standard DLs. Meta-modeling and meta-querying are both popular in the semantic web, and they are supported by RDF and SPARQL.

There have been a few extensions of DLs with meta-modeling functionalities [92, 171, 157, 72, 70], which are obtained by introducing features from higher-order logics. It has been shown that, under certain conditions, these higher-order extensions of DLs do not increase the worst-case complexity of reasoning [72]. However, it has also been observed that even when the straightforward adaptation of reasoning algorithms to the setting of higher-order DLs does not increase their worst-case complexity, it can make them less practicable, and improved algorithms for CQ answering in the higher-order version of DL-Lite$_\mathcal{R}$ have been proposed [143].

8.11 Extending the Applicability of Horn DL Techniques

As we have discussed, non-Horn DLs require significantly more involved query answering algorithms that the ones presented in this chapter, and they usually have a higher computational complexity. Even for traditional reasoning tasks that have the same worst case complexity in the Horn and the non-Horn case

(e.g., satisfiability in $\mathcal{SHIQ}$ vs. Horn-$\mathcal{SHIQ}$, which are both EXP-complete), the techniques for Horn logics are in general more amenable to implementation and more efficient in practice. For this reason, some researchers have recently aimed at understanding when reasoning in a non-Horn DL can be efficiently reduced to reasoning in a Horn one [65, 63]. In [116, 115], the authors follow a similar idea, but instead of rewriting into a Horn ontology, they rewrite into Datalog, for which efficient off-the-shelf reasoners are available. Unfortunately, most of these results apply only to satisfiability and instance queries, and only [116] presents some results for CQs.

Datalog has also been exploited to achieve scalable query answering in non-Horn logics. For example, the authors of [212] first use a Datalog reasoner to approximate the answers both from below and from above. That is, they obtain a sound, possibly incomplete set of answers (lower bound), and a complete, possible unsound set of answers (upper bound). Both computations can be done efficiently by reducing them to the evaluation of suitable Datalog programs. If the upper and lower bounds coincide, then running an expensive exact algorithm becomes unnecessary. If they are different, then the difference gives the set of potential answers for which an exact algorithm is necessary. Moreover, even where expensive exact algorithms are needed, it is possible to exploit the candidate answers to optimize the algorithm and reduce the search space.

We also note that there have been a few works aiming at applying FO- and Datalog-rewriting to non-Horn DLs. The problem of deciding existence of an FO- or Datalog-rewriting of IQs for DLs between $\mathcal{ALC}$ and $\mathcal{SHI}$ was shown in [28] to be NEXPTIME-complete by establishing a tight connection between OMQA with expressive DLs and non-uniform constraint satisfaction problems.

8.12 Rule-based Ontology Languages

In this chapter, we have only discussed ontologies expressed in DLs. Another important and closely related alternative is to express domain knowledge using rules. Indeed, in the absence of existential quantification, most of the Horn DLs we have discussed could be expressed as Datalog rules. Expressive rule languages that extend Datalog with existentially quantified variables in rule heads have been devised with the explicit purpose of expressing DLs, and ontological knowledge in general. Since Datalog with existential quantification is well known to be undecidable, these extensions must be done in a cautious and controlled way, and restrictions must be imposed, such as certain acyclicity conditions or allowing only *guarded* quantification. The resulting families of languages are known under the names of *existential rules* or Datalog$^{\pm}$, and they can be seen as generalizations of DLs to predicates of arbitrary arity, rather than only unary and binary. For an overview of the area and its main results, we refer the reader to recent tutorials in the Reasoning Web series of summer schools [161, 96] and to the Datalog$^{\pm}$ tutorial in this volume. Here we only point out that there is a large and very active research community studying the OMQA problem in the presence of existential rules and Datalog$^{\pm}$ and that they share much of the same research agenda as for OMQA with DLs. In fact, several of the results

and techniques we have discussed in this chapter have been extended to rule-based ontologies, including a significant amount of work on query rewriting, see e.g.,[95, 124, 206, 125] and references therein. The combined approach has also been extended to existential rules [94], and the saturation approach discussed in Section 4 has been adapted, and combined with a technique similar in spirit to the rewriting in Section 5, to reduce the OMQA problem for existential rules to plain Datalog reasoning [97].

9 Concluding Remarks

In this chapter, we have given an introduction to the OMQA problem, an active area of ongoing research. By allowing semantic knowledge to be exploited when querying data, OMQA opens many new perspectives for modern information systems. However, taking into account this additional knowledge raises significant computational challenges. We have discussed some algorithmic techniques, based on the key ideas of query rewriting and saturation, which allow us to overcome these challenges and effectively answer different kinds of queries. We have focused on so-called *Horn* DLs for which conjunctive query answering can be performed in polynomial time in the size of the data, and we have briefly discussed what happens if we adopt more expressive DLs. Table 5 summarizes some of the main complexity results for the OMQA problem, for a range of DLs and query languages.

We have also surveyed many recent results and current research directions. The current OMQA/OBDA technologies are mature enough to be deployed in all kinds of application areas. They have been successfully applied in many challenging real life applications, including investment risk analysis, configuration and data management management of mobile telecommunication data [49], and management of public debt data [5]. The large ongoing project Optique is applying these technologies in the energy sector, supporting diagnosis engineers at power plants service centres and experts in oil exploration. Another large ongoing project called EPNet[19] uses OBDA to help access and manage data about food transportation in the Roman empire. These projects witness the versatility and potential of exploiting ontological knowledge when querying data.

While there has been much progress over the past few years, many open questions remain, and there are many more challenges to be overcome. Readers interested in keeping up with the latest results and research trends in OMQA can refer to the Informal Proceedings of the International Workshop on Description Logics, the annual gathering of the DL research community. The proceedings are published in the free, open-access CEUR Workshop Proceedings series (`http://ceur-ws.org/`), and a historic archive of the workshop editions with links to the proceedings can be found on the Description Logics website (`http://dl.kr.org/workshops/`).

