Ontology-Based Data Access with Databases: A Short Course

Roman Kontchakov[1], Mariano Rodríguez-Muro[2] and Michael Zakharyaschev[1]

[1] Department of Computer Science and Information Systems, Birkbeck, University of London, U.K.
[2] Faculty of Computer Science, Free University of Bozen-Bolzano, Italy

Abstract. Ontology-based data access (OBDA) is regarded as a key ingredient of the new generation of information systems. In the OBDA paradigm, an ontology defines a high-level global schema of (already existing) data sources and provides a vocabulary for user queries. An OBDA system rewrites such queries and ontologies into the vocabulary of the data sources and then delegates the actual query evaluation to a suitable query answering system such as a relational database management system or a datalog engine. In this chapter, we mainly focus on OBDA with the ontology language *OWL 2 QL*, one of the three profiles of the W3C standard Web Ontology Language *OWL 2*, and relational databases, although other possible languages will also be discussed. We consider different types of conjunctive query rewriting and their succinctness, different architectures of OBDA systems, and give an overview of the OBDA system *Ontop*.

1 Introduction

Do you like movies? Do you want to know more about stars, directors, writers, casts, etc.? Then you should probably query IMDb, the Internet Movie Database available at `www.imdb.com`. You do not know how to use databases. But you have already taken the 'Introduction to Description Logics' course. Then what you need is ontology-based data access (OBDA, for short). There is a simple movie ontology MO at `www.movieontology.org` describing the application domain in terms of concepts (classes), such as *mo:Movie* and *mo:Person*, and roles and attributes (object and datatype properties), such as *mo:cast* and *mo:year*:

$$\mathit{mo{:}Movie} \equiv \exists \mathit{mo{:}title}, \qquad \mathit{mo{:}Movie} \sqsubseteq \exists \mathit{mo{:}year},$$
$$\mathit{mo{:}Movie} \equiv \exists \mathit{mo{:}cast}, \qquad \exists \mathit{mo{:}cast}^- \sqsubseteq \mathit{mo{:}Person}, \qquad \text{etc.}$$

And you can query the IMDb data in terms of concepts and roles of the MO ontology; for example,

$$\boldsymbol{q}(t, y) = \exists m \,\big(\mathit{mo{:}Movie}(m) \land \mathit{mo{:}title}(m,t) \land \mathit{mo{:}year}(m,y) \land (y > 2010)\big)$$

is a conjunctive query asking for the titles (the variable t) of recent movies with their production year (the variable y). An OBDA system such as *Ontop* available at `ontop.inf.unibz.it` will automatically rewrite your query into the language of IMDb, optimise the rewriting and use a conventional relational database management system (RDBMS) to find the answers. (We will return to the IMDb example in Section 6.)

The idea of OBDA was explicitly formulated in 2008 [17, 27, 56], though query answering over description logic knowledge bases has been investigated since at least 2005. Nowadays, OBDA is often deemed to be an important ingredient of the new generation of information systems as it (*i*) gives a high-level conceptual view of the data, (*ii*) provides the user with a convenient vocabulary for queries, (*iii*) allows the system to enrich incomplete data with background knowledge, and (*iv*) supports queries to multiple and possibly heterogeneous data sources.

One can distinguish between several types of OBDA depending on the expressive power of description logics (DLs).

OBDA with databases: some DLs, such as the logics of the *DL-Lite* family (and *OWL 2 QL*), allow a reduction of conjunctive queries over ontologies to first-order queries over standard relational databases [11, 56, 4, 38];

OBDA with datalog engines: other DLs encompassing logics in the $\mathcal{EL}$ family (*OWL 2 EL*), Horn-$\mathcal{SHIQ}$ and Horn-$\mathcal{SROIQ}$, support a datalog reduction and can be used with datalog engines [65, 46, 52, 18];

OBDA with expressive DLs such as $\mathcal{ALC}$ or $\mathcal{SHIQ}$ require some special techniques for answering conjunctive queries; see [28, 47, 51, 22, 19, 13, 34] and references therein for details.

In this chapter, we give a brief and easy introduction to the theory and practice of OBDA with relational databases, assuming that the reader has some basic knowledge of description logic. Our plan is as follows. In Section 2, we introduce and discuss the DLs supporting first-order rewritability of conjunctive queries. Then, in Section 3, we show how to compute first-order and nonrecursive datalog rewritings of conjunctive queries over *OWL 2 QL* ontologies. The size of rewritings is discussed in Section 4. In Section 5, we introduce the basics of the combined approach to OBDA. Finally, in Section 6, we present the OBDA system *Ontop*, which is available as a plugin for the Protégé 4 ontology editor as well as OWLAPI and Sesame libraries and a SPARQL end-point.

2 Description Logics for OBDA with Databases

The key notion of OBDA with databases is query rewriting. The user formulates a query $\boldsymbol{q}$ in the vocabulary of a given ontology $\mathcal{T}$. (Such a pair $(\mathcal{T}, \boldsymbol{q})$ is sometimes called an *ontology-mediated query.*) The task of an OBDA system is to 'rewrite' $\boldsymbol{q}$ and $\mathcal{T}$ into a new query $\boldsymbol{q}'$ in the vocabulary of the data such that, for any possible data $\mathcal{A}$ (in this vocabulary), the answers to $\boldsymbol{q}$ over $(\mathcal{T}, \mathcal{A})$ are precisely the same as the answers to $\boldsymbol{q}'$ over $\mathcal{A}$. Thus, the problem of querying data $\mathcal{A}$ (the structure of which is not known to the user) in terms of the ontology $\mathcal{T}$ (accessible to the user) is reduced to the problem of querying $\mathcal{A}$ directly. As witnessed by the 40 years history of relational databases, RDBMSs are usually very efficient in query evaluation. Other query answering systems, for example datalog engines can also be employed. In this section, we consider the ontology languages supporting query rewriting. To be more focused, we concentrate on description logics (DLs) as ontology formalisms and only provide the reader with references to languages of other types. We also assume in this section that $\mathcal{A}$ is simply a DL ABox stored in a relational database (proper databases will be considered in Section 6).

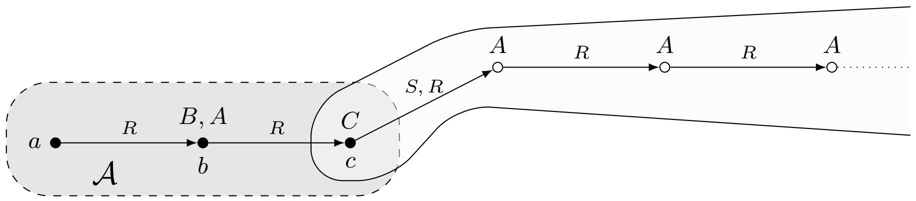
Example 1. Consider the query $\boldsymbol{q}(x) = \exists y\, \big(R(x,y) \land A(y)\big)$ that asks for the individuals x in the ABox such that $R(x,y)$ and $A(y)$, for some y (which does not have to be an ABox individual). Suppose also that we are given the DL ontology (or TBox)

$$\mathcal{T} \;=\; \{\;\; B \sqsubseteq A, \quad C \sqsubseteq \exists S, \quad \exists S^- \sqsubseteq A, \quad \exists R^- \sqsubseteq \exists R, \quad S \sqsubseteq R \;\;\},$$

where A and B are concept names and R and S are role names (thus, the second axiom says that C is a subset of the domain of S, and the third that the range of S is a subset of A). It takes a moment's thought to see that we obtain $R(x,y)$, for some y, if we have one of $R(x,y)$, $S(x,y)$ or $C(x)$ (in the last case y may not even exist among the ABox individuals). It follows from $\mathcal{T}$ that, in the second case, $A(y)$ also holds, and, in the third case, there exists some y such that $R(x,y)$ and $A(y)$. Similarly, we obtain $A(y)$ if we have one of $A(y)$, $B(y)$ or $S(z,y)$, for some z. These observations give the following first-order rewriting of $\boldsymbol{q}$ and $\mathcal{T}$:

$$\boldsymbol{q}'(x) = \exists y\, \big[R(x,y) \land \big(A(y) \lor B(y) \lor \exists z\, S(z,y)\big)\big] \lor \exists y\, S(x,y) \lor C(x).$$

Now, suppose $\mathcal{A} = \{\, R(a,b),\ B(b),\ R(b,c),\ C(c)\,\}$. It is easy to compute the answers to $\boldsymbol{q}'(x)$ over $\mathcal{A}$: these are $x = a$ and $x = c$. What about the answers to $\boldsymbol{q}$ over $(\mathcal{T},\mathcal{A})$? We can compute them by first applying the axioms of $\mathcal{T}$ to $\mathcal{A}$, always creating *fresh* witnesses for the existential quantifiers $\exists R$ and $\exists S$ if they do not already exist, and then evaluating $\boldsymbol{q}(x)$ over the resulting (possibly infinite) structure known as the *canonical model* of $(\mathcal{T},\mathcal{A})$. The canonical model is illustrated in the picture below (where the fresh witnesses are shown as $\circ$). Taking into consideration that we only need answers from the data (the individuals from $\mathcal{A}$), we see again that they are $x = a$ and $x = c$.



Formulas such as $\boldsymbol{q}(\boldsymbol{x})$ in Example 1 are called *conjunctive queries* (CQs, for short). CQs are the most basic type of database queries, also known as SELECT-PROJECT-JOIN SQL queries. More precisely, in the context of OBDA over DL ontologies, a CQ $\boldsymbol{q}(\boldsymbol{x})$ is a first-order formula $\exists \boldsymbol{y}\, \varphi(\boldsymbol{x},\boldsymbol{y})$, where φ is a conjunction of atoms of the form $A(t_1)$ or $P(t_1, t_2)$, and each t_i is a *term* (an individual or a variable in $\boldsymbol{x}$ or $\boldsymbol{y}$). The variables in $\boldsymbol{x}$ are called *answer variables* and those in $\boldsymbol{y}$ *existentially quantified variables*. Formulas such as $\boldsymbol{q}'(x)$ in Example 1 can also use disjunctions and are called *positive existential queries* (PE-queries). If all Boolean connectives (conjunction, disjunction and negation) as well as both quantifiers, $\forall$ and $\exists$, are allowed then we call the queries *first-order* (FO-queries). FO-queries (more precisely, domain-independent FO-queries) roughly correspond to the class of queries expressible in SQL. A query $\boldsymbol{q}(\boldsymbol{x})$ is called *Boolean* if $\boldsymbol{x} = \emptyset$.

A tuple $\boldsymbol{a}$ of individuals in $\mathcal{A}$ (of the same length as $\boldsymbol{x}$) is a *certain answer* to $\boldsymbol{q}(\boldsymbol{x})$ over $(\mathcal{T}, \mathcal{A})$ if $\mathcal{I} \models \boldsymbol{q}(\boldsymbol{a})$ for all models $\mathcal{I}$ of $(\mathcal{T}, \mathcal{A})$; in this case we write $(\mathcal{T}, \mathcal{A}) \models \boldsymbol{q}(\boldsymbol{a})$. In other words, $\boldsymbol{a}$ is a certain answer to $\boldsymbol{q}(\boldsymbol{x})$ over $(\mathcal{T}, \mathcal{A})$ if $\boldsymbol{q}(\boldsymbol{a})$ follows logically from $\mathcal{A}$ and $\mathcal{T}$. For a Boolean $\boldsymbol{q}$, the certain answer is 'yes' if $\boldsymbol{q}$ holds in all models of $(\mathcal{T}, \mathcal{A})$, and 'no' otherwise. Finally, an *FO-rewriting* of $\boldsymbol{q}(\boldsymbol{x})$ and $\mathcal{T}$ is an FO-query $\boldsymbol{q}'(\boldsymbol{x})$ such that $(\mathcal{T}, \mathcal{A}) \models \boldsymbol{q}(\boldsymbol{a})$ iff $\mathcal{A} \models \boldsymbol{q}'(\boldsymbol{a})$, for *any* ABox $\mathcal{A}$ and *any* tuple $\boldsymbol{a} \subseteq \mathsf{ind}(\mathcal{A})$, where $\mathsf{ind}(\mathcal{A})$ is the set of individuals in $\mathcal{A}$.

For the purposes of OBDA we are naturally interested in maximal ontology languages that ensure FO-rewritability of CQs. To distinguish between DLs with and without FO-rewritability, we require a few facts from the computational complexity theory. For details, the reader is referred to [3, 29, 40, 44]. The hierarchy of the complexity classes we use in this chapter is shown below:

$$\mathrm{AC}^0 \subsetneq \textsc{NLogSpace} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \textsc{PSpace} \subseteq \textsc{ExpTime}.$$

It is also known that $\mathrm{P} \subsetneq \textsc{ExpTime}$ and $\textsc{NLogSpace} \subsetneq \textsc{PSpace}$ (whether the remaining inclusions are proper is a major open problem in computer science). Consider, for example, the problem of evaluating a (Boolean) FO-query over relational databases: given an ABox $\mathcal{A}$ and a query $\boldsymbol{q}$, decide whether $\mathcal{A} \models \boldsymbol{q}$. If both $\mathcal{A}$ and $\boldsymbol{q}$ are taken as an input, then the problem is PSpace-complete for FO-queries (due to alternation of quantifiers), and NP-complete for CQs and PE-queries. In this case, we refer to the *combined complexity* of query answering. If only the ABox is an input (and the query is fixed), the problem is in AC^0 in *data complexity* (this is regarded as an appropriate measure in databases because the database instance is much smaller than the query).

Returning to FO-rewritability of CQs $\boldsymbol{q}$ and DL TBoxes $\mathcal{T}$, we observe that if the problem '$(\mathcal{T}, \mathcal{A}) \models \boldsymbol{q}$?' is at least NLogSpace-hard for data complexity (that is, with fixed $\mathcal{T}$ and $\boldsymbol{q}$), then $\boldsymbol{q}$ and $\mathcal{T}$ cannot be FO-rewritable. Indeed, if there was an FO-rewriting then this problem could be solved in AC^0. This observation allows us to delimit the DL constructs that ruin FO-rewritability.