[19] `http://www.roman-ep.net`

	IQs		CQs		2RPQs		C2RPQs	
	data complexity	combined complexity	data complexity	combined complexity	data complexity	combined complexity	data complexity	combined complexity
DL-Lite DL-Lite$_{\mathcal{R}}$	in AC_0	NLogSpace	in AC_0	NP	NLogSpace	P	NLogSpace	PSpace
$\mathcal{EL}, \mathcal{ELH}$	P	P	P	NP	P	P	P	PSpace
$\mathcal{ELI}$, $\mathcal{ELHI}_\bot$, Horn-$\mathcal{SHOIQ}$	P	Exp	P	Exp	P	Exp	P	Exp
$\mathcal{ALC}$, $\mathcal{ALCHQ}$	coNP	Exp	coNP	Exp	coNP	Exp	coNP-hard	2Exp
$\mathcal{ALCI}$, $\mathcal{SH}$, $\mathcal{SHIQ}$	coNP	Exp	coNP	2Exp	coNP	Exp	coNP-hard	2Exp
$\mathcal{SHOIQ}$	coNP-hard	coNExp	coNP-hard[1]	coN2Exp-hard[1]	coNP-hard	coNExp	coNP-hard[2]	coN2Exp-hard[2]

Table 5: The complexity of OMQA. All results are completeness results, unless stated otherwise. For references, please refer to the sections on the corresponding query languages.

[1] Decidability if only simple roles occur in the query follows from [191], but no complexity upper bounds are known.

[2] Decidability remains open.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison Wesley Publ. Co. (1995)
2. Acciarri, A., Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Palmieri, M., Rosati, R.: QuOnto: QUerying ONTOlogies. In: Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005). pp. 1670–1671 (2005)
3. Ahmetaj, S., Fischl, W., Pichler, R., Šimkus, M., Skritek, S.: Towards reconciling SPARQL and certain answers. In: Proc. of the 24th Int. Conf. on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015. pp. 23–33 (2015)
4. Angles, R., Gutierrez, C.: The expressive power of SPARQL. In: The Semantic Web - ISWC 2008, Lecture Notes in Computer Science, vol. 5318, pp. 114–129. Springer Berlin Heidelberg (2008)
5. Antonioli, N., Castanò, F., Coletta, S., Grossi, S., Lembo, D., Lenzerini, M., Poggi, A., Virardi, E., Castracane, P.: Ontology-based data management for the italian public debt. In: Proc. 8th Int. Conf. Formal Ontology in Information Systems (FOIS 2014). Frontiers in Artificial Intelligence and Applications, vol. 267, pp. 372–385. IOS Press (2014)
6. Arenas, M., Barceló, P., Libkin, L., Murlak, F.: Foundations of Data Exchange. Cambridge University Press (2014)
7. Arenas, M., Gottlob, G., Pieris, A.: Expressive languages for querying the semantic web. In: Proc. of the 33rd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2014). pp. 14–26. ACM, New York, NY, USA (2014)
8. Arenas, M., Grau, B.C., Kharlamov, E., Marciuška, S., Zheleznyakov, D.: Faceted search over ontology-enhanced RDF data. In: Proc. of the 23rd ACM Int. Conf. on Conference on Information and Knowledge Management (CIKM). pp. 939–948 (2014)
9. Arora, S., Barak, B.: Computational Complexity - A Modern Approach. Cambridge University Press (2009)
10. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The DL-Lite family and relations. J. Artif. Intell. Res. (JAIR) 36, 1–69 (2009)
11. Artale, A., Franconi, E.: A temporal description logic for reasoning about actions and plans. J. of Artificial Intelligence Research 9, 463–506 (1998)
12. Artale, A., Franconi, E.: Temporal description logics. In: Handbook of Time and Temporal Reasoning in Artificial Intelligence. The MIT Press (2001)
13. Artale, A., Franconi, E.: Temporal description logics. In: Handbook of Temporal Reasoning in Artificial Intelligence, pp. 375–388. Foundations of Artificial Intelligence, Elsevier (2005)
14. Artale, A., Kontchakov, R., Kovtunova, A., Ryzhikov, V., Wolter, F., Zakharyaschev, M.: First-order rewritability of ontology-mediated temporal queries. In: Proc. of the 24th Int. Joint Conf. on Artificial Intelligence (IJCAI 2015) (2015)
15. Artale, A., Kontchakov, R., Ryzhikov, V., Zakharyaschev, M.: A cookbook for temporal conceptual data modelling with description logics. ACM Trans. Comput. Logic 15(3), 25:1–25:50 (Jul 2014)
16. Artale, A., Kontchakov, R., Wolter, F., Zakharyaschev, M.: Temporal description logic for ontology-based data access. In: Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI 2013) (2013)
17. Baader, F., Bienvenu, M., Lutz, C., Wolter, F.: Query and predicate emptiness in description logics. In: Proc. of the 12th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2010) (2010)

18. Baader, F., Borgwardt, S., Lippmann, M.: Temporal query entailment in the description logic $\mathcal{SHQ}$. Web Semantics: Science, Services and Agents on the World Wide Web. In press, doi:10.1016/j.websem.2014.11.008 (2014)
19. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005) (2005)
20. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope further. In: Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008) (2008)
21. Barceló, P., Libkin, L., Lin, A.W., Wood, P.T.: Expressive languages for path queries over graph-structured data. ACM Trans. on Database Systems 37(4), 31 (2012)
22. Barceló, P., Pérez, J., Reutter, J.L.: Relative expressiveness of nested regular expressions. In: Proc. of AMW'12. pp. 180–195. CEUR Workshop Proc. 866 (2012)
23. Barceló Baeza, P.: Querying graph databases. In: Proc. of the 32nd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2013). pp. 175–188 (2013)
24. Bertossi, L.E.: Database Repairing and Consistent Query Answering. Synthesis Lectures on Data Management, Morgan & Claypool Publishers (2011)
25. Bienvenu, M.: On the complexity of consistent query answering in the presence of simple ontologies. In: Proc. of the 26th AAAI Conference on Artificial Intelligence (AAAI 2012) (2012)
26. Bienvenu, M., Bourgaux, C., Goasdoué, F.: Querying inconsistent description logic knowledge bases under preferred repair semantics. In: Proc. of the 28th AAAI Conference on Artificial Intelligence (AAAI 2014) (2014)
27. Bienvenu, M., Calvanese, D., Ortiz, M., Šimkus, M.: Nested regular path queries in description logics. In: Proc. of the 14th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2014) (2014)
28. Bienvenu, M., ten Cate, B., Lutz, C., Wolter, F.: Ontology-based data access: A study through disjunctive datalog, csp, and MMSNP. ACM Trans. Database Syst. 39(4), 33:1–33:44 (2014)
29. Bienvenu, M., Kikot, S., Podolskii, V.V.: Tree-like queries in OWL 2 QL: Succinctness and complexity results. In: Proc. of the 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2015). IEEE (2015)
30. Bienvenu, M., Lutz, C., Wolter, F.: First-order rewritability of atomic queries in horn description logics. In: Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI 2013). IJCAI/AAAI (2013)
31. Bienvenu, M., Ortiz, M., Šimkus, M.: Answering expressive path queries over lightweight DL knowledge bases. In: Proc. of the 25th Int. Workshop on Description Logic (DL 2012) (2012)
32. Bienvenu, M., Ortiz, M., Šimkus, M.: Conjunctive regular path queries in lightweight description logics. In: Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI 2013) (2013)
33. Bienvenu, M., Ortiz, M., Šimkus, M.: Navigational queries based on frontier-guarded datalog: Preliminary results. In: Proc. of the Ninth Alberto Mendelzon International Workshop on Foundations of Data Management (AMW 2015) (2015)
34. Bienvenu, M., Ortiz, M., Šimkus, M., Xiao, G.: Tractable queries for lightweight description logics. In: Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI 2013). AAAI Press (2013)
35. Bienvenu, M., Rosati, R.: Tractable approximations of consistent query answering for robust ontology-based data access. In: Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI 2013) (2013)

36. Bienvenu, M., Rosati, R.: Query-based comparison of OBDA specifications. In: Proc. of the 29th Int. Workshop on Description Logic (DL 2015) (2015)
37. Borgida, A., Calvanese, D., Rodriguez-Muro, M.: Explanation in the DL-Lite family of description logics. In: Proc. of OTM (2008)
38. Borgida, A., Franconi, E., Horrocks, I.: Explaining ALC subsumption. In: Proc. of ECAI (2000)
39. Borgwardt, S., Lippmann, M., Thost, V.: Temporalizing rewritable query languages over knowledge bases. Web Semantics: Science, Services and Agents on the World Wide Web. In press, doi:10.1016/j.websem.2014.11.007 (2014)
40. Borgwardt, S., Thost, V.: Temporal query answering in the description logic $\mathcal{EL}$. In: Proc. of the 24th Int. Joint Conf. on Artificial Intelligence (IJCAI 2015) (2015)
41. Botoeva, E., Kontchakov, R., Ryzhikov, V., Wolter, F., Zakharyaschev, M.: Query inseparability for description logic knowledge bases. In: Proc. of the 14th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2014) (2014)
42. Bourgaux, C., Bienvenu, M., Goasdoué, F.: Explaining query answers under inconsistency-tolerant semantics over description logic knowledge bases (extended abstract). In: Proc. of the 29th Int. Workshop on Description Logic (DL 2015) (2015)
43. Bourhis, P., Krötzsch, M., Rudolph, S.: How to best nest regular path queries. In: Proc. of the 27th Int. Workshop on Description Logic (DL 2014). vol. 1193, pp. 404–415. CEUR-WS.org (2014)
44. Bourhis, P., Krötzsch, M., Rudolph, S.: Query containment for highly expressive datalog fragments. CoRR abs/1406.7801 (2014), `http://arxiv.org/abs/1406.7801`
45. Bursztyn, D., Goasdoué, F., Manolescu, I.: Efficient query answering in DL-Lite through FOL reformulation. In: Proc. of the 29th Int. Workshop on Description Logic (DL 2015) (2015)
46. Bursztyn, D., Goasdoué, F., Manolescu, I.: Optimizing reformulation-based query answering in RDF. In: Proc. of the 18th Int. Conf. on Extending Database Technology (EDBT). pp. 265–276 (2015)
47. Cadoli, M., Donini, F.M., Schaerf, M.: Closed world reasoning in hybrid systems. In: Proc. of the 5th Int. Symp. on Methodologies for Intelligent Systems (ISMIS'90). pp. 474–481. North-Holland Publ. Co. (1990)
48. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R.: Ontologies and databases: The DL-Lite approach. In: Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School 2009. Lecture Notes in Computer Science, vol. 5689, pp. 255–356. Springer (2009)
49. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., Savo, D.F.: The MASTRO system for ontology-based data access. Semantic Web 2(1), 43–53 (2011)
50. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: DL-Lite: Tractable description logics for ontologies. In: Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005). pp. 602–607 (2005)
51. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: EQL-Lite: Effective first-order query processing in description logics. In: Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007). pp. 274–279 (2007)
52. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. Journal of Automated Reasoning 39(3), 385–429 (2007)

53. Calvanese, D., De Giacomo, G., Lenzerini, M.: On the decidability of query containment under constraints. In: Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98). pp. 149–158 (1998)
54. Calvanese, D., De Giacomo, G., Lenzerini, M.: Conjunctive query containment and answering under description logics constraints. ACM Trans. on Computational Logic 9(3), 22.1–22.31 (2008)
55. Calvanese, D., De Giacomo, G., Lenzerini, M., Vardi, M.Y.: Containment of conjunctive regular path queries with inverse. In: Proc. of the 7th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2000). pp. 176–185 (2000)
56. Calvanese, D., Eiter, T., Ortiz, M.: Answering regular path queries in expressive description logics: An automata-theoretic approach. In: Proc. of the 22nd AAAI Conf. on Artificial Intelligence (AAAI 2007). pp. 391–396 (2007)
57. Calvanese, D., Eiter, T., Ortiz, M.: Regular path queries in expressive description logics with nominals. In: Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI 2009). pp. 714–720 (2009)
58. Calvanese, D., Eiter, T., Ortiz, M.: Answering regular path queries in expressive description logics via alternating tree-automata. Inf. Comput. 237, 12–55 (2014)
59. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. In: Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006). pp. 260–270. AAAI Press (2006)
60. Calvanese, D., Kharlamov, E., Nutt, W., Thorne, C.: Aggregate queries over ontologies. In: Proc. of the 2nd International Workshop on Ontologies and Information Systems for the Semantic Web, ONISW 2008, Napa Valley, California, USA, October 30, 2008. pp. 97–104 (2008)
61. Calvanese, D., Ortiz, M., Šimkus, M.: Containment of regular path queries under description logic constraints. In: Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI 2011) (2011)
62. Calvanese, D., Ortiz, M., Šimkus, M., Stefanoni, G.: Reasoning about explanations for negative query answers in DL-Lite. J. Artif. Intell. Res. (JAIR) 48, 635–669 (2013)
63. Carral, D., Feier, C., Grau, B.C., Hitzler, P., Horrocks, I.: *EL*-ifying ontologies. In: Automated Reasoning - 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8562, pp. 464–479. Springer (2014)
64. Carral, D., Feier, C., Grau, B.C., Hitzler, P., Horrocks, I.: Pushing the boundaries of tractable ontology reasoning. In: The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part II. pp. 148–163 (2014)
65. Carral, D., Feier, C., Romero, A.A., Grau, B.C., Hitzler, P., Horrocks, I.: Is your ontology as hard as you think? Rewriting ontologies into simpler dls. In: Informal Proc. of the 27th International Workshop on Description Logics, Vienna, Austria, July 17-20, 2014. CEUR Workshop Proceedings, vol. 1193, pp. 128–140. CEUR-WS.org (2014)
66. Ceri, S., Gottlob, G., Tanca, L.: Logic Programming and Databases. Springer, Berlin (Germany) (1990)