Example 2. A typical example of an NLogSpace-complete problem is the *reachability problem* for directed graphs: given a directed graph $G = (V, R)$ with vertices V and arcs R and two distinguished vertices $s, t \in V$, decide whether there is a directed path from s to t in G. We represent the input by means of the ABox

$$\mathcal{A}_{G,s,t} \quad = \quad \{\, R(v_1, v_2) \mid (v_1, v_2) \in R \,\} \quad \cup \quad \{A(s),\, B(t)\}.$$

Consider now the following TBox and Boolean CQ

$$\mathcal{T} \;=\; \{\, \exists R.B \sqsubseteq B \,\}, \qquad \boldsymbol{q} = \exists y\, (A(y) \land B(y)).$$

It is readily seen that $(\mathcal{T}, \mathcal{A}_{G,s,t}) \models \boldsymbol{q}$ iff there is a path from s to t in G. As $\mathcal{T}$ and $\boldsymbol{q}$ do not depend on G, s and t, the problem '$(\mathcal{T}, \mathcal{A}_{G,s,t}) \models \boldsymbol{q}$?' is NLogSpace-hard for data complexity, and so $\boldsymbol{q}$ and $\mathcal{T}$ cannot be FO-rewritable.

In other words, TBoxes capable of computing the transitive closure of some relations in ABoxes do not allow FO-rewritability.

Example 3. The *path system accessibility* problem is an example of a P-complete problem [21]: given a finite set V of vertices and a relation $E \subseteq V \times V \times V$ with a set of source vertices $S \subseteq V$ and a terminal vertex $t \in V$, decide whether t is accessible, where all $v \in S$ are accessible, and if $(v_1, v_2, v) \in E$, for accessible inputs v_1 and v_2, then v is also accessible. The path system can be encoded by an ABox $\mathcal{A}$ in the following way:

$$\{A(v) \mid v \in S\} \;\cup\; \{\; P_1(e, v_1),\; P_2(e, v_2),\; R(v, e) \;\mid\; e = (v_1, v_2, v) \in E \;\}.$$

Consider now the TBox $\mathcal{T}$ and Boolean CQ $\boldsymbol{q}$ given by

$$\mathcal{T} \;=\; \{\; \exists P_1.A \sqcap \exists P_2.A \sqsubseteq B,\;\; \exists R.B \sqsubseteq A \;\}, \qquad \boldsymbol{q} = A(t).$$

It should be clear that $(\mathcal{T}, \mathcal{A}) \models A(v)$ iff v is accessible (and that $(\mathcal{T}, \mathcal{A}) \models B(e)$ iff *both* inputs of e are accessible, that is, both belong to A). Therefore, the answer to $\boldsymbol{q}$ is 'yes' iff t is accessible. Thus, the problem '$(\mathcal{T}, \mathcal{A}) \models \boldsymbol{q}$' is P-hard for data complexity, and so $\boldsymbol{q}$ and $\mathcal{T}$ cannot be FO-rewritable.

Note that the TBox $\mathcal{T}$ in Example 3 is formulated in the DL $\mathcal{EL}$.

Example 4. A good example of an NP-complete problem is *graph 3-colouring*: given an (undirected) graph $G = (V, E)$, decide whether each of its vertices can be painted in one of three given colours in such a way that no adjacent vertices have the same colour. We represent the input graph G by means of the ABox

$$\mathcal{A}_G \;=\; \{\, R(v_1, v_2) \mid \{v_1, v_2\} \in E \,\}.$$

Consider now the Boolean CQ $\boldsymbol{q} = \exists y\, N(y)$ and the following TBox

$$\mathcal{T} \;=\; \{\, \top \sqsubseteq C_1 \sqcup C_2 \sqcup C_3 \,\} \;\cup\; \{\, C_i \sqcap C_j \sqsubseteq N \mid 1 \le i < j \le 3 \,\} \;\cup$$
$$\{\, C_i \sqcap \exists R.C_i \sqsubseteq N \mid 1 \le i \le 3 \,\},$$

where the C_i are concept names representing the given three colours. It is not hard to see that the answer to $\boldsymbol{q}$ over $(\mathcal{T}, \mathcal{A}_G)$ is 'no' iff G is 3-colourable. It follows that the problem '$(\mathcal{T}, \mathcal{A}_G) \models \boldsymbol{q}$?' is coNP-hard for data complexity, and so $\boldsymbol{q}$ and $\mathcal{T}$ are not FO-rewritable.

The moral of this example is that to allow FO-rewritability, TBoxes should not contain axioms with disjunctive information (which can be satisfied in essentially different ways when applied to ABoxes). The TBox in Example 4 is formulated in the DL $\mathcal{ALC}$.

The data complexity of answering CQs over ontologies formulated in various DLs has been intensively investigated since 2005; see, e.g., [12, 41, 11, 51, 4]. Thus, answering CQs over $\mathcal{EL}$ ontologies is P-complete for data complexity, while for $\mathcal{ALC}$ it is coNP-hard. One of the results of this research was the inclusion in the current W3C standard Web Ontology Language *OWL 2* of a special sublanguage (or profile) that is suitable for OBDA with databases and called *OWL 2 QL*. The DLs underlying *OWL 2 QL* belong to the so-called *DL-Lite* family [11, 4]. Below, we present *OWL 2 QL* in the DL parlance rather than the *OWL 2* syntax.

The language of *OWL 2 QL* contains *individual names* a_i, *concept names* A_i, and *role names* P_i ($i = 1, 2, \dots$). *Roles* R, *basic concepts* B and *concepts* C are defined by the grammar:

$$\begin{array}{lcl} R & ::= & P_i \quad | \quad P_i^-, \\ B & ::= & \bot \quad | \quad A_i \quad | \quad \exists R, \\ C & ::= & B \quad | \quad \exists R.B \end{array}$$

(here P_i^- is the inverse of P_i and $\exists R$ is regarded as an abbreviation for $\exists R.\top$). An *OWL 2 QL TBox*, $\mathcal{T}$, is a finite set of *concept* and *role inclusions* of the form

$$B \sqsubseteq C, \qquad R_1 \sqsubseteq R_2$$

and *concept* and *role disjointness constraints* of the form

$$B_1 \sqcap B_2 \sqsubseteq \bot, \qquad R_1 \sqcap R_2 \sqsubseteq \bot.$$

Apart from this, $\mathcal{T}$ may contain assertions stating that certain roles P_i are reflexive and irreflexive. Note that symmetry and asymmetry of a role R can be expressed in *OWL 2 QL* as, respectively,

$$R \sqsubseteq R^- \quad \text{and} \quad R \sqcap R^- \sqsubseteq \bot.$$

An *OWL 2 QL ABox*, $\mathcal{A}$, is a finite set of *assertions* of the form $A_k(a_i)$ and $P_k(a_i, a_j)$ and *inequality constraints* $a_i \neq a_j$ for $i \neq j$. $\mathcal{T}$ and $\mathcal{A}$ together constitute the *knowledge base* (KB) $\mathcal{K} = (\mathcal{T}, \mathcal{A})$.

It is to be noted that concepts of the form $\exists R.B$ can only occur in the right-hand side of concept inclusions in *OWL 2 QL*. An inclusion $B' \sqsubseteq \exists R.B$ can be regarded as an abbreviation for three inclusions:

$$B' \sqsubseteq \exists R_B, \qquad \exists R_B^- \sqsubseteq B \quad \text{and} \quad R_B \sqsubseteq R,$$

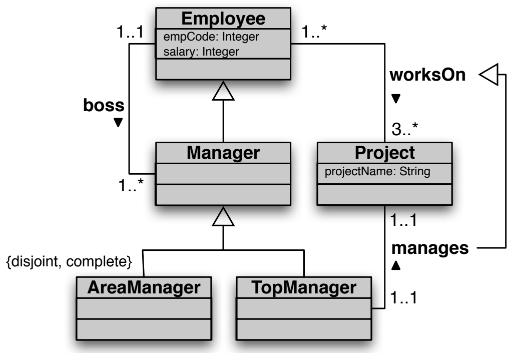
where R_B is a fresh role name. Thus, inclusions of the form $B' \sqsubseteq \exists R.B$ are just convenient syntactic sugar. To simplify presentation, in the remainder of this chapter we consider the sugar-free *OWL 2 QL*, assuming that every concept inclusion is of the form $B_1 \sqsubseteq B_2$, where both B_1 and B_2 are basic concepts.

Unlike standard DLs, *OWL 2* does not adopt the *unique name assumption* (UNA, for short) according to which, for any interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ and any distinct individual names a_i and a_j, we must have $a_i^{\mathcal{I}} \neq a_j^{\mathcal{I}}$. That is why *OWL 2 QL* has inequality constraints $a_i \neq a_j$ in its syntax.

Theorem 1 ([11, 4]). (*i*) *The satisfiability problem for OWL 2 QL knowledge bases is* NLogSpace*-complete for combined complexity and in* AC^0 *for data complexity.*

(*ii*) *CQs and OWL 2 QL TBoxes are FO-rewritable, and so CQ answering over OWL 2 QL TBoxes is in* AC^0 *for data complexity.*

We will explain how one can compute FO-rewritings for CQs over *OWL 2 QL* TBoxes in the next section. Meanwhile, we are going to illustrate the expressive power of the language and discuss whether it can be extended without losing FO-rewritability. The following example shows what can and what cannot be represented in *OWL 2 QL* TBoxes. As the primary aim of *OWL 2 QL* is to facilitate OBDA with relational databases, our example is from the area of conceptual data modelling.



Example 5. Consider the UML class diagram in the picture above representing (part of) a company information system. According to the diagram, all managers are employees and are partitioned into area managers and top managers. In *OWL 2 QL*, we can write

$$Manager \sqsubseteq Employee,$$
$$AreaManager \sqsubseteq Manager, \qquad TopManager \sqsubseteq Manager,$$
$$AreaManager \sqcap TopManager \sqsubseteq \bot.$$

However, the covering constraint $Manager \sqsubseteq AreaManager \sqcup TopManager$ uses union $\sqcup$, which is not allowed in *OWL 2 QL*. Each employee has two attributes, *empCode* and *salary*, with integer values. Unlike OWL, here we do not distinguish between abstract objects and data values (there are, however, approaches based on concrete domains [5, 66]). Hence we model a datatype, such as *Integer*, by a concept, and an attribute, such as employee's salary, by a role:

$$\mathit{Employee} \sqsubseteq \exists \mathit{salary}, \qquad \exists \mathit{salary}^- \sqsubseteq \mathit{Integer}.$$

However, the constraint $\geq 2\mathit{salary} \sqsubseteq \bot$ (saying that the attribute *salary* is functional) is not allowed. The attribute *empCode* with values in *Integer* is represented in the same way. The binary relation *worksOn* has *Employee* as its domain and *Project* as its range:

$$\exists \mathit{worksOn} \sqsubseteq \mathit{Employee}, \qquad \exists \mathit{worksOn}^- \sqsubseteq \mathit{Project}.$$

The relation *boss* with domain *Employee* and range *Manager* is treated similarly. Of the constraints that each employee works on a project and has exactly one boss, and a project must involve at least three employees, we can only capture the following:

$$\mathit{Employee} \sqsubseteq \exists \mathit{worksOn}, \qquad \mathit{Employee} \sqsubseteq \exists \mathit{boss}.$$

Both cardinality constraints $\geq 2\ \mathit{boss} \sqsubseteq \bot$ and $\mathit{Project} \sqsubseteq \geq 3\mathit{worksOn}^-$ require a more powerful language. Finally, we have to say that a top manager manages exactly one project and also works on that project, while a project is managed by exactly one top manager. In *OWL 2 QL*, we can only write:

$$\begin{array}{ll} \exists \mathit{manages} \sqsubseteq \mathit{TopManager}, & \exists \mathit{manages}^- \sqsubseteq \mathit{Project}, \\ \mathit{TopManager} \sqsubseteq \exists \mathit{manages}, & \mathit{Project} \sqsubseteq \exists \mathit{manages}^-, \\ \mathit{manages} \sqsubseteq \mathit{worksOn}, & \end{array}$$

but not $\geq 2\mathit{manages} \sqsubseteq \bot$ and $\geq 2\mathit{manages}^- \sqsubseteq \bot$. We cannot, obviously, represent constraints such as $\mathit{CEO} \sqcap (\geq 5\ \mathit{worksOn}) \sqcap \exists \mathit{manages} \sqsubseteq \bot$ (no CEO may work on five projects and be a manager of one of them) either.

As we saw in the example above, some constructs that are important for conceptual modelling are not available in *OWL 2 QL*. Can we add these constructs to the language without destroying FO-rewritability?

Let us recall from Example 2 that we cannot extend *OWL 2 QL* with the construct $\exists R.B$ in the left-hand side of concept inclusions or with transitivity constraints (stating that certain roles are interpreted by transitive relations). Example 4 shows that $\sqcup$ in the right-hand side can be dangerous. On the other hand, we can safely use concept and role inclusions with $\sqcap$ in the left-hand side:

$$B_1 \sqcap \cdots \sqcap B_n \sqsubseteq B, \qquad R_1 \sqcap \cdots \sqcap R_m \sqsubseteq R, \qquad \text{for } m, n \geq 1.$$

Unqualified number restrictions $\geq k\,R$, for $k \geq 2$, are a bit trickier. As *OWL 2 QL* does not adopt the UNA, an axiom such as $(\geq 3\,R \sqsubseteq \bot)$ over an ABox containing the atoms $R(a, b_i)$, for $i \geq 3$, means that some of the b_i must coincide, and there are various ways to make it so by identifying some of the b_i. In fact, unqualified number restrictions added to *OWL 2 QL* make it coNP-hard for data complexity; and even role functionality makes it P-hard for data complexity [4].

One can argue, however, that in the context of OBDA it is more natural and important to adopt the UNA. Indeed, after all, databases do respect the UNA. It turns out that if we stipulate that the UNA is respected in *OWL 2 QL*, then unqualified number restrictions are 'harmless' provided that we do not use both axioms ($\geq k\,R \sqsubseteq B$), for $k \geq 2$, and $R' \sqsubseteq R$ in the same TBox (consult [4] for a more precise condition).