67. Chortaras, A., Trivela, D., Stamou, G.: Optimized query rewriting for OWL 2 QL. In: Proc. of the 23rd Int. Conf. On Automated Deduction. pp. 192–206. CADE'11, Springer, Berlin, Heidelberg (2011)
68. Chortaras, A., Trivela, D., Stamou, G.B.: Goal-oriented query rewriting for OWL 2 QL. In: Proc. of the 24th International Workshop on Description Logics (DL) (2011)
69. Clark, J., DeRose, S.: XML path language (XPath) version 1.0. W3C Recommendation, World Wide Web Consortium (1999)
70. Colucci, S., Noia, T.D., Sciascio, E.D., Donini, F.M., Ragone, A.: Second-order description logics: Semantics, motivation, and a calculus. In: Proc. of the 23rd Int. Workshop on Description Logic (DL 2010). CEUR Workshop Proceedings, vol. 573. CEUR-WS.org (2010)
71. De Giacomo, G., Lenzerini, M.: Boosting the correspondence between description logics and propositional dynamic logics. In: Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI'94). pp. 205–212 (1994)
72. De Giacomo, G., Lenzerini, M., Rosati, R.: Higher-order description logics for domain metamodeling. In: Proc. of the 25th AAAI Conference on Artificial Intelligence (AAAI 2011) (2011)
73. Du, J., Qi, G., Shen, Y.D.: Weight-based consistent query answering over inconsistent $\mathcal{SHIQ}$ knowledge bases. Knowledge and Information Systems 34(2), 335–371 (2013)
74. Du, J., Wang, K., Shen, Y.: A tractable approach to abox abduction over description logic ontologies. In: Proc. of the 28th AAAI Conference on Artificial Intelligence (AAAI 2014) (2014)
75. Du, J., Wang, K., Shen, Y.: Towards tractable and practical ABox abduction over inconsistent description logic ontologies. In: Proc. of the 29th AAAI Conference on Artificial Intelligence (AAAI 2015) (2015)
76. Ebbinghaus, H.D., Flum, J.: Finite Model Theory. Springer, second edition edn. (1999)
77. Ehrenfeucht, A., Zeiger, P.: Complexity measures for regular expressions. In: Proc. of the Sixth Annual ACM Symposium on Theory of Computing (STOC 1974) (1974)
78. Eiter, T., Gottlob, G., Ortiz, M., Šimkus, M.: Query answering in the description logic Horn-$\mathcal{SHIQ}$. In: Proc. of the 11th Eur. Conference on Logics in Artificial Intelligence (JELIA 2008). pp. 166–179. Springer, Berlin, Heidelberg (2008)
79. Eiter, T., Lutz, C., Ortiz, M., Šimkus, M.: Query answering in description logics: The knots approach. In: Logic, Language, Information and Computation, 16th International Workshop, WoLLIC 2009. Lecture Notes in Computer Science, vol. 5514, pp. 26–36. Springer (2009)
80. Eiter, T., Lutz, C., Ortiz, M., Šimkus, M.: Query answering in description logics with transitive roles. In: Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI 2009). pp. 759–764 (2009)
81. Eiter, T., Ortiz, M., Simkus, M., Tran, T., Xiao, G.: Query rewriting for Horn-SHIQ plus rules. In: Proc. of the 26th AAAI Conference on Artificial Intelligence (AAAI 2012). AAAI Press (2012)
82. Eiter, T., Ortiz, M., Šimkus, M.: Conjunctive query answering in the description logic SH using knots. J. Comput. Syst. Sci. 78(1), 47–85 (2012)
83. Feier, C., Carral, D., Stefanoni, G., Grau, B.C., Horrocks, I.: The combined approach to query answering beyond the OWL 2 profiles. In: Proc. of the 24th Int. Joint Conf. on Artificial Intelligence (IJCAI 2015) (2015)

84. Florescu, D., Levy, A., Suciu, D.: Query containment for conjunctive queries with regular expressions. In: Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98). pp. 139–148 (1998)
85. Franconi, E., Guagliardo, P., Trevisan, M., Tessaris, S.: Quelo: an ontology-driven query interface. In: Proc. of the 24th International Workshop on Description Logics (DL) (2011)
86. Franconi, E., Ibáñez-García, Y.A., Seylan, I.: Query answering with DBoxes is hard. Electr. Notes Theor. Comput. Sci. 278, 71–84 (2011)
87. Gabbay, D., Kurusz, A., Wolter, F., Zakharyaschev, M.: Many-dimensional Modal Logics: Theory and Applications. Elsevier Science Publishers (2003)
88. Glimm, B.: Using SPARQL with RDFS and OWL entailment. In: Reasoning Web. Semantic Technologies for the Web of Data, Lecture Notes in Computer Science, vol. 6848, pp. 137–201. Springer Berlin Heidelberg (2011)
89. Glimm, B., Horrocks, I., Lutz, C., Sattler, U.: Conjunctive query answering for the description logic $\mathcal{SHIQ}$. J. of Artificial Intelligence Research 31, 151–198 (2008)
90. Glimm, B., Horrocks, I., Sattler, U.: Unions of conjunctive queries in SHOQ. In: Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2008). pp. 252–262. AAAI Press/The MIT Press (2008)
91. Glimm, B., Kazakov, Y., Lutz, C.: Status QIO: An update. In: Proc. of the 22nd Int. Workshop on Description Logic (DL 2009). CEUR Workshop Proceedings, vol. 745 (2011)
92. Glimm, B., Rudolph, S., Völker, J.: Integrated metamodeling and diagnosis in OWL 2. In: Proc. of the 9th International Semantic Web Conference on The Semantic Web - Volume Part I. pp. 257–272. ISWC'10, Springer, Berlin, Heidelberg (2010)
93. Gottlob, G., Kikot, S., Kontchakov, R., Podolskii, V., Schwentick, T., Zakharyaschev, M.: The price of query rewriting in ontology-based data access. Artificial Intelligence 213, 42–59 (2014)
94. Gottlob, G., Manna, M., Pieris, A.: Polynomial combined rewritings for existential rules. In: Proc. of the 14th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2014) (2014)
95. Gottlob, G., Orsi, G., Pieris, A.: Ontological queries: Rewriting and optimization. In: (ICDE), IEEE 27th Int. Conf. on Data Engineering 2011. pp. 2 –13 (april 2011)
96. Gottlob, G., Orsi, G., Pieris, A., Šimkus, M.: Datalog and its extensions for the semantic web. In: Reasoning Web, 8th International Summer School 2012. Springer (2012)
97. Gottlob, G., Rudolph, S., Šimkus, M.: Expressiveness of guarded existential rule languages. In: Proc. of the 33rd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2014). pp. 27–38. ACM, New York, NY, USA (2014)
98. Gottlob, G., Schwentick, T.: Rewriting ontological queries into small nonrecursive datalog programs. In: Rosati, R., Rudolph, S., Zakharyaschev, M. (eds.) Description Logics. CEUR Workshop Proceedings, vol. 745. CEUR-WS.org (2011)
99. Grahne, G., Thomo, A.: Query containment and rewriting using views for regular path queries under constraints. In: Proc. of the 22nd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2003). pp. 111–122 (2003)
100. Grau, B.C., Horrocks, I., Kazakov, Y., Sattler, U.: Modular reuse of ontologies: Theory and practice. Journal of Artificial Intelligence Research (JAIR) 31, 273–318 (2008)

101. Gutiérrez-Basulto, V., Ibáñez-García, Y.A., Kontchakov, R., Kostylev, E.V.: Conjunctive queries with negation over DL-Lite: A closer look. In: Proc. of RR 2013. pp. 109–122 (2013)
102. Gutiérrez-Basulto, V., Jung, J.C., Schneider, T.: Lightweight description logics and branching time: A troublesome marriage. In: Proc. of the 14th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2014) (2014)
103. Gutierrez-Basulto, V., Jung, J.C., Schneider, T.: Lightweight temporal description logics with rigid roles and restricted tboxes. In: Proc. of the 24th Int. Joint Conf. on Artificial Intelligence (IJCAI 2015) (2015)
104. Gutiérrez-Basulto, V., Klarman, S.: Towards a unifying approach to representing and querying temporal data in description logics. In: Proc. of RR 2012. pp. 90–105 (2012)
105. Halpern, J.Y., Shoham, Y.: A propositional modal logic of time intervals. J. of the ACM 38, 935–962 (1991)
106. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language. W3C Recommendation (2013), available at `http://www.w3.org/TR/sparql11-query/`
107. Horridge, M., Bail, S., Parsia, B., Sattler, U.: The cognitive complexity of OWL justifications. In: Proc. of ISWC (2011)
108. Horridge, M., Parsia, B., Sattler, U.: Extracting justifications from bioportal ontologies. In: Proc. of ISWC (2012)
109. Horrocks, I., Kutz, O., Sattler, U.: The irresistible $\mathcal{SRIQ}$. In: Proc. of the 1st Int. Workshop on OWL: Experiences and Directions (OWLED 2005) (2005)
110. Horrocks, I., Tessaris, S.: A conjunctive query language for description logic ABoxes. In: Proc. of the 17th Nat. Conf. on Artificial Intelligence (AAAI 2000). pp. 399–404 (2000)
111. Hustadt, U., Motik, B., Sattler, U.: Data complexity of reasoning in very expressive description logics. In: Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005). pp. 466–471 (2005)
112. Imielinski, T., Jr., W.L.: Incomplete information in relational databases. J. of the ACM 31(4), 761–791 (1984)
113. Immerman, N.: Relational queries computable in polynomial time. Information and Control 68, 86–104 (1986)
114. Kalyanpur, A., Parsia, B., Sirin, E., Hendler, J.A.: Debugging unsatisfiable classes in OWL ontologies. J. Web Sem. 3(4), 268–293 (2005)
115. Kaminski, M., Grau, B.C.: Computing Horn rewritings of description logics ontologies. In: Proc. of the 24th Int. Joint Conf. on Artificial Intelligence (IJCAI 2015) (2015), `http://arxiv.org/abs/1504.05150`
116. Kaminski, M., Nenov, Y., Grau, B.C.: Computing datalog rewritings for disjunctive datalog programs and description logic ontologies. In: Proc. of the 8th Int. Conf. on Web Reasoning and Rule Systems (RR). pp. 76–91 (2014)
117. Kazakov, Y.: Consequence-driven reasoning for horn SHIQ ontologies. In: Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI 2009). pp. 2040–2045 (2009)
118. Kikot, S., Kontchakov, R., Zakharyaschev, M.: On (In)Tractability of OBDA with OWL 2 QL. In: Proc. of the 24th Int. Workshop on Description Logic (DL 2011). CEUR Workshop Proceedings, vol. 745. CEUR-WS.org (2011)
119. Kikot, S., Kontchakov, R., Zakharyaschev, M.: Conjunctive query answering with OWL 2 QL. In: Proc. of the 13th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2012). pp. 275–285. AAAI Press (2012)