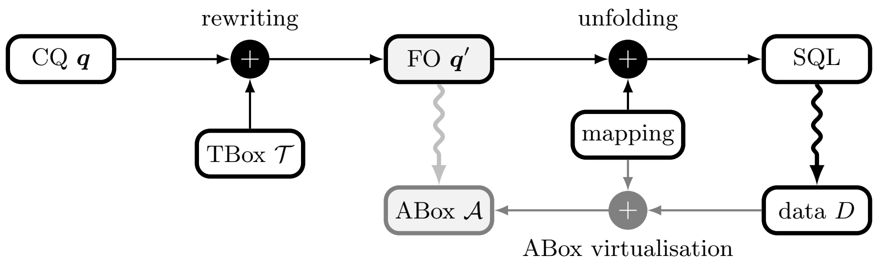
The results discussed so far guarantee that if our ontology is formulated in a DL with the help of such and such constructs then it can safely be used for OBDA with databases. A different approach to understanding the phenomenon of FO-rewritability has recently been suggested [49]: it attempts to classify all TBoxes $\mathcal{T}$ in some master DL, say $\mathcal{ALCI}$, according to the complexity of answering CQs over $\mathcal{T}$.

Finally, we note that another family of ontology languages suitable for OBDA with databases has been designed by the datalog community. We refer the reader to the recent papers [10, 9] for a survey. Connections of query answering via DL ontologies with disjunctive datalog and constraint satisfaction problems have been established in [7].

In the next section, we shall see how one can construct FO-rewritings of CQs and *OWL 2 QL* TBoxes.

3 Tree-Witness Rewriting

A standard architecture of an OBDA system over relational data sources can be represented as follows:



The user is given an *OWL 2 QL* TBox $\mathcal{T}$ and can formulate CQs $\boldsymbol{q}(\boldsymbol{x})$ in the signature of $\mathcal{T}$. The system rewrites $\boldsymbol{q}(\boldsymbol{x})$ and $\mathcal{T}$ into an FO-query $\boldsymbol{q}'(\boldsymbol{x})$ such that $(\mathcal{T}, \mathcal{A}) \models \boldsymbol{q}(\boldsymbol{a})$ iff $\mathcal{A} \models \boldsymbol{q}'(\boldsymbol{a})$, for any ABox $\mathcal{A}$ and any tuple $\boldsymbol{a}$ of individuals in $\mathcal{A}$. The rewriting $\boldsymbol{q}'$ is called a *PE-rewriting* if it is a PE-query and an *NDL-rewriting* if it is an NDL-query.

The rewriting $\boldsymbol{q}'(\boldsymbol{x})$ is formulated in the signature of $\mathcal{T}$ and has to be further transformed into the vocabulary of the data source D before being evaluated. For instance, $\boldsymbol{q}'(\boldsymbol{x})$ can be unfolded into an SQL query by means of a mapping $\mathcal{M}$ relating the signature of $\mathcal{T}$ to the vocabulary of D. We consider unfolding in Section 6, but before that we assume the data to be given as an ABox (say, as a universal table in a database or as a triple store) with a trivial mapping.

A number of different rewriting techniques have been proposed and implemented for *OWL 2 QL* (PerfectRef [56], Presto/Prexto [64, 63], Rapid [16]) and its extensions ([37], Nyaya [23], Requiem/Blackout [54, 55], Clipper [18]). In this section, we discuss the tree-witness rewriting [33].

3.1 Canonical Model

All types of FO-rewritings are based on the fact that, if an *OWL 2 QL* KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is consistent, then there is a special model of $\mathcal{K}$, called a *canonical model* and denoted $\mathcal{C}_\mathcal{K}$, such that $\mathcal{K} \models \boldsymbol{q}(\boldsymbol{a})$ iff $\mathcal{C}_\mathcal{K} \models \boldsymbol{q}(\boldsymbol{a})$, for any CQ $\boldsymbol{q}(\boldsymbol{x})$ and any $\boldsymbol{a} \subseteq \mathsf{ind}(\mathcal{A})$. We have already seen one canonical model in Example 1. Intuitively, the construction of $\mathcal{C}_\mathcal{K}$ is pretty straightforward: we start with $\mathcal{A}$ and then apply to it recursively the axioms of $\mathcal{T}$ wherever possible. The only nontrivial case is when we apply an axiom of the form $B \sqsubseteq \exists R$ to some $w \in B$. Then either we already have an R-arrow starting from w, in which case we do not have to do anything, or such an R-arrow does not exist, and then we create a *fresh* individual, say w_R, in the model under construction and draw an R-arrow from w to w_R.

Formally, let

$$[R] \quad = \quad \{\ S \quad | \quad \mathcal{T} \models R \sqsubseteq S \text{ and } \mathcal{T} \models S \sqsubseteq R\ \}.$$

We write $[R] \leq_\mathcal{T} [S]$ if $\mathcal{T} \models R \sqsubseteq S$; thus, $\leq_\mathcal{T}$ is a partial order on the set of equivalence classes $[R]$, for roles R in $\mathcal{T}$. For each $[R]$, we introduce a *witness* $w_{[R]}$ and define a *generating relation* $\leadsto_\mathcal{K}$ on the set of these witnesses together with $\mathsf{ind}(\mathcal{A})$ by taking:

$a \leadsto_\mathcal{K} w_{[R]}$ if $a \in \mathsf{ind}(\mathcal{A})$ and $[R]$ is $\leq_\mathcal{T}$-minimal such that $\mathcal{K} \models \exists R(a)$ and $\mathcal{K} \not\models R(a, b)$ for any $b \in \mathsf{ind}(\mathcal{A})$;

$w_{[S]} \leadsto_\mathcal{K} w_{[R]}$ if $[R]$ is $\leq_\mathcal{T}$-minimal with $\mathcal{T} \models \exists S^- \sqsubseteq \exists R$ and $[S^-] \neq [R]$.

A *$\mathcal{K}$-path* is a finite sequence $aw_{[R_1]} \cdots w_{[R_n]}$, $n \geq 0$, such that $a \in \mathsf{ind}(\mathcal{A})$ and, if $n > 0$, then $a \leadsto_\mathcal{K} w_{[R_1]}$ and $w_{[R_i]} \leadsto_\mathcal{K} w_{[R_{i+1}]}$, for $i < n$. Denote by $\mathsf{tail}(\sigma)$ the last element in the path σ. The canonical model $\mathcal{C}_\mathcal{K} = (\Delta^{\mathcal{C}_\mathcal{K}}, \cdot^{\mathcal{C}_\mathcal{K}})$ is defined by taking $\Delta^{\mathcal{C}_\mathcal{K}}$ to be the set of all $\mathcal{K}$-paths and setting:

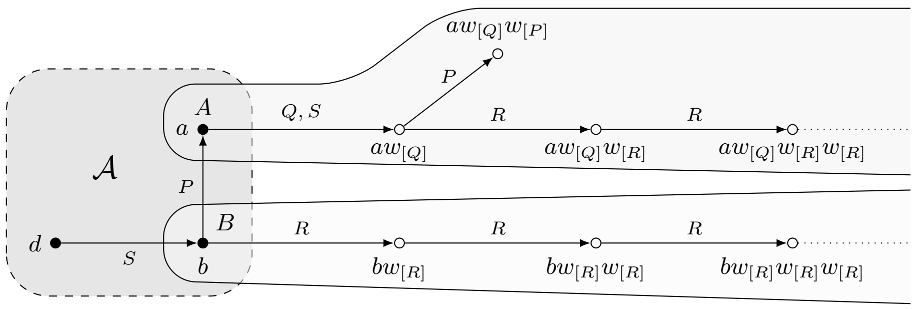
$$\begin{aligned}
a^{\mathcal{C}_\mathcal{K}} &= a, \text{ for all } a \in \mathsf{ind}(\mathcal{A}),\\
A^{\mathcal{C}_\mathcal{K}} &= \{a \in \mathsf{ind}(\mathcal{A}) \mid \mathcal{K} \models A(a)\} \cup \\
&\ \{\sigma \cdot w_{[R]} \mid \mathcal{T} \models \exists R^- \sqsubseteq A\}, \text{ for all concept names } A,\\
P^{\mathcal{C}_\mathcal{K}} &= \{(a, b) \in \mathsf{ind}(\mathcal{A}) \times \mathsf{ind}(\mathcal{A}) \mid R(a, b) \in \mathcal{A} \text{ with } [R] \leq_\mathcal{T} [P]\} \cup \\
&\ \{(\sigma, \sigma \cdot w_{[R]}) \mid \mathsf{tail}(\sigma) \leadsto_\mathcal{K} w_{[R]},\ [R] \leq_\mathcal{T} [P]\} \cup \\
&\ \{(\sigma \cdot w_{[R]}, \sigma) \mid \mathsf{tail}(\sigma) \leadsto_\mathcal{K} w_{[R]},\ [R] \leq_\mathcal{T} [P^-]\}, \text{ for all role names } P.
\end{aligned}$$

We call $\mathsf{ind}(\mathcal{A})$ the *ABox part* of $\mathcal{C}_\mathcal{K}$, and $\Delta^{\mathcal{C}_\mathcal{K}} \setminus \mathsf{ind}(\mathcal{A})$ the *anonymous part.*

Example 6. Consider the KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, where

$$\begin{aligned}
\mathcal{T} &= \{\, A \sqsubseteq \exists Q,\ \ \exists Q^- \sqsubseteq B,\ \ Q \sqsubseteq S,\ \ \exists S^- \sqsubseteq \exists R,\ \ \exists R^- \sqsubseteq \exists R,\ \ B \sqsubseteq \exists P \,\},\\
\mathcal{A} &= \{\, A(a),\ B(b),\ P(b,a),\ S(d,b) \,\}.
\end{aligned}$$

The canonical model $\mathcal{C}_\mathcal{K}$ for $\mathcal{K}$ is depicted below:



Theorem 2. *For any consistent OWL 2 QL KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, any CQ $\boldsymbol{q}(\boldsymbol{x})$ and any $\boldsymbol{a} \subseteq \mathsf{ind}(\mathcal{A})$, we have $\mathcal{K} \models \boldsymbol{q}(\boldsymbol{a})$ iff $\mathcal{C}_\mathcal{K} \models \boldsymbol{q}(\boldsymbol{a})$.*

Thus, to compute certain answers to $\boldsymbol{q}(\boldsymbol{x})$ over $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, it is enough to find answers to $\boldsymbol{q}(\boldsymbol{x})$ in the canonical model $\mathcal{C}_\mathcal{K}$. To do this, we have to check all possible homomorphisms from $\boldsymbol{q}$ to $\mathcal{C}_\mathcal{K}$ that map the answer variables $\boldsymbol{x}$ to the ABox part of $\mathcal{C}_\mathcal{K}$. More precisely, let us regard $\boldsymbol{q}(\boldsymbol{x}) = \exists \boldsymbol{y}\, \varphi(\boldsymbol{x}, \boldsymbol{y})$ as simply the set of atoms in φ, so we can write $A(x) \in \boldsymbol{q}$, $P(x,y) \in \boldsymbol{q}$, etc. If $P(x,y) \in \boldsymbol{q}$ then we also write $P^-(y,x) \in \boldsymbol{q}$. By a *homomorphism* (or a *match*) from $\boldsymbol{q}(\boldsymbol{x})$ to $\mathcal{C}_\mathcal{K}$ we understand any map $h\colon \boldsymbol{x} \cup \boldsymbol{y} \cup \mathsf{ind}(\mathcal{A}) \to \Delta^{\mathcal{C}_\mathcal{K}}$ such that the following conditions hold:

- $h(a) = a^{\mathcal{C}_\mathcal{K}}$, for every $a \in \mathsf{ind}(\mathcal{A})$,
- $h(x) \in \mathsf{ind}(\mathcal{A})$, for every x in $\boldsymbol{x}$,
- if $A(z) \in \boldsymbol{q}$ then $h(z) \in A^{\mathcal{C}_\mathcal{K}}$,
- if $P(z,z') \in \boldsymbol{q}$ then $(h(z), h(z')) \in P^{\mathcal{C}_\mathcal{K}}$.

In this case we write $h\colon \boldsymbol{q} \to \mathcal{C}_\mathcal{K}$ (a homomorphism is easily extended from terms to the set of atoms of $\boldsymbol{q}$). It is readily seen that $\mathcal{C}_\mathcal{K} \models \boldsymbol{q}(\boldsymbol{a})$ iff there is a homomorphism $h\colon \boldsymbol{q} \to \mathcal{C}_\mathcal{K}$ such that $h(\boldsymbol{x}) = \boldsymbol{a}$. We are now fully equipped to introduce the tree-witness rewriting of a CQ $\boldsymbol{q}(\boldsymbol{x})$ and an *OWL 2 QL* TBox $\mathcal{T}$.

3.2 PE-rewritings

Following the divide and conquer strategy, we first define our rewriting in two simplified cases.

Flat TBoxes. To begin with, let us assume that the given TBox $\mathcal{T}$ is *flat* in the sense that it contains no axioms of the form $B \sqsubseteq \exists R$. This means that the anonymous part of the canonical model $\mathcal{C}_\mathcal{K}$, for any $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, is *empty*, and so we have to look for homomorphisms into to the ABox part of the canonical model. Recall that the canonical model $\mathcal{C}_\mathcal{K}$ is constructed by extending $\mathcal{A}$ with all the consequences of $\mathcal{A}$ with respect to $\mathcal{T}$. For example, if $P(a, b) \in \mathcal{A}$ and $\mathcal{T}$ contains $\exists P \sqsubseteq A$, then $\mathcal{A}$ is extended with the atom $A(a)$. So, to obtain an FO-rewriting of $\boldsymbol{q}$ and $\mathcal{T}$, it is enough to replace every atom α in $\boldsymbol{q}$ with a disjunction of all atoms that imply α over $\mathcal{T}$. More precisely, for any concept name A and role name P, we take the formulas:

$$\mathsf{ext}_A(u) = \bigvee_{\mathcal{T} \models A' \sqsubseteq A} A'(u) \;\lor \bigvee_{\mathcal{T} \models \exists R \sqsubseteq A} \exists v\, R(u, v), \tag{1}$$

$$\mathsf{ext}_P(u, v) = \bigvee_{\mathcal{T} \models R \sqsubseteq P} R(u, v). \tag{2}$$

We define a PE-query $\boldsymbol{q}_{\mathsf{ext}}(\boldsymbol{x})$ as the result of replacing every atom $A(u)$ in $\boldsymbol{q}$ with $\mathsf{ext}_A(u)$ and every atom $P(u, v)$ in $\boldsymbol{q}$ with $\mathsf{ext}_P(u, v)$. It is not hard to see that, for any ABox $\mathcal{A}$ and any $\boldsymbol{a} \subseteq \mathsf{ind}(\mathcal{A})$, we have $\mathcal{C}_\mathcal{K} \models \boldsymbol{q}(\boldsymbol{a})$ iff $\mathcal{A} \models \boldsymbol{q}_{\mathsf{ext}}(\boldsymbol{a})$. Thus, we arrive at the following:

Proposition 1. *For any CQ $\boldsymbol{q}(\boldsymbol{x})$ and any flat OWL 2 QL TBox $\mathcal{T}$, $\boldsymbol{q}_{\mathsf{ext}}(\boldsymbol{x})$ is a PE-rewriting of $\boldsymbol{q}$ and $\mathcal{T}$.*

Thus, in the flat case, it is really easy to compute PE-rewritings.