120. Kikot, S., Kontchakov, R., Podolskii, V.V., Zakharyaschev, M.: Exponential lower bounds and separation for query rewriting. In: Proc. of the 39th Int. Colloquium on Automata, Languages, and Programming (ICALP 2012), Part II. Lecture Notes in Computer Science, vol. 7392, pp. 263–274. Springer (2012)
121. Kikot, S., Kontchakov, R., Podolskii, V.V., Zakharyaschev, M.: On the succinctness of query rewriting over OWL 2 QL ontologies with shallow chases. In: Proc. of the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2014). ACM Press (2014)
122. Klenke, T.: Über die Entscheidbarkeit von Konjunktiv Anfragen mit Ungleichheit in der Beschreibungslogik $\mathcal{EL}$. Master's thesis, Universität Bremen (2010)
123. Konev, B., Kontchakov, R., Ludwig, M., Schneider, T., Wolter, F., Zakharyaschev, M.: Conjunctive query inseparability of OWL 2 QL TBoxes. In: Proc. of the 29th AAAI Conference on Artificial Intelligence (AAAI 2015) (2011)
124. König, M., Leclère, M., Mugnier, M.L., Thomazo, M.: A sound and complete backward chaining algorithm for existential rules. In: Proc. of the 6th Int. Conf. on Web Reasoning and Rule Systems (RR 2012). Lecture Notes in Computer Science, vol. 7497, pp. 122–138. Springer (2012)
125. König, M., Leclère, M., Mugnier, M.L.: Query rewriting for existential rules with compiled preorder. In: Proc. of the 24th Int. Joint Conf. on Artificial Intelligence (IJCAI 2015) (2015)
126. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: The combined approach to ontology-based data access. In: Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI 2011). pp. 2656–2661. IJCAI/AAAI (2011)
127. Kontchakov, R., Pulina, L., Sattler, U., Schneider, T., Selmer, P., Wolter, F., Zakharyaschev, M.: Minimal module extraction from DL-Lite ontologies using QBF solvers. In: Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI 2009). pp. 836–841 (2009)
128. Kontchakov, R., Rodriguez-Muro, M., Zakharyaschev, M.: Ontology-based data access with databases: A short course. In: Reasoning Web. Semantic Technologies for Intelligent Data Access - 9th International Summer School 2013, Mannheim, Germany, July 30 - August 2, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8067, pp. 194–229. Springer (2013)
129. Kontchakov, R., Wolter, F., Zakharyaschev, M.: Logic-based ontology comparison and module extraction, with an application to DL-Lite. Artificial Intelligence 174(15), 1093–1141 (2010)
130. Kostylev, E.V., Grau, B.C.: On the semantics of SPARQL queries with optional matching under entailment regimes. In: The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part II. Lecture Notes in Computer Science, vol. 8797, pp. 374–389. Springer (2014)
131. Kostylev, E.V., Reutter, J.L.: Answering counting aggregate queries over ontologies of the DL-Lite family. In: Proc. of the 27th AAAI Conference on Artificial Intelligence (AAAI 2013) (2013)
132. Kostylev, E.V., Reutter, J.L., Vrgoc, D.: XPath for DL ontologies. In: Proc. of the 29th AAAI Conference on Artificial Intelligence (AAAI 2015) (2015)
133. Krisnadhi, A., Lutz, C.: Data complexity in the $\mathcal{EL}$ family of description logics. In: Logic for Programming, Artificial Intelligence, and Reasoning, 14th Int. Conf. , LPAR 2007, Proceedings. Lecture Notes in Computer Science, vol. 4790, pp. 333–347. Springer (2007)
134. Krötzsch, M., Rudolph, S.: Conjunctive queries for $\mathcal{EL}$ with composition of roles. In: Proc. of the 20th Int. Workshop on Description Logic (DL 2007) (2007)

135. Krötzsch, M., Rudolph, S., Hitzler, P.: Conjunctive queries for a tractable fragment of OWL 1.1. In: The Semantic Web, 6th Int. Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007. Lecture Notes in Computer Science, vol. 4825, pp. 310–323. Springer (2007)
136. Krötzsch, M., Rudolph, S., Hitzler, P.: Complexities of Horn description logics. ACM Trans. Comp. Log. 14(1), 2:1–2:36 (2013)
137. Krotzsch, M., Simancik, F., Horrocks, I.: Description logics. IEEE Intelligent Systems 29(1), 12–19 (2014)
138. Lembo, D., Lenzerini, M., Rosati, R., Ruzzi, M., Savo, D.F.: Inconsistency-tolerant semantics for description logics. In: Proc. of RR 2010 (2010)
139. Lembo, D., Lenzerini, M., Rosati, R., Ruzzi, M., Savo, D.F.: Query rewriting for inconsistent DL-Lite ontologies. In: Rudolph, S., Gutierrez, C. (eds.) Proc. of RR. Lecture Notes in Computer Science, vol. 6902. Springer (2011)
140. Lembo, D., Mora, J., Rosati, R., Savo, D.F., Thorstensen, E.: Towards mapping analysis in ontology-based data access. In: Proc. of the 8th Int. Conf. on Web Reasoning and Rule Systems (RR). pp. 108–123 (2014)
141. Lembo, D., Mora, J., Rosati, R., Savo, D.F., Thorstensen, E.: Mapping analysis in ontology-based data access: Algorithms and complexity. In: Proc. of the 29th Int. Workshop on Description Logic (DL 2015) (2015)
142. Lenzerini, M.: Data integration: A theoretical perspective. In: Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002). pp. 233–246 (2002)
143. Lenzerini, M., Lepore, L., Poggi, A.: Making metaquerying practical for Hi(DL-LiteR) Knowledge Bases. In: Proc. of OTM 2014. pp. 580–596 (2014)
144. Levy, A.Y., Rousset, M.C.: Combining Horn rules and description logics in CARIN. Artificial Intelligence 104(1–2), 165–209 (1998)
145. Libkin, L.: Elements of Finite Model Theory. Springer (2004)
146. Lutz, C., Toman, D., Wolter, F.: Conjunctive query answering in the description logic $\mathcal{EL}$ using a relational database system. In: Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI 2009). pp. 2070–2075. AAAI Press (2009)
147. Lutz, C.: The complexity of conjunctive query answering in expressive description logics. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings. Lecture Notes in Computer Science, vol. 5195, pp. 179–193. Springer (2008)
148. Lutz, C.: Two upper bounds for conjunctive query answering in SHIQ. In: Proc. of the 22st Int. Workshop on Description Logic (DL 2008). CEUR Workshop Proceedings, vol. 353. CEUR-WS.org (2008)
149. Lutz, C., Seylan, I., Toman, D., Wolter, F.: The combined approach to OBDA: taming role hierarchies using filters. In: The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I. Lecture Notes in Computer Science, vol. 8218, pp. 314–330. Springer (2013)
150. Lutz, C., Seylan, I., Wolter, F.: Ontology-based data access with closed predicates is inherently intractable(sometimes). In: Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI 2013). IJCAI/AAAI (2013)
151. Lutz, C., Seylan, I., Wolter, F.: Ontology-mediated queries with closed predicates. In: Proc. of the 24th Int. Joint Conf. on Artificial Intelligence (IJCAI 2015) (2015)
152. Lutz, C., Toman, D., Wolter, F.: Conjunctive query answering in the description logic $\mathcal{EL}$ using a relational database system. In: Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI 2009). pp. 2070–2075 (2009)

153. Lutz, C., Wolter, F.: Deciding inseparability and conservative extensions in the description logic $\mathcal{EL}$. Journal of Symbolic Computation 45(2), 194–228 (2010)
154. Lutz, C., Wolter, F., Zakharyaschev, M.: Temporal description logics: A survey. In: Proc. 15th International Symposium on Temporal Representation and Reasoning (TIME 2008). pp. 3–14. IEEE Computer Society (2008)
155. McGuinness, D.L., Borgida, A.: Explaining subsumption in description logics. In: Proc. of the 14th Int. Joint Conf. on Artificial Intelligence (IJCAI 1995) (1995)
156. Mora, J., Corcho, Ó.: Engineering optimisations in query rewriting for OBDA. In: Proc. of the 9th Int. Conf. on Semantic Systems (I-SEMANTICS). pp. 41–48 (2013)
157. Motik, B.: On the properties of metamodeling in OWL. In: In 4th Int. Semantic Web Conf. (ISWC 2005). pp. 548–562 (2005)
158. Motik, B.: Reasoning in Description Logics using Resolution and Deductive Databases. Ph.D. thesis, Univesität Karlsruhe (TH), Karlsruhe, Germany (January 2006)
159. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology Language Profiles. W3C Recommendation (2012), available at `http://www.w3.org/TR/owl2-profiles/`
160. Motik, B., Sattler, U., Studer, R.: Query answering for OWL-DL with rules. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) Proc. of the Third International Semantic Web Conference (ISWC). Lecture Notes in Computer Science, vol. 3298, pp. 549–563. Springer (2004)
161. Mugnier, M., Thomazo, M.: An introduction to ontology-based query answering with existential rules. In: Reasoning Web. Reasoning on the Web in the Big Data Era - 10th International Summer School 2014, Athens, Greece, September 8-13, 2014. Proceedings. pp. 245–278 (2014)
162. Ngo, N., Ortiz, M., Šimkus, M.: The combined complexity of reasoning with closed predicates in description logics. In: Proc. of the 29th Int. Workshop on Description Logic (DL 2015) (2015)
163. Ortiz, M.: Ontology based query answering: The story so far. In: Proc. of the Seventh Alberto Mendelzon International Workshop on Foundations of Data Management (AMW 2013) (2013)
164. Ortiz, M., Calvanese, D., Eiter, T.: Data complexity of query answering in expressive description logics via tableaux. J. of Automated Reasoning 41(1), 61–98 (2008)
165. Ortiz, M., Rudolph, S., Šimkus, M.: Query answering is undecidable in DLs with regular expressions, inverses, nominals, and counting. Tech. Rep. INFSYS RR-1843-10-03, Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria (Apr 2010)
166. Ortiz, M., Rudolph, S., Šimkus, M.: Query answering in the Horn fragments of the description logics SHOIQ and SROIQ. In: Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI 2011). pp. 1039–1044. IJCAI/AAAI (2011)
167. Ortiz, M., Šimkus, M., Eiter, T.: Worst-case optimal conjunctive query answering for an expressive description logic without inverses. In: Proc. of the 23rd AAAI Conf. on Artificial Intelligence (AAAI 2008). pp. 504–510. AAAI Press (2008)
168. Ortiz, M., Šimkus, M.: Reasoning and query answering in description logics. In: Proc. of the Eighth Reasoning Web Summer School (RW 2012) (2012)
169. Ortiz, M., Šimkus, M.: Revisiting the hardness of query answering in expressive description logics. In: Proc. of the Eighth Int. Conf. on Web Reasoning and Rule Systems (RR 2014) (2014)

170. OWL Working Group, W.: OWL 2 Web Ontology Language: Document Overview. W3C Recommendation (27 October 2009), available at http://www.w3.org/TR/owl2-overview/
171. Pan, J.Z., Horrocks, I.: OWL FA: A metamodeling extension of OWL DL. In: Proc. of the 15th Int. Conf. on World Wide Web. pp. 1065–1066. WWW '06, ACM, New York, NY, USA (2006)
172. Papadimitriou, C.H.: Computational Complexity. Addison Wesley Publ. Co. (1994)
173. Peñaloza, R., Sertkaya, B.: Complexity of axiom pinpointing in the DL-Lite family of description logics. In: Proc. of ECAI (2010)
174. Pérez, J., Arenas, M., Gutierrez, C.: nSPARQL: A navigational language for RDF. J. of Web Semantics 8(4), 255–270 (2010)
175. Pérez-Urbina, H., Horrocks, I., Motik, B.: Efficient query answering for OWL 2. In: Proc. of ISWC (2009)
176. Pérez-Urbina, H., Motik, B., Horrocks, I.: Tractable query answering and rewriting under description logic constraints. J. Applied Logic 8(2), 186–209 (2010)
177. Pinto, F.D., Lembo, D., Lenzerini, M., Mancini, R., Poggi, A., Rosati, R., Ruzzi, M., Savo, D.F.: Optimizing query rewriting in ontology-based data access. In: Proc. of the 16th Int. Conf. on Extending Database Technology (EDBT). pp. 561–572 (2013)
178. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. J. on Data Semantics 10, 133–173 (2008)
179. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C Recommendation (2008), http://www.w3.org/TR/rdf-sparql-query/
180. Reutter, J., Romero, M., Vardi, M.Y.: Regular queries on graph databases. In: Proc. of ICDT'15 (2015)
181. Rodríguez-Muro, M., Kontchakov, R., Zakharyaschev, M.: Ontology-based data access: Ontop of databases. In: Proc. of the 12th Int. Semantic Web Conf. (ISWC 2013). Lecture Notes in Computer Science, vol. 8218, pp. 558–573. Springer (2013)
182. Rodriguez-Muro, M., Calvanese, D.: High performance query answering over DL-Lite ontologies. In: Brewka, G., Eiter, T., McIlraith, S.A. (eds.) Proc. of the 13th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2012). AAAI Press (2012)
183. Rosati, R.: Prexto: Query rewriting under extensional constraints in DL-Lite. In: Proc. of the 9th Extended Semantic Web Conf. (EWSC 2012). Lecture Notes in Computer Science, vol. 7295, pp. 360–374. Springer (2012)
184. Rosati, R.: DL+log: Tight integration of description logics and disjunctive datalog. In: Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006). pp. 68–98 (2006)
185. Rosati, R.: The limits of querying ontologies. In: Proc. of the 11th Int. Conf. Database Theory (ICDT). pp. 164–178 (2007)
186. Rosati, R.: On conjunctive query answering in $\mathcal{EL}$. In: Proc. of the 20th Int. Workshop on Description Logic (DL 2007) (2007)
187. Rosati, R.: On the complexity of dealing with inconsistency in description logic ontologies. In: Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI 2011) (2011)
188. Rosati, R., Almatelli, A.: Improving query answering over DL-Lite ontologies. In: Proc. of the 12th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2010) (2010)