Example 7. Consider the CQ $\boldsymbol{q}(x) = \exists y\, \big(A(x) \land P(x, y)\big)$ and the flat TBox $\mathcal{T}$ with the axioms

$$A' \sqsubseteq A, \qquad \exists P \sqsubseteq A, \qquad \exists R' \sqsubseteq A',$$
$$R^- \sqsubseteq P, \qquad R' \sqsubseteq P, \qquad S \sqsubseteq R.$$

Then

$$\mathsf{ext}_A(x) = A(x) \lor A'(x) \lor \exists z\, P(x, z) \lor \exists z\, R'(x, z) \lor \exists z\, R(z, x) \lor \exists z\, S(z, x),$$
$$\mathsf{ext}_P(x, y) = P(x, y) \lor R(y, x) \lor R'(x, y) \lor S(y, x).$$

Therefore, the PE-rewriting is as follows:

$$\boldsymbol{q}_{\mathsf{ext}}(x) = \exists y\, \big[\big(A(x) \lor A'(x) \lor \exists z\, P(x, z) \lor \exists z\, R'(x, z) \lor \exists z\, R(z, x) \lor \exists z\, S(z, x)\big) \;\land \big(P(x, y) \lor R(y, x) \lor R'(x, y) \lor S(y, x)\big)\big].$$

H-complete ABoxes. In the second simplified case, we assume that all the ABoxes for which we have to construct an FO-rewriting of $\boldsymbol{q}(\boldsymbol{x})$ and (not necessarily flat) $\mathcal{T}$ are *H-complete with respect to* $\mathcal{T}$ in the sense that

$$A(a) \in \mathcal{A} \quad \text{if} \quad A'(a) \in \mathcal{A} \text{ and } \mathcal{T} \models A' \sqsubseteq A,$$
$$A(a) \in \mathcal{A} \quad \text{if} \quad R(a, b) \in \mathcal{A} \text{ and } \mathcal{T} \models \exists R \sqsubseteq A,$$
$$P(a, b) \in \mathcal{A} \quad \text{if} \quad R(a, b) \in \mathcal{A} \text{ and } \mathcal{T} \models R \sqsubseteq P,$$

for all concept names A, A', roles R and role names P. An FO-query $\boldsymbol{q}'(\boldsymbol{x})$ is an *FO-rewriting of $\boldsymbol{q}$ and $\mathcal{T}$ over H-complete ABoxes* if, for any H-complete (with respect to $\mathcal{T}$) ABox $\mathcal{A}$ and any $\boldsymbol{a} \subseteq \mathsf{ind}(\mathcal{A})$, we have $(\mathcal{T}, \mathcal{A}) \models \boldsymbol{q}(\boldsymbol{a})$ iff $\mathcal{A} \models \boldsymbol{q}'(\boldsymbol{a})$. Note that if $\mathcal{T}$ is flat then $\boldsymbol{q}$ itself can clearly serve as a rewriting of $\boldsymbol{q}$ and $\mathcal{T}$ over H-complete ABoxes. The following example illustrates the notions we need in order to introduce the tree-witness rewriting over H-complete ABoxes.

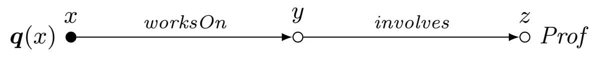
Example 8. Consider an ontology $\mathcal{T}$ with the axioms

$$\begin{aligned} &\mathit{RA} \sqsubseteq \exists \mathit{worksOn}.\mathit{Project}, && (3)\\ &\mathit{Project} \sqsubseteq \exists \mathit{isManagedBy}.\mathit{Prof}, && (4)\\ &\mathit{worksOn}^- \sqsubseteq \mathit{involves}, && (5)\\ &\mathit{isManagedBy} \sqsubseteq \mathit{involves} && (6)\end{aligned}$$

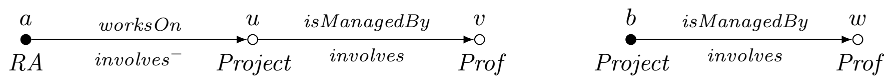
and the CQ asking to find those who work with professors:

$$\boldsymbol{q}(x) = \exists y, z\, \big(\mathit{worksOn}(x, y) \land \mathit{involves}(y, z) \land \mathit{Prof}(z)\big),$$

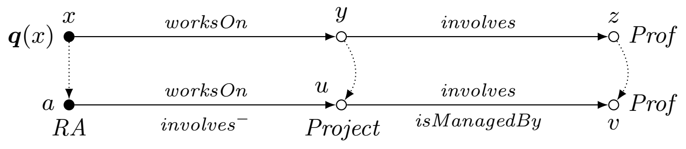
or graphically:



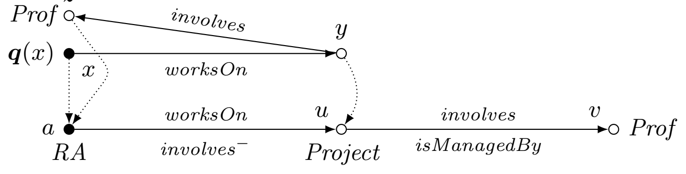
Observe that if the canonical model $\mathcal{C}_\mathcal{K}$ of $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, for some ABox $\mathcal{A}$, contains individuals $a \in \mathit{RA}^{\mathcal{C}_\mathcal{K}}$ and $b \in \mathit{Project}^{\mathcal{C}_\mathcal{K}}$, then $\mathcal{C}_\mathcal{K}$ must also contain the following fragments:



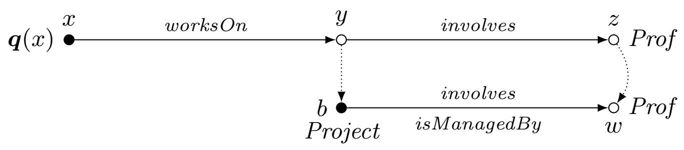
Here the vertices $\circ$ are either named individuals from the ABox or anonymous witnesses for the existential quantifiers (generated by the axioms (3) and (4)). It follows then that a is an answer to $\boldsymbol{q}(x)$ if $a \in \mathit{RA}^{\mathcal{C}_\mathcal{K}}$ because we have the following homomorphism from $\boldsymbol{q}$ to $\mathcal{C}_\mathcal{K}$:



Alternatively, if a is in both $\mathit{RA}^{\mathcal{C}_\mathcal{K}}$ and $\mathit{Prof}^{\mathcal{C}_\mathcal{K}}$, then we obtain the following homomorphism:



Another option is to map x and y to ABox individuals, a and b, and if b is in $\mathit{Project}^{\mathcal{C}_\mathcal{K}}$, then the last two atoms of $\boldsymbol{q}$ can be mapped to the anonymous part:

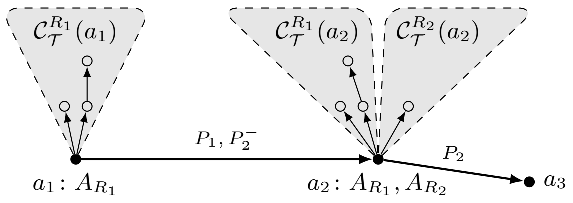


Finally, all the atoms of $\boldsymbol{q}$ can be mapped to ABox individuals. The possible ways of mapping parts of $\boldsymbol{q}$ to the anonymous part of the canonical model are called *tree witnesses*. The tree-witnesses for $\boldsymbol{q}$ found above give the following tree-witness rewriting $\boldsymbol{q}_{\mathsf{tw}}$ of $\boldsymbol{q}$ and $\mathcal{T}$ over H-complete ABoxes:

$$\boldsymbol{q}_{\mathsf{tw}}(x) = \exists y, z \Big[\big(\mathit{worksOn}(x,y) \land \mathit{involves}(y,z) \land \mathit{Prof}(z)\big) \lor \\ \mathit{RA}(x) \lor \big(\mathit{RA}(x) \land \mathit{Prof}(x)\big) \lor \big(\mathit{worksOn}(x,y) \land \mathit{Project}(y)\big)\Big].$$

We now give a general definition of tree-witness rewriting over H-complete ABoxes. For every role name R in $\mathcal{T}$, we take two fresh concept names A_R, A_{R^-} and add to $\mathcal{T}$ the axioms $A_R \equiv \exists R$ and $A_{R^-} \equiv \exists R^-$. We say that the resulting TBox is in *normal form* and assume, without loss of generality, that every TBox in this section is in normal form.

Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$. Every individual $a \in \mathsf{ind}(\mathcal{A})$ with $\mathcal{K} \models A_R(a)$ is a root of a (possibly infinite) subtree $\mathcal{C}^R_\mathcal{T}(a)$ of $\mathcal{C}_\mathcal{K}$, which may intersect another such tree only on their common root a. Every $\mathcal{C}^R_\mathcal{T}(a)$ is isomorphic to the canonical model of $(\mathcal{T}, \{A_R(a)\})$.



$\mathcal{C}^{R_1}_\mathcal{T}(a_1)$
$\mathcal{C}^{R_1}_\mathcal{T}(a_2)$
$\mathcal{C}^{R_2}_\mathcal{T}(a_2)$
P_1, P_2^-
P_2
$a_1 : A_{R_1}$
$a_2 : A_{R_1}, A_{R_2}$
a_3

Suppose now that there is a homomorphism $h\colon \boldsymbol{q} \to \mathcal{C}_\mathcal{K}$. Then h splits $\boldsymbol{q}$ into the subquery mapped by h to the ABox part and the subquery mapped to the anonymous part of $\mathcal{C}_\mathcal{K}$, which is a union of the trees $\mathcal{C}^R_\mathcal{T}(a)$. We can think of a rewriting of $\boldsymbol{q}$ and $\mathcal{T}$ as listing possible splits of $\boldsymbol{q}$ into such subqueries.

Suppose $\boldsymbol{q}' \subseteq \boldsymbol{q}$ and there is a homomorphism $h\colon \boldsymbol{q}' \to \mathcal{C}^R_\mathcal{T}(a)$, for some a, such that h maps all answer variables in $\boldsymbol{q}'$ to a. Let $\mathsf{t}_\mathsf{r} = h^{-1}(a)$ and let t_i be the remaining set of (existentially quantified) variables in $\boldsymbol{q}'$. Suppose $\mathsf{t}_\mathsf{i} \neq \emptyset$. We call the pair $\mathsf{t} = (\mathsf{t}_\mathsf{r}, \mathsf{t}_\mathsf{i})$ a *tree witness for* $\boldsymbol{q}$ *and* $\mathcal{T}$ *generated by* R if the query $\boldsymbol{q}'$ is a *minimal* subset of $\boldsymbol{q}$ such that, for any $y \in \mathsf{t}_\mathsf{i}$, every atom in $\boldsymbol{q}$ containing y belongs to $\boldsymbol{q}'$. In this case, we denote $\boldsymbol{q}'$ by $\boldsymbol{q}_\mathsf{t}$. By definition, we have

$$\boldsymbol{q}_\mathsf{t} = \big\{ S(\boldsymbol{z}) \in \boldsymbol{q} \mid \boldsymbol{z} \subseteq \mathsf{t}_\mathsf{r} \cup \mathsf{t}_\mathsf{i} \text{ and } \boldsymbol{z} \not\subseteq \mathsf{t}_\mathsf{r} \big\}.$$

Note that the same tree witness $\mathfrak{t} = (\mathfrak{t}_\mathsf{r}, \mathfrak{t}_\mathsf{i})$ can be generated by different roles R. We denote the set of all such roles by $\Omega_\mathfrak{t}$ and define the formula

$$\mathsf{tw}_\mathfrak{t} \quad = \quad \bigvee_{R \in \Omega_\mathfrak{t}} \exists z \left(A_R(z) \quad \wedge \quad \bigwedge_{x \in \mathfrak{t}_\mathsf{r}} (x = z) \right). \tag{7}$$

(From a practical point of view, it is enough to take only A_R for $\leq_\mathcal{T}$-maximal roles R.)

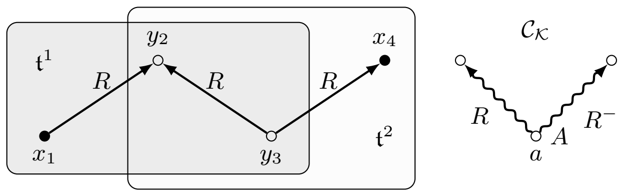
Tree witnesses $\mathfrak{t}$ and $\mathfrak{t}'$ are called *consistent* if $\boldsymbol{q}_\mathfrak{t} \cap \boldsymbol{q}_{\mathfrak{t}'} = \emptyset$. Each *consistent set* Θ of tree witnesses (in which any pair of distinct tree witnesses is consistent) determines a subquery $\boldsymbol{q}_\Theta$ of $\boldsymbol{q}$ that comprises all atoms of the $\boldsymbol{q}_\mathfrak{t}$, for $\mathfrak{t} \in \Theta$. The subquery $\boldsymbol{q}_\Theta$ is to be mapped to the $\mathcal{C}^R_\mathcal{T}(a)$, whereas the remainder, $\boldsymbol{q} \setminus \boldsymbol{q}_\Theta$, obtained by removing the atoms of $\boldsymbol{q}_\Theta$ from $\boldsymbol{q}$, is mapped to $\mathsf{ind}(\mathcal{A})$. The following PE-query $\boldsymbol{q}_\mathsf{tw}$ is called the *tree-witness rewriting of* $\boldsymbol{q}$ *and* $\mathcal{T}$ *over H-complete ABoxes*:

$$\boldsymbol{q}_\mathsf{tw}(\boldsymbol{x}) \quad = \quad \bigvee_{\Theta \text{ consistent}} \exists \boldsymbol{y} \Big((\boldsymbol{q} \setminus \boldsymbol{q}_\Theta) \ \wedge \ \bigwedge_{\mathfrak{t} \in \Theta} \mathsf{tw}_\mathfrak{t} \Big). \tag{8}$$

Example 9. Consider the KB $\mathcal{K} = (\mathcal{T}, \{A(a)\})$, where

$$\mathcal{T} \quad = \quad \big\{ A \sqsubseteq \exists R,\ A \sqsubseteq \exists R^-,\ A_R \equiv \exists R,\ A_{R^-} \equiv \exists R^- \big\},$$

and the CQ $\boldsymbol{q}(x_1, x_4) = \{\, R(x_1, y_2),\, R(y_3, y_2),\, R(y_3, x_4) \,\}$ shown in the picture below alongside the canonical model $\mathcal{C}_\mathcal{K}$ (with A_R and A_{R^-} omitted).



There are two tree witnesses for $\boldsymbol{q}$ and $\mathcal{T}$: $\mathfrak{t}^1 = (\mathfrak{t}^1_\mathsf{r}, \mathfrak{t}^1_\mathsf{i})$ generated by R, and $\mathfrak{t}^2 = (\mathfrak{t}^2_\mathsf{r}, \mathfrak{t}^2_\mathsf{i})$ generated by R^-, with

$$\mathfrak{t}^1_\mathsf{r} = \{x_1, y_3\}, \qquad \mathfrak{t}^1_\mathsf{i} = \{y_2\}, \qquad \mathsf{tw}_{\mathfrak{t}^1} = \exists z \, (A_R(z) \wedge (x_1 = z) \wedge (y_3 = z)),$$
$$\mathfrak{t}^2_\mathsf{r} = \{y_2, x_4\}, \qquad \mathfrak{t}^2_\mathsf{i} = \{y_3\}, \qquad \mathsf{tw}_{\mathfrak{t}^2} = \exists z \, (A_{R^-}(z) \wedge (x_4 = z) \wedge (y_2 = z)).$$

We have $\boldsymbol{q}_{\mathfrak{t}^1} = \{R(x_1, y_2), R(y_3, y_2)\}$ and $\boldsymbol{q}_{\mathfrak{t}^2} = \{R(y_3, y_2), R(y_3, x_4)\}$, so $\mathfrak{t}^1$ and $\mathfrak{t}^2$ are inconsistent. Thus, we obtain the following tree-witness rewriting over H-complete ABoxes:

$$\boldsymbol{q}_\mathsf{tw}(x_1, x_4) \ = \ \exists y_2, y_3 \big[(R(x_1, y_2) \wedge R(y_3, y_2) \wedge R(y_3, x_4)) \vee \\ (R(y_3, x_4) \wedge \mathsf{tw}_{\mathfrak{t}^1}) \vee (R(x_1, y_2) \wedge \mathsf{tw}_{\mathfrak{t}^2}) \big].$$

Theorem 3 ([33]). *For any ABox* $\mathcal{A}$ *that is H-complete with respect to* $\mathcal{T}$ *and any* $\boldsymbol{a} \subseteq \mathsf{ind}(\mathcal{A})$*, we have* $\mathcal{C}_{(\mathcal{T},\mathcal{A})} \models \boldsymbol{q}(\boldsymbol{a})$ *iff* $\mathcal{A} \models \boldsymbol{q}_\mathsf{tw}(\boldsymbol{a})$.

Tree-witness rewriting. Finally, to obtain an FO-rewriting of $\boldsymbol{q}(\boldsymbol{x})$ and $\mathcal{T}$ over *arbitrary* ABoxes, it is enough to take the tree-witness rewriting $\boldsymbol{q}_{\mathsf{tw}}$ over H-complete ABoxes and replace every atom $\alpha(\boldsymbol{u})$ in it with $\mathsf{ext}_\alpha(\boldsymbol{u})$.

3.3 NDL-rewritings

Next, we show how the tree-witness rewriting can be represented as an NDL-query. We remind the reader that a *datalog program*, Π, is a finite set of Horn clauses (or rules) of the form

$$\forall \boldsymbol{x}\, (\gamma_1 \land \dots \land \gamma_m \to \gamma_0),$$

where each γ_i is an atom $P(x_1, \dots, x_l)$ with $x_i \in \boldsymbol{x}$ (see e.g., [1]). The atom γ_0 is called the *head* of the clause, and $\gamma_1, \dots, \gamma_m$ its *body*. In the datalog literature, a standard agreement is to omit the universal quantifiers, replace $\land$ with a comma, and put the head before the body; thus, the rule above is written as

$$\gamma_0 \leftarrow \gamma_1, \dots, \gamma_m.$$

All variables occurring in the head must also occur in the body. We will also assume that the heads do not contain constant symbols. A predicate P *depends* on a predicate Q in Π if Π contains a clause whose head is P and whose body contains Q. Π is called *nonrecursive* if this dependence relation for Π is acyclic. For example, we can define the ext predicates (1) and (2) by a nonrecursive datalog program with the following rules:

$$\mathsf{ext}_A(x) \leftarrow A'(x), \qquad \text{for a concept name } A \text{ with } \mathcal{T} \models A' \sqsubseteq A, \tag{9}$$

$$\mathsf{ext}_A(x) \leftarrow R(x,y), \qquad \text{for a concept name } A \text{ with } \mathcal{T} \models \exists R \sqsubseteq A, \tag{10}$$

$$\mathsf{ext}_P(x,y) \leftarrow R(x,y), \qquad \text{for a role name } P \text{ with } \mathcal{T} \models R \sqsubseteq P. \tag{11}$$

(Note that $\forall x, y\, (R(x,y) \to \mathsf{ext}_A(x))$ is equivalent to $\forall x\, (\exists y\, R(x,y) \to \mathsf{ext}_A(x))$.)

Let $\boldsymbol{q}(\boldsymbol{x})$ be a CQ and $\mathcal{T}$ an *OWL 2 QL* TBox. For a nonrecursive datalog program Π and an atom $\boldsymbol{q}'(\boldsymbol{x})$, we say that $(\Pi, \boldsymbol{q}')$ is an *NDL-rewriting of* $\boldsymbol{q}(\boldsymbol{x})$ *and* $\mathcal{T}$ (over H-complete ABoxes) in case $(\mathcal{T}, \mathcal{A}) \models \boldsymbol{q}(\boldsymbol{a})$ iff $\Pi, \mathcal{A} \models \boldsymbol{q}'(\boldsymbol{a})$, for any (H-complete) ABox $\mathcal{A}$ and any $\boldsymbol{a} \subseteq \mathsf{ind}(\mathcal{A})$. An NDL-rewriting over arbitrary ABoxes can clearly be obtained from an NDL-rewriting over H-complete ABoxes by plugging in the ext rules above (at the price of a polynomial blowup).

Let us see how the tree-witness PE-rewriting (8) will look like if we represent it as an NDL-rewriting over H-complete ABoxes. Suppose $\mathfrak{t} = (\mathfrak{t}_\mathsf{r}, \mathfrak{t}_\mathsf{i})$ is a tree witness for $\boldsymbol{q}$ and $\mathcal{T}$ with $\mathfrak{t}_\mathsf{r} = \{t_1, \dots, t_k\}$, $k \geq 0$. We associate with $\mathfrak{t}$ a k-ary predicate $\mathsf{tw}_\mathfrak{t}$ defined by the following set of rules:

$$\mathsf{tw}_\mathfrak{t}(x, \dots, x) \leftarrow A_R(x), \qquad \text{for } R \in \Omega_\mathfrak{t}. \tag{12}$$

If $\mathfrak{t}_\mathsf{r} \neq \emptyset$ then (12) makes all the arguments of $\mathsf{tw}_\mathfrak{t}$ equal; otherwise, $\mathfrak{t}_\mathsf{r} = \emptyset$ and $\mathsf{tw}_\mathfrak{t}$ is a propositional variable, with x being existentially quantified in the body of (12). As the arguments of $\mathsf{tw}_\mathfrak{t}$ play identical roles, we can write $\mathsf{tw}_\mathfrak{t}(\mathfrak{t}_\mathsf{r})$ without specifying any order on the set $\mathsf{t_r}$. We obtain an NDL-rewriting $(\Pi, \boldsymbol{q}_{\mathsf{tw}}(\boldsymbol{x}))$ of $\boldsymbol{q}(\boldsymbol{x})$ and $\mathcal{T}$ over H-complete ABoxes by taking Π to be the nonrecursive datalog program containing the rules of the form (12) together with the rules

$$\boldsymbol{q}_{\mathsf{tw}}(\boldsymbol{x}) \leftarrow (\boldsymbol{q} \setminus \boldsymbol{q}_{\Theta}), \mathsf{tw}_{\mathsf{t}^1}(\mathsf{t}^1_\mathsf{r}), \dots, \mathsf{tw}_{\mathsf{t}^k}(\mathsf{t}^k_\mathsf{r}), \quad \text{for consistent } \Theta = \{\mathsf{t}^1_\mathsf{r}, \dots, \mathsf{t}^k_\mathsf{r}\}. \quad (13)$$

Example 10. Let $\boldsymbol{q}$ and $\mathcal{T}$ be the same as in Example 9. Denote by Π the datalog program given below:

$$\begin{array}{rcl}
\boldsymbol{q}_{\mathsf{tw}}(x_1, x_4) & \leftarrow & R(x_1, y_2), R(y_3, y_2), R(y_3, x_4), \\
\boldsymbol{q}_{\mathsf{tw}}(x_1, x_4) & \leftarrow & R(y_3, x_4), \mathsf{tw}_{\mathsf{t}^1}(x_1, y_3), \\
\boldsymbol{q}_{\mathsf{tw}}(x_1, x_4) & \leftarrow & R(x_1, y_2), \mathsf{tw}_{\mathsf{t}^2}(y_2, x_4), \\
\mathsf{tw}_{\mathsf{t}^1}(x, x) & \leftarrow & A_R(x), \\
\mathsf{tw}_{\mathsf{t}^2}(x, x) & \leftarrow & A_{R^-}(x).
\end{array}$$

Then $(\Pi, \boldsymbol{q}_{\mathsf{tw}}(x_1, x_4))$ is an NDL-rewriting of $\boldsymbol{q}$ and $\mathcal{T}$ over H-complete ABoxes. To obtain an NDL-rewriting over arbitrary ABoxes, we replace all predicates in the rules above with their ext counterparts and add the appropriate set of rules (9)–(11).

4 Long Rewritings, Short Rewritings

The attractive idea of OBDA with databases relies upon the empirical fact that answering CQs in RDBMSs is usually very efficient in practice. A complexity-theoretic justification for this fact is as follows. In general, to evaluate a Boolean CQ $\boldsymbol{q}$ with existentially quantified variables $\boldsymbol{y}$ over a database instance D, we require—in the worst case—time $O(|\boldsymbol{q}| \cdot |D|^{|\boldsymbol{y}|})$, where $|\boldsymbol{q}|$ is the number of atoms in $\boldsymbol{q}$. The problem of CQ evaluation is $W[1]$-complete [53], and so can be really hard for RDBMSs if the input queries are large. However, if $\boldsymbol{q}$ is tree-shaped (of bounded treewidth, to be more precise)—which is often the case in practice—it can be evaluated in polynomial time, *poly*$(|\boldsymbol{q}|, |D|)$ in symbols [36, 15, 26].

In the context of OBDA, an RDBMS is evaluating not the original CQ $\boldsymbol{q}$ but an FO-rewriting of $\boldsymbol{q}$ and the given *OWL 2 QL* TBox $\mathcal{T}$. All standard FO-rewritings are of size $O((|\boldsymbol{q}| \cdot |\mathcal{T}|)^{|\boldsymbol{q}|})$ in the worst case. For example, the size of the tree-witness rewriting (8) is $|\boldsymbol{q}_{\mathsf{tw}}| = O(|\Xi| \cdot |\boldsymbol{q}| \cdot |\mathcal{T}|)$, where Ξ is the collection of all consistent sets of tree witnesses for $\boldsymbol{q}$ and $\mathcal{T}$ (the $|\mathcal{T}|$ factor comes from the $\mathsf{tw_t}$-formulas and multiple roles that may generate a tree witness). Thus, in principle, if there are many consistent sets of tree witnesses for $\boldsymbol{q}$ and $\mathcal{T}$, the rewriting $\boldsymbol{q}_{\mathsf{tw}}$ may become prohibitively large for RDBMSs. Recall also that $\boldsymbol{q}_{\mathsf{tw}}$ is a rewriting over *H-complete* ABoxes; to obtain a rewriting over arbitrary ABoxes, we have to replace each atom α in $\boldsymbol{q}_{\mathsf{tw}}$ with the respective ext_α. This will add another factor $|\mathcal{T}|$ to the size of $\boldsymbol{q}_{\mathsf{tw}}$. RDBMSs are known to be most efficient for evaluating CQs and unions of CQs (UCQs for short) of reasonable size. But transforming a PE-rewriting to the UCQ form can cause an exponential blowup in the size of the query (consider, for example, the PE-rewriting $\boldsymbol{q}_{\mathsf{ext}}(x)$

in Example 7 and imagine that the original CQ $\boldsymbol{q}$ contains many atoms). These observations put forward the followings questions:

- What is the overhead of answering CQs via *OWL 2 QL* ontologies compared to standard database query answering in the worst case?
- What is the size of FO-rewritings of CQs and *OWL 2 QL* ontologies in the worst case?
- Can rewritings of one type (say, FO) be substantially shorter than rewritings of another type (say, PE)?
- Are there interesting and useful sufficient conditions on CQs and ontologies under which rewritings are short?