189. Rosati, R., Ruzzi, M., Graziosi, M., Masotti, G.: Evaluation of techniques for inconsistency handling in OWL 2 QL ontologies. In: Proc. of ISWC (2012)
190. Rudolph, S.: Foundations of description logics. In: Proc. of the Seventh International Reasoning Web Summer School (RW 2011). pp. 76–136 (2011)
191. Rudolph, S., Glimm, B.: Nominals, inverses, counting, and conjunctive queries or: Why infinity is your friend! J. of Artificial Intelligence Research 39, 429–481 (2010)
192. Rudolph, S., Krötzsch, M.: Flag & check: Data access with monadically defined queries. In: Proc. of the 32nd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2013). pp. 151–162. ACM (2013)
193. Schaerf, A.: Reasoning with individuals in concept languages. In: Proc. of the 3rd Conf. of the Italian Assoc. for Artificial Intelligence (AI*IA'93). Lecture Notes in Artificial Intelligence, Springer (1993)
194. Schild, K.: Combining terminological logics with tense logic. In: Proc. of the 6th Portuguese Conf. on Artificial Intelligence (EPIA'93). Lecture Notes in Computer Science, vol. 727, pp. 105–120. Springer (1993)
195. Schlobach, S., Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. In: Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003) (2003)
196. Sebastiani, R., Vescovi, M.: Axiom pinpointing in lightweight description logics via horn-sat encoding and conflict analysis. In: Proc. of CADE (2009)
197. Seylan, I., Franconi, E., de Bruijn, J.: Effective query rewriting with ontologies over DBoxes. In: Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI 2009). pp. 923–925 (2009)
198. Shmueli, O.: Decidability and expressiveness aspects of logic queries. In: Proc. of the 6th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'87). pp. 237–249 (1987)
199. Sioutos, N., de Coronado, S., Haber, M., Hartel, F., Shaiu, W., Wright, L.: NCI thesaurus: a semantic model integrating cancer-related clinical and molecular information. Journal of Biomedical Informatics 40(1), 30–43 (2006)
200. Soylu, A., Kharlamov, E., Zheleznyakov, D., Jiménez-Ruiz, E., Giese, M., Horrocks, I.: OptiqueVQS: Visual query formulation for OBDA. In: Informal Proc. of the 27th International Workshop on Description Logics (DL). pp. 725–728 (2014)
201. Stefanoni, G., Motik, B.: Answering conjunctive queries over EL knowledge bases with transitive and reflexive roles. In: Proc. of the 29th AAAI Conference on Artificial Intelligence (AAAI 2015). pp. 1611–1617. AAAI Press (2015)
202. Stefanoni, G., Motik, B., Horrocks, I.: Introducing nominals to the combined query answering approaches for EL. In: Proc. of the 22nd AAAI Conf. on Artificial Intelligence (AAAI 2007). AAAI Press (2013)
203. Stefanoni, G., Motik, B., Krötzsch, M., Rudolph, S.: The complexity of answering conjunctive and navigational queries over OWL 2 EL knowledge bases. J. Artif. Intell. Res. (JAIR) 51, 645–705 (2014)
204. Stuckenschmidt, H., Parent, C., Spaccapietra, S. (eds.): Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization, Lecture Notes in Computer Science, vol. 5445. Springer (2009)
205. The Gene Ontology Consortium: Gene ontology: Tool for the unification of biology. Nature Genetics 25, 25–29 (2000)
206. Thomazo, M.: Compact rewritings for existential rules. In: Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI 2013) (2013)
207. Trivela, D., Stoilos, G., Chortaras, A., Stamou, G.: Optimising resolution-based rewriting algorithms for owl ontologies. J. of Web Semantics (2015), to appear

208. Trivela, D., Stoilos, G., Chortaras, A., Stamou, G.: Query rewriting in Horn-$\mathcal{SHIQ}$ (extended abstract). In: Proc. of the 29th Int. Workshop on Description Logic (DL 2015) (2015)
209. Vardi, M.Y.: The complexity of relational query languages. In: Proc. of the 14th ACM SIGACT Symp. on Theory of Computing (STOC'82). pp. 137–146 (1982)
210. Vardi, M.Y.: On the complexity of bounded-variable queries. In: Proc. of the 14th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'95). pp. 266–276 (1995)
211. Venetis, T., Stoilos, G., Stamou, G.B.: Query extensions and incremental query rewriting for OWL 2 QL ontologies. J. Data Semantics 3(1), 1–23 (2014)
212. Zhou, Y., Nenov, Y., Cuenca Grau, B., Horrocks, I.: Pay-as-you-go owl query answering using a triple store. Proceedings of the 28th Conference on Artificial Intelligence (AAAI14) pp. 1142–1148 (2014)