In this section, we give an overview of the answers to the above questions obtained so far [32, 35, 33, 24, 31].

In general, succinctness problems such as the second and third questions above can be very hard to solve. Perhaps one of the most interesting and important examples is succinctness of various formalisms for computing Boolean functions such as propositional formulas, branching programs and circuits, which has been investigated since Shannon's seminal work of 1949 [67], where he suggested the size of the smallest circuit computing a Boolean function as a measure of the complexity of that function.

We remind the reader (see, e.g., [3, 30] for more details) that an *n-ary Boolean function*, for $n \geq 1$, is a function $f\colon \{0,1\}^n \to \{0,1\}$. An *$n$-input Boolean circuit*, $\boldsymbol{C}$, for $n \geq 1$, is a directed acyclic graph with n *sources* (inputs) and one *sink* (output). Every non-source vertex of $\boldsymbol{C}$ is called a *gate*; it is labelled with either $\wedge$ or $\vee$, in which case it has two incoming edges, or with $\neg$, in which case there is one incoming edge. The number of vertices in $\boldsymbol{C}$ will be denoted by $|\boldsymbol{C}|$. We think of a *Boolean formula* as a circuit in which every gate has at most one outgoing edge. If $\boldsymbol{x} \in \{0,1\}^n$, then $\boldsymbol{C}(\boldsymbol{x})$ is the *output* of $\boldsymbol{C}$ on input $\boldsymbol{x}$. We say that $\boldsymbol{C}$ *computes* a Boolean function f if $\boldsymbol{C}(\boldsymbol{x}) = f(\boldsymbol{x})$, for every $\boldsymbol{x} \in \{0,1\}^n$.

In the circuit complexity theory, we are interested in *families of Boolean functions*, that is, sequences $f^1, f^2, \dots$, where each f^n is an n-ary Boolean function. For example, we can consider the family $\textsc{Clique}_{n,k}(\boldsymbol{e})$ of Boolean functions of $n(n-1)/2$ variables $e_{jj'}$, $1 \leq j < j' \leq n$, that return 1 iff the graph with vertices $\{1, \dots, n\}$ and edges $\{\{j, j'\} \mid e_{jj'} = 1\}$ contains a k-clique, for some fixed k.

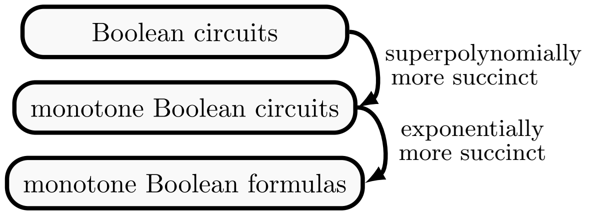
Given a function $T\colon \mathbb{N} \to \mathbb{N}$, by a *$T$-size family of circuits* we mean a sequence $\boldsymbol{C}^1, \boldsymbol{C}^2, \dots$, where each $\boldsymbol{C}^n$ is an n-input Boolean circuits of size $|\boldsymbol{C}^n| \leq T(n)$. Every family f^n of Boolean functions can clearly be computed by circuits of size $n \cdot 2^n$ (take disjunctive normal forms representing the f^n). The class of languages that are decidable by families of *polynomial-size* circuits is denoted by P/poly. It is known that P $\subsetneqq$ P/poly. Thus, one might hope to prove that P $\neq$ NP by showing that NP $\not\subseteq$ P/poly. In other words, to crack one of the most important problems in computer science and mathematics,[1] it is enough to find a family of Boolean functions in NP that cannot be computed by a polynomial-size family

[1] It is actually one of the seven Millennium Prize Problems worth of \$1 000 000 each; consult `www.claymath.org/millennium`.

of circuits. This does not look like a particularly hard problem! After all, it has been known since 1949 that the majority of Boolean functions can only be computed by exponential-size circuits. Yet, so far no one has managed to find a concrete family of functions in NP that need circuits with more than $4.5n - o(n)$ gates [42].

Investigating *restricted* classes of Boolean circuits has proved to be much more successful. The class that is relevant in the context of PE-rewritings consists of *monotone* Boolean functions, that is, those computable by *monotone circuits* with only $\wedge$- and $\vee$-gates. For example, the function $\textsc{Clique}_{n,k}(\boldsymbol{e})$ is monotone. As $\textsc{Clique}_{n,k}$ is NP-complete, the question whether $\textsc{Clique}_{n,k}$ can be computed by polynomial-size circuits is equivalent to the open NP $\subseteq$ P/poly problem. A series of papers, started by Razborov's [59], gave an exponential lower bound for the size of *monotone* circuits computing $\textsc{Clique}_{n,k}$: $2^{\Omega(\sqrt{k})}$ for $k \leq \frac{1}{4}(n/\log n)^{2/3}$ [2]. For monotone formulas, an even better lower bound was obtained: $2^{\Omega(k)}$ for $k = 2n/3$ [58]. It follows that, if we assume NP $\not\subseteq$ P/poly, then no polynomial-size family of (not necessarily monotone) circuits can compute $\textsc{Clique}_{n,k}$.

A few other interesting examples of monotone functions have also been found. Thus, [57] gave a family of monotone Boolean functions that can be computed by polynomial-size monotone circuits, but any monotone *formulas* computing this family are of size $2^{n^{\varepsilon}}$, for some $\varepsilon > 0$. Or there is a family of monotone functions [58, 8] showing that non-monotone Boolean circuits are in general superpolynomially more succinct than monotone circuits.


Boolean circuits
superpolynomially more succinct
monotone Boolean circuits
exponentially more succinct
monotone Boolean formulas

Let us now return to rewritings of CQs and *OWL 2 QL* TBoxes. As we shall see later on in this section, there is a close correspondence between (arbitrary) Boolean formulas and FO-rewritings, monotone Boolean formulas and PE-rewritings, and between monotone Boolean circuits and NDL-rewritings:

Boolean formulas	$\approx$	FO-rewritings
monotone Boolean circuits	$\approx$	NDL-rewritings
monotone Boolean formulas	$\approx$	PE-rewritings

To begin with, we associate with the tree-witness (PE- or NDL-) rewritings $\boldsymbol{q}_{\mathsf{tw}}$ certain monotone Boolean functions that will be called hypergraph functions.

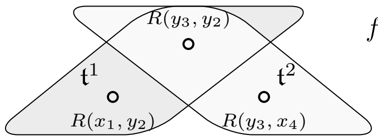
A *hypergraph* $H = (V, E)$ is given by its *vertices* $v \in V$ and *hyperedges* $e \in E$, where $E \subseteq 2^V$. We call a subset $X \subseteq E$ *independent* if $e \cap e' = \emptyset$, for any distinct $e, e' \in X$. The set of vertices that occur in the hyperedges of X is denoted by V_X.

With each vertex $v \in V$ and each hyperedge $e \in E$ we associate propositional variables p_v and p_e, respectively. The *hypergraph function* f_H for a hypergraph H is computed by the Boolean formula

$$f_H \quad = \bigvee_{X \text{ independent}} \Big(\bigwedge_{v \in V \setminus V_X} p_v \ \land \ \bigwedge_{e \in X} p_e \Big). \tag{14}$$

This definition is clearly inspired by (8). The PE-rewriting $\boldsymbol{q}_{\mathsf{tw}}$ of $\boldsymbol{q}$ and $\mathcal{T}$ defines a hypergraph whose vertices are the atoms of $\boldsymbol{q}$ and hyperedges are the sets $\boldsymbol{q}_{\mathfrak{t}}$, for tree witnesses $\mathfrak{t}$ for $\boldsymbol{q}$ and $\mathcal{T}$. We denote this hypergraph by $H^{\boldsymbol{q}}_{\mathcal{T}}$. The formula (14) defining $f_{H^{\boldsymbol{q}}_{\mathcal{T}}}$ is basically the same as the rewriting (8) with the atoms $S(\boldsymbol{z})$ in $\boldsymbol{q}$ and tree witness formulas $\mathsf{tw}_{\mathfrak{t}}$ treated as propositional variables. We denote these variables by $p_{S(\boldsymbol{z})}$ and $p_{\mathfrak{t}}$ (rather than p_v and p_e), respectively.

Example 11. Consider again the CQ $\boldsymbol{q}$ and TBox $\mathcal{T}$ from Example 9. The hypergraph $H^{\boldsymbol{q}}_{\mathcal{T}}$ and its hypergraph function $f_{H^{\boldsymbol{q}}_{\mathcal{T}}}$ are shown below:



$$f_{H^{\boldsymbol{q}}_{\mathcal{T}}} = (p_{R(x_1,y_2)} \land p_{R(y_3,y_2)} \land p_{R(y_3,x_4)}) \lor (p_{R(y_3,x_4)} \land p_{\mathfrak{t}^1}) \lor (p_{R(x_1,y_2)} \land p_{\mathfrak{t}^2}).$$

Suppose now that the hypergraph function $f_{H^{\boldsymbol{q}}_{\mathcal{T}}}$ is computed by some Boolean formula $\chi_{H^{\boldsymbol{q}}_{\mathcal{T}}}$. Consider the FO-formula $\widehat{\chi}_{H^{\boldsymbol{q}}_{\mathcal{T}}}$ obtained by replacing each $p_{S(\boldsymbol{z})}$ in $\chi_{H^{\boldsymbol{q}}_{\mathcal{T}}}$ with $S(\boldsymbol{z})$, each $p_{\mathfrak{t}}$ with $\mathsf{tw}_{\mathfrak{t}}$, and adding the prefix $\exists \boldsymbol{y}$. By comparing (14) and (8), we see that the resulting FO-formula is a rewriting of $\boldsymbol{q}$ and $\mathcal{T}$ over H-complete ABoxes. A monotone circuit computing $f_{H^{\boldsymbol{q}}_{\mathcal{T}}}$ can be converted to an NDL-rewriting of $\boldsymbol{q}$ and $\mathcal{T}$ over H-complete ABoxes. This gives us the following:

Theorem 4. (i) *If the function $f_{H^{\boldsymbol{q}}_{\mathcal{T}}}$ is computed by a Boolean formula $\chi_{H^{\boldsymbol{q}}_{\mathcal{T}}}$, then $\widehat{\chi}_{H^{\boldsymbol{q}}_{\mathcal{T}}}$ is an FO-rewriting of $\boldsymbol{q}$ and $\mathcal{T}$ over H-complete ABoxes.*

(ii) *If $f_{H^{\boldsymbol{q}}_{\mathcal{T}}}$ is computed by a monotone Boolean circuit $\boldsymbol{C}$, then there is an NDL-rewriting of $\boldsymbol{q}$ and $\mathcal{T}$ over H-complete ABoxes of size $O(|\boldsymbol{C}| \cdot (|\boldsymbol{q}| + |\mathcal{T}|))$.*

Thus, the problem of constructing short rewritings is reducible to the problem of finding short Boolean formulas or circuits computing the hypergraph functions. We call a hypergraph H *representable* if there are a CQ $\boldsymbol{q}$ and an *OWL 2 QL* TBox $\mathcal{T}$ such that H is isomorphic to $H^{\boldsymbol{q}}_{\mathcal{T}}$.

Let us consider first hypergraphs of *degree* ≤ 2, in which every vertex belongs to *at most two* hyperedges. There is a striking correspondence between such hypergraphs and *OWL 2 QL* TBoxes $\mathcal{T}$ of *depth one*, which *cannot* have chains of *more than* 1 point in the anonymous part of their canonical models (more precisely, whenever $aw_{[R_1]} \cdots w_{[R_n]}$ is an element of some canonical model for $\mathcal{T}$ then $n \leq 1$). As observed above, PE-rewritings of CQs and flat TBoxes (that is, TBoxes of depth 0) can always be made of polynomial size.

Theorem 5 ([31]). (i) *If $\boldsymbol{q}$ is a CQ and $\mathcal{T}$ a TBox of depth one, then the hypergraph $H^{\boldsymbol{q}}_{\mathcal{T}}$ is of degree ≤ 2.*

(*ii*) *The number of distinct tree witnesses for* $\boldsymbol{q}$ *and* $\mathcal{T}$ *does not exceed the number of variables in* $\boldsymbol{q}$.

(*iii*) *Any hypergraph* H *of degree* ≤ 2 *is representable by means of some CQ and TBox of depth one.*

Here we only consider (*iii*). Suppose that $H = (V, E)$ is a hypergraph of degree ≤ 2. To simplify notation, we assume that any vertex in H belongs to exactly 2 hyperedges, so H comes with two fixed maps $i_1, i_2 \colon V \to E$ such that $i_1(v) \neq i_2(v)$, $v \in i_1(v)$ and $v \in i_2(v)$, for any $v \in V$. For every $e \in E$, we take an individual variable z_e and denote by $\boldsymbol{z}$ the vector of all such variables. For every $v \in V$, we take a role name R_v and define a Boolean CQ $\boldsymbol{q}_H$ by taking:

$$\boldsymbol{q}_H \ = \ \exists \boldsymbol{z} \bigwedge_{v \in V} R_v(z_{i_1(v)}, z_{i_2(v)}).$$

For every hyperedge $e \in E$, let A_e be a concept name and S_e a role name. Consider the TBox $\mathcal{T}_H$ with the following inclusions, for $e \in E$:

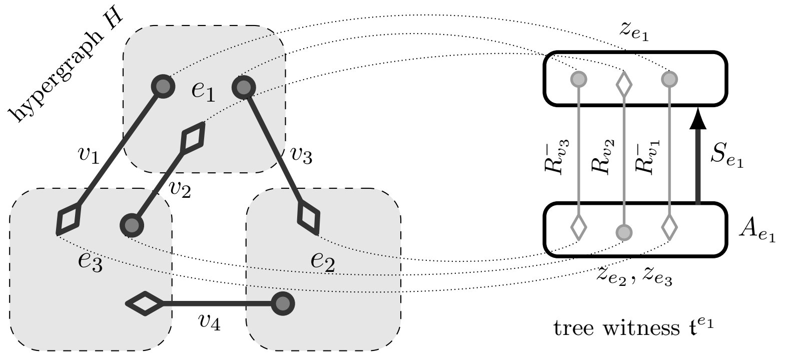
$$\begin{array}{ll} A_e \equiv \exists S_e, & \\ S_e \sqsubseteq R_v^-, & \text{for } v \in V \text{ with } i_1(v) = e, \\ S_e \sqsubseteq R_v, & \text{for } v \in V \text{ with } i_2(v) = e. \end{array}$$

We claim that the hypergraph H is isomorphic to $H_{\mathcal{T}_H}^{\boldsymbol{q}_H}$ and illustrate the proof by an example.

Example 12. Let $H = (V, E)$ with $V = \{v_1, v_2, v_3, v_4\}$ and $E = \{e_1, e_2, e_3\}$, where $e_1 = \{v_1, v_2, v_3\}$, $e_2 = \{v_3, v_4\}$, $e_3 = \{v_1, v_2, v_4\}$. Suppose also that

$$\begin{array}{l} i_1 \colon v_1 \mapsto e_1, \quad v_2 \mapsto e_3, \quad v_3 \mapsto e_1, \quad v_4 \mapsto e_2, \\ i_2 \colon v_1 \mapsto e_3, \quad v_2 \mapsto e_1, \quad v_3 \mapsto e_2, \quad v_4 \mapsto e_3. \end{array}$$

The hypergraph H is shown below, where each vertex v_i is represented by an edge, $i_1(v_i)$ is indicated by the circle-shaped end of the edge and $i_2(v_i)$ by the diamond-shaped one; the hyperedges e_j are shown as large grey squares:



Then

$$\boldsymbol{q}_H = \exists z_{e_1} z_{e_2} z_{e_3} \big(R_{v_1}(z_{e_1}, z_{e_3}) \land R_{v_2}(z_{e_3}, z_{e_1}) \land R_{v_3}(z_{e_1}, z_{e_2}) \land R_{v_4}(z_{e_2}, z_{e_3})\big)$$

and the TBox $\mathcal{T}_H$ contains the following inclusions:

$$\begin{array}{llll}
A_{e_1} \equiv \exists S_{e_1}, & S_{e_1} \sqsubseteq R^-_{v_1}, & S_{e_1} \sqsubseteq R_{v_2}, & S_{e_1} \sqsubseteq R^-_{v_3}, \\
A_{e_2} \equiv \exists S_{e_2}, & S_{e_2} \sqsubseteq R_{v_3}, & S_{e_2} \sqsubseteq R^-_{v_4}, & \\
A_{e_3} \equiv \exists S_{e_3}, & S_{e_3} \sqsubseteq R_{v_1}, & S_{e_3} \sqsubseteq R^-_{v_2}, & S_{e_3} \sqsubseteq R_{v_4}.
\end{array}$$

The canonical model $\mathcal{C}^{S_{e_1}}_{\mathcal{T}_H}(a)$ is shown on the right-hand side of the picture above. We observe now that each variable z_e uniquely determines the tree witness $\mathfrak{t}^e$ with $\boldsymbol{q}_{\mathfrak{t}^e} = \{R_v(z_{i_1(v)}, z_{i_2(v)}) \mid v \in e\}$; $\boldsymbol{q}_{\mathfrak{t}^e}$ and $\boldsymbol{q}_{\mathfrak{t}^{e'}}$ are consistent iff $e \cap e' \neq \emptyset$. It follows that H is isomorphic to $H^{\boldsymbol{q}_H}_{\mathcal{T}_H}$.

It turns out that answering the CQ $\boldsymbol{q}_H$ over $\mathcal{T}_H$ and certain single-individual ABoxes amounts to computing the Boolean function f_H. Let $H = (V, E)$ be a hypergraph of degree 2 with $V = \{v_1, \dots, v_n\}$ and $E = \{e_1, \dots, e_m\}$. We denote by $\boldsymbol{\alpha}(v_i)$ the i-th component of $\boldsymbol{\alpha} \in \{0,1\}^n$ and by $\boldsymbol{\beta}(e_j)$ the j-th component of $\boldsymbol{\beta} \in \{0,1\}^m$. Define an ABox $\mathcal{A}_{\boldsymbol{\alpha},\boldsymbol{\beta}}$ with a single individual a by taking

$$\mathcal{A}_{\boldsymbol{\alpha},\boldsymbol{\beta}} \quad = \quad \{R_{v_i}(a,a) \mid \boldsymbol{\alpha}(v_i) = 1\} \quad \cup \quad \{A_{e_j}(a) \mid \boldsymbol{\beta}(e_j) = 1\}.$$

Theorem 6 ([31]). *Let $H = (V, E)$ be a hypergraph of degree 2. Then, for any tuples $\boldsymbol{\alpha} \in \{0,1\}^{|V|}$ and $\boldsymbol{\beta} \in \{0,1\}^{|E|}$,*

$$(\mathcal{T}_H, \mathcal{A}_{\boldsymbol{\alpha},\boldsymbol{\beta}}) \models \boldsymbol{q}_H \quad \textit{iff} \quad f_H(\boldsymbol{\alpha}, \boldsymbol{\beta}) = 1.$$

What is so special about hypergraphs of degree ≤ 2? First, one can show that all hypergraph functions of degree ≤ 2 can be computed by polynomial-size monotone Boolean circuits—in fact, by polynomial-size monotone nondeterministic branching programs (NBPs), also known as switching-and-rectifier networks [30]. Moreover, the converse also holds: if a family of Boolean functions is computable by polynomial-size NBPs then it can be represented by a family of polynomial-size hypergraphs of degree ≤ 2. As a consequence of this fact, we obtain a positive result on the size of NDL-rewritings:

Theorem 7 ([31]). *For any CQ $\boldsymbol{q}$ and any TBox $\mathcal{T}$ of depth one, there is an NDL-rewriting of $\boldsymbol{q}$ and $\mathcal{T}$ of polynomial size.*

On the 'negative' side, there are families of Boolean functions f_n that are computable by polynomial-size monotone NBPs, but any monotone Boolean formulas computing f_n are of superpolynomial size, at least $2^{\Omega(\log^2 n)}$, to be more precise. For example, Grigni and Sipser [25] consider functions that take the adjacency matrix of a directed graph of n vertices with a distinguished vertex s as input and return 1 iff there is a directed path from s to some vertex of outdegree at least two. Can we use this lower bound to establish a corresponding superpolynomial lower bound for the size of PE-rewritings? The answer naturally depends on what syntactical means we can use in our rewritings.

Let us assume first that the FO- and NDL-rewritings of CQs $\boldsymbol{q}$ and *OWL 2 QL* TBoxes $\mathcal{T}$ can contain equality ($=$), any non-predifined predicates and only those constant symbols that occur in $\boldsymbol{q}$. Thus, we do not allow new constants and various built-in predicates in our rewritings. Such rewritings are sometimes called *pure*.

As any NBP corresponds to a polynomial-size hypergraph of degree ≤ 2, we obtain a sequence H_n of polynomial hypergraphs of degree 2 such that $f_n = f_{H_n}$. We take the sequence of CQs $\boldsymbol{q}_n$ and TBoxes $\mathcal{T}_n$ associated with the hypergraphs of degree ≤ 2 for the sequence f_n of Boolean functions chosen above. We show that any pure PE-rewriting $\boldsymbol{q}'_n$ of $\boldsymbol{q}_n$ and $\mathcal{T}_n$ can be transformed into a monotone Boolean formula χ_n computing f_n and having size $\leq |\boldsymbol{q}'_n|$.

To define χ_n, we eliminate the quantifiers in $\boldsymbol{q}'_n$ in the following way: take a constant a and replace every subformula of the form $\exists x\, \psi(x)$ in $\boldsymbol{q}'_n$ with $\psi(a)$, repeating this operation as long as possible. The resulting formula $\boldsymbol{q}''_n$ is built from atoms of the form $A_e(a)$, $R_v(a,a)$ and $S_e(a,a)$ using $\wedge$ and $\vee$. For every ABox $\mathcal{A}$ with a single individual a, we have $(\mathcal{T}_n, \mathcal{A}) \models \boldsymbol{q}_n$ iff $\mathcal{A} \models \boldsymbol{q}''_n$. Let χ_n be the result of replacing $S_e(a,a)$ in $\boldsymbol{q}''_n$ with $\bot$, $A_e(a)$ with p_e and $R_v(a,a)$ with p_v. Clearly, $|\chi_n| \leq |\boldsymbol{q}'_n|$.

By the definition of $\mathcal{A}_{\boldsymbol{\alpha},\boldsymbol{\beta}}$ and Theorem 6, we then obtain:

$$\chi_n(\boldsymbol{\alpha},\boldsymbol{\beta}) = 1 \quad \text{iff} \quad \mathcal{A}_{\boldsymbol{\alpha},\boldsymbol{\beta}} \models \boldsymbol{q}''_n \quad \text{iff} \quad \mathcal{A}_{\boldsymbol{\alpha},\boldsymbol{\beta}} \models \boldsymbol{q}'_n \quad \text{iff} \\ (\mathcal{T}_n, \mathcal{A}_{\boldsymbol{\alpha},\boldsymbol{\beta}}) \models \boldsymbol{q}_n \quad \text{iff} \quad f_{H_n}(\boldsymbol{\alpha},\boldsymbol{\beta}).$$

It remains to recall that $|\boldsymbol{q}'_n| \geq |\chi_n| \geq 2^{\Omega(\log^2 n)}$. This gives us the first superpolynomial lower bound for the size of pure PE-rewritings.

Theorem 8 ([31]). *There is a sequence of CQs $\boldsymbol{q}_n$ and TBoxes $\mathcal{T}_n$ of depth one such that any pure PE-rewriting of $\boldsymbol{q}_n$ and $\mathcal{T}_n$ (over H-complete ABoxes) is of size $\geq 2^{\Omega(\log^2 n)}$.*

Why are the PE-rewritings in Theorem 8 so large? Let us have another look at the tree-witness rewriting (8) of $\boldsymbol{q}$ and $\mathcal{T}$. Its size depends on the number of distinct consistent sets of tree witnesses for $\boldsymbol{q}$ and $\mathcal{T}$. In view of Theorem 5 (*ii*), $\boldsymbol{q}_n$ and $\mathcal{T}_n$ in Theorem 8 have a polynomial number of tree witnesses. However, many of them are not consistent with each other (see Example 12), which makes it impossible to simplify (8) to a polynomial-size PE-rewriting.

For TBoxes of depth two, we can obtain an even stronger 'negative' result:

Theorem 9 ([31]). *There exists a sequence of CQs $\boldsymbol{q}_n$ and TBoxes $\mathcal{T}_n$ of depth two, such that any pure PE- and NDL-rewriting of $\boldsymbol{q}_n$ and $\mathcal{T}_n$ is of exponential size, while any FO-rewriting of $\boldsymbol{q}_n$ and $\mathcal{T}_n$ is of superpolynomial size* (*unless* NP $\subseteq$ P/*poly*).

This theorem can be proved by showing that the hypergraphs $H_{n,k}$ computing $\textsc{Clique}_{n,k}$ are representable as subgraphs of $H^{\boldsymbol{q}_{n,k}}_{\mathcal{T}_{n,k}}$ for suitable $\boldsymbol{q}_{n,k}$ and $\mathcal{T}_{n,k}$, and then applying quantifier elimination over single-individual ABoxes as above.

Using TBoxes of unbounded depth, one can show that pure FO-rewritings can be superpolynomially more succinct than pure PE-rewritings:

Theorem 10 ([35]). *There is a sequence of CQs $\boldsymbol{q}_n$ of size $O(n)$ and TBoxes $\mathcal{T}_n$ of size $O(n)$ that has polynomial-size FO-rewritings, but its pure PE-rewritings are of size $\geq 2^{\Omega(2^{\log^{1/2} n})}$.*

We can also use the machinery developed above to understand the overhead of answering CQs via *OWL 2 QL* ontologies compared to standard database query answering:

Theorem 11 ([32]). *There is a sequence of CQs $\boldsymbol{q}_n$ and OWL 2 QL TBoxes $\mathcal{T}_n$ such that the problem '$\mathcal{A} \models \boldsymbol{q}_n$?' is in* P, *while the problem '$(\mathcal{T}_n, \mathcal{A}) \models \boldsymbol{q}_n$?' is* NP*-hard (for combined complexity).*

On the other hand, answering tree-shaped CQs (or CQs of bounded treewidth) over TBoxes without role inclusions can be done in polynomial time [6]. On the other hand, a sufficient condition on CQs and *OWL 2 QL* TBoxes that guarantees the existence of polynomial-size pure PE-rewritings can be found in [33].

We conclude this section with a few remarks on 'impure' rewritings that can use new constant symbols not occurring in the given CQ. To prove the superpolynomial and exponential lower bounds on the size of pure rewritings above, we established a connection between monotone circuits for Boolean functions and rewritings for certain CQs and *OWL 2 QL* TBoxes. In fact, this connection also suggests a way of making rewritings substantially shorter. Indeed, recall that although no monotone Boolean circuit of polynomial size can compute $\textsc{Clique}_{n,k}$, we can construct such a circuit if we are allowed to use advice inputs. Indeed, for any family $f^1, f^2, \dots$ of Boolean functions in NP, there exist polynomials p and q and, for each $n \geq 1$, a Boolean circuit $\boldsymbol{C}^n$ with $n + p(n)$ inputs such that $|\boldsymbol{C}^n| \leq q(n)$ and, for any $\boldsymbol{\alpha} \in \{0,1\}^n$, we have

$$f^n(\boldsymbol{\alpha}) = 1 \qquad \text{iff} \qquad \boldsymbol{C}^n(\boldsymbol{\alpha}, \boldsymbol{\beta}) = 1, \quad \text{for some } \boldsymbol{\beta} \in \{0,1\}^{p(n)}.$$

The additional $p(n)$ inputs for $\boldsymbol{\beta}$ in the $\boldsymbol{C}^n$ are called *advice inputs*. These advice inputs make Boolean circuits *nondeterministic* (in the sense that $\boldsymbol{\beta}$ is a guess) and, as a result, exponentially more succinct—in the same way as nondeterministic automata are exponentially more succinct than deterministic ones [50]. To introduce corresponding nondeterministic guesses into query rewritings, we can use additional existentially quantified variables—provided that the domain of quantification contains at least two elements. For this purpose, we can assume that all relevant ABoxes contain two fixed individuals (among others).

Theorem 12 ([24, 35]). *For any CQ $\boldsymbol{q}$ and OWL 2 QL TBox $\mathcal{T}$, there are impure polynomial-size PE- and NDL-rewritings of $\boldsymbol{q}$ and $\mathcal{T}$ over ABoxes containing two fixed constants; the rewritings use $O(|\boldsymbol{q}|)$ additional existential quantifiers.*

This polynomial-size impure rewriting hides the superpolynomial and exponential blowups in the case of pure rewritings behind the additional existential quantification over the newly introduced constants.

5 The Combined Approach

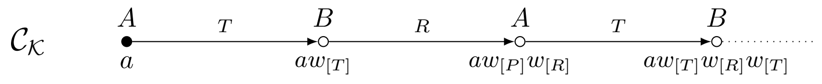
A different approach to OBDA has been suggested in [46, 38, 39]. It is known as the *combined approach* and aims at scenarios where one can manipulate the database. Suppose we are given a CQ $\boldsymbol{q}$ and a TBox $\mathcal{T}$, and we want to query an ABox $\mathcal{A}$. So far, we have first constructed an FO-rewriting $\boldsymbol{q}'$ of $\boldsymbol{q}$ and $\mathcal{T}$, independently of $\mathcal{A}$, and then used an RDBMS to evaluate $\boldsymbol{q}'$ over $\mathcal{A}$. The advantage of this 'classical' approach is that it works even when we cannot modify the data in the data source. However, as we saw above, the rewriting $\boldsymbol{q}'$ can be too large for RDBMSs to cope.

But let us assume that we can manipulate the given ABox $\mathcal{A}$. In this case, a natural question would be: why cannot we first apply $\mathcal{T}$ to $\mathcal{A}$ and then evaluate $\boldsymbol{q}$ over the resulting canonical model? However, many *OWL 2 QL* TBoxes are of infinite depth, and so the canonical model of $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ can be infinite; even if $\mathcal{T}$ is of bounded depth, the canonical model of $\mathcal{K}$ can be of exponential size.

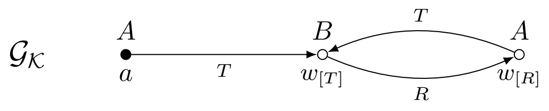
Example 13. Consider the *OWL 2 QL* TBox

$$\mathcal{T} \quad = \quad \{\ A \sqsubseteq \exists T,\ \ \exists T^- \sqsubseteq B,\ \ B \sqsubseteq \exists R,\ \ \exists R^- \sqsubseteq A\ \}.$$

The infinite canonical model of $\mathcal{C}_\mathcal{K}$, for $\mathcal{K} = (\mathcal{T}, \{A(a)\})$, looks as follows:



But what if we fold the infinite chain of alternating R- and T-arrows in a simple cycle such as in the picture below?



Note that $\mathcal{G}_\mathcal{K}$ is still a model of $\mathcal{K}$. Now, consider the CQ

$$\boldsymbol{q}(x) = \exists y, z\,(T(x,y) \land R(y,z) \land T(z,y)).$$

Clearly, we have $\mathcal{G}_\mathcal{K} \models \boldsymbol{q}(a)$, but $\mathcal{C}_\mathcal{K} \not\models \boldsymbol{q}(a)$. Observe, however, that to get rid of this spurious answer a, it is enough to slightly modify $\boldsymbol{q}$ by adding to it the 'filtering conditions' $(z \neq w_{[R]})$, $(y \neq w_{[R^-]})$, $(y \neq w_{[T]})$ and $(z \neq w_{[T^-]})$ as conjuncts (in the scope of $\exists y, z$). Indeed, it is not hard to see that $\boldsymbol{q}$ and $\mathcal{T}$ have no tree witnesses, and so the variables y and z cannot be mapped anywhere in the anonymous part of the canonical model.

It turns out that this idea works perfectly well at least for the case when *OWL 2 QL* TBoxes *do not contain role inclusions*. Suppose $\mathcal{T}$ is such a TBox and $\mathcal{A}$ an arbitrary ABox. We define the *generating model* $\mathcal{G}_\mathcal{K}$ of $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ by taking:

$$\begin{aligned}
\Delta^{\mathcal{G}_\mathcal{K}} &= \{\mathsf{tail}(\sigma) \mid \sigma \in \Delta^{\mathcal{C}_\mathcal{K}}\},\\
a^{\mathcal{G}_\mathcal{K}} &= a, \quad \text{for all } a \in \mathsf{ind}(\mathcal{A}),\\
A^{\mathcal{G}_\mathcal{K}} &= \{\mathsf{tail}(\sigma) \mid \sigma \in A^{\mathcal{C}_\mathcal{K}}\}, \quad \text{for all concept names } A,\\
P^{\mathcal{G}_\mathcal{K}} &= \{(\mathsf{tail}(\sigma), \mathsf{tail}(\sigma')) \mid (\sigma, \sigma') \in P^{\mathcal{C}_\mathcal{K}}\}, \quad \text{for all role names } P,
\end{aligned}$$

where $\mathcal{C}_\mathcal{K}$ is the standard canonical model of $\mathcal{K}$. It is not hard to see that the map $\mathsf{tail}\colon \Delta^{\mathcal{C}_\mathcal{K}} \to \Delta^{\mathcal{G}_\mathcal{K}}$ is a homomorphism from $\mathcal{C}_\mathcal{K}$ onto $\mathcal{G}_\mathcal{K}$, and $\mathcal{C}_\mathcal{K}$ can be viewed as the 'unraveling' of $\mathcal{G}_\mathcal{K}$. The generating model $\mathcal{G}_\mathcal{K}$ can be constructed in polynomial time in $|\mathcal{K}|$. As positive existential queries are preserved under homomorphisms, we obtain:

Theorem 13 ([38]). *For any consistent OWL 2 QL KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, which does not contain role inclusion axioms, any CQ $\boldsymbol{q}(\boldsymbol{x})$ and any tuple $\boldsymbol{a} \subseteq \mathsf{ind}(\mathcal{A})$, if $\mathcal{C}_\mathcal{K} \models \boldsymbol{q}(\boldsymbol{a})$ then $\mathcal{G}_\mathcal{K} \models \boldsymbol{q}(\boldsymbol{a})$ (but not the other way round).*

Suppose now that we are given a CQ $\boldsymbol{q}(\boldsymbol{x}) = \exists \boldsymbol{y}\, \varphi(\boldsymbol{x}, \boldsymbol{y})$. Our aim is to rewrite $\boldsymbol{q}$ to an FO-query $\boldsymbol{q}^\dagger(\boldsymbol{x})$ in such a way that $\mathcal{C}_\mathcal{K} \models \boldsymbol{q}(\boldsymbol{a})$ iff $\mathcal{G}_\mathcal{K} \models \boldsymbol{q}^\dagger(\boldsymbol{a})$, for any tuple $\boldsymbol{a} \subseteq \mathsf{ind}(\mathcal{A})$, and the size of $\boldsymbol{q}^\dagger$ is polynomial in the size of $\boldsymbol{q}$ and $\mathcal{T}$. (Note that the rewriting given in [38, 39] does not depend on $\mathcal{T}$ because of a different definition of tree witness.) We define the rewriting $\boldsymbol{q}^\dagger$ as a conjunction

$$\boldsymbol{q}^\dagger(\boldsymbol{x}) \quad = \quad \exists \boldsymbol{y}\, (\varphi \land \varphi_1 \land \varphi_2 \land \varphi_3),$$

where φ_1, φ_2 and φ_3 are Boolean combinations of equalities $t_1 = t_2$, and each of the t_i is either a term in $\boldsymbol{q}$ or a constant of the form $w_{[R]}$. The purpose of the conjunct

$$\varphi_1 \quad = \quad \bigwedge_{x \in \boldsymbol{x}} \quad \bigwedge_{R \text{ a role in } \mathcal{T}} (x \neq w_{[R]})$$

is to ensure that the answer variables $\boldsymbol{x}$ receive their values from $\mathsf{ind}(\mathcal{A})$ only.

The conjunct φ_2 implements the matching dictated by the tree witnesses. Suppose $\mathfrak{t} = (\mathfrak{t}_\mathsf{r}, \mathfrak{t}_\mathsf{i})$ is a tree witness for $\boldsymbol{q}$ and $\mathcal{T}$ generated by R. If $R(t, t') \in \boldsymbol{q}$ and t' is mapped to $w_{[R]}$, then all the variables in $\mathfrak{t}_\mathsf{r}$ are to be mapped to the same point as t:

$$\varphi_2 \quad = \quad \bigwedge_{\substack{R(t,t') \in \boldsymbol{q} \\ \text{there is a tree witness } \mathfrak{t} \text{ with } R \in \Omega_\mathfrak{t}}} \big((t' = w_{[R]}) \to \bigwedge_{s \in \mathfrak{t}_\mathsf{r}} (s = t)\big).$$

If there is no tree witness $\mathfrak{t}$ generated by R and such that $R(t, t') \in \boldsymbol{q}_\mathfrak{t}$, then t' cannot be mapped to the witness $w_{[R]}$ at all. This is ensured by the conjunct

$$\varphi_3 \quad = \quad \bigwedge_{\substack{R(t,t') \in \boldsymbol{q} \\ \text{no tree witness } \mathfrak{t} \text{ with } R \in \Omega_\mathfrak{t} \text{ exists}}} \big(t' \neq w_{[R]}\big).$$

Theorem 14 ([38]). *For any consistent OWL 2 QL KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ containing no role inclusion axioms, any CQ $\boldsymbol{q}(\boldsymbol{x})$ and any tuple $\boldsymbol{a} \subseteq \mathsf{ind}(\mathcal{A})$, we have $\mathcal{C}_\mathcal{K} \models \boldsymbol{q}(\boldsymbol{a})$ iff $\mathcal{G}_\mathcal{K} \models \boldsymbol{q}^\dagger(\boldsymbol{a})$.*

The rewriting $\boldsymbol{q}^\dagger$ can be computed in polynomial time in $\boldsymbol{q}$ and $\mathcal{T}$ and is of size $O(|\boldsymbol{q}| \cdot |\mathcal{T}|)$.

A slightly different idea was proposed in [48] to extend the combined approach to unrestricted *OWL 2 QL* TBoxes. As before, given a CQ $\boldsymbol{q}(\boldsymbol{x})$, an *OWL 2 QL* TBox $\mathcal{T}$ and an ABox $\mathcal{A}$, we first construct a polynomial-size interpretation $\mathcal{G}'_\mathcal{K}$ representing the (possibly infinite) canonical model $\mathcal{C}_\mathcal{K}$ (in fact, $\mathcal{G}'_\mathcal{K}$ is more involved than $\mathcal{G}_\mathcal{K}$). Then we use an RDBMS to compute the answers to the *original* CQ $\boldsymbol{q}(\boldsymbol{x})$ over $\mathcal{G}'_\mathcal{K}$ stored in the RDBMS. As we saw above, some of the answers can be spurious. To eliminate them, instead of φ_1, φ_2 and φ_3, one can use a 'filtering procedure' that is installed as a user-defined function in the RDBMS. This procedure takes as input a match of the query in $\mathcal{G}'_\mathcal{K}$ and returns 'false' if this match is spurious (that is, cannot be realised in the real canonical model $\mathcal{C}_\mathcal{K}$) and 'true' otherwise. The filtering procedure runs in polynomial time, although it may have to run exponentially many times (to check exponentially many matches for the same answer tuple) in the worst case.

6 OBDA with *Ontop*

In this final section, we consider the architecture of the OBDA system *Ontop* (`ontop.inf.unibz.it`) implemented at the Free University of Bozen-Bolzano and available as a plugin for the ontology editor Protégé 4, a SPARQL end-point and OWLAPI and Sesame libraries.

In OBDA with databases, the data comes from a relational database rather than an ABox. From a logical point of view, a *database schema* [1] contains predicate symbols (with their arity) for both stored database relations (also known as tables) and views (with their definitions in terms of stored relations) as well as a set Σ of *integrity constraints* (in the form of functional and inclusion dependencies; for example, primary and foreign keys). Any instance $\boldsymbol{I}$ of the database schema must satisfy its integrity constraints Σ.

The vocabularies of the database schema and the given TBox are linked together by means of mappings produced by a domain expert or extracted (semi)automatically. There are different known types of mappings: LAV (local-as-views), GAV (global-as-views), GLAV, etc.; consult, e.g., [43] for an overview. Here we concentrate on GAV mappings because they guarantee low complexity of query answering (in what follows we call them simply mappings). A *mapping*, $\mathcal{M}$, is a set of rules of the form

$$S(\boldsymbol{x}) \leftarrow \varphi(\boldsymbol{x}, \boldsymbol{z}),$$

where S is a concept or role name in the ontology and $\varphi(\boldsymbol{x}, \boldsymbol{z})$ a conjunction of atoms with database relations (both stored and views) and a *filter*, that is, a Boolean combination of built-in predicates such as $=$ and $<$. (Note that, by including views in the schema, we can express any SQL query in mappings, which is important from the practical point of view.)

Given a mapping $\mathcal{M}$ from a database schema to an *OWL 2 QL* TBox $\mathcal{T}$ and an instance $\boldsymbol{I}$ of this schema, the ground atoms

$$S(\boldsymbol{a}), \qquad \text{for } S(\boldsymbol{x}) \leftarrow \varphi(\boldsymbol{x}, \boldsymbol{z}) \text{ in } \mathcal{M} \text{ and } \boldsymbol{I} \models \exists \boldsymbol{z}\, \varphi(\boldsymbol{a}, \boldsymbol{z}),$$

comprise the ABox, $\mathcal{A}_{\boldsymbol{I},\mathcal{M}}$, which is called the *virtual ABox* for $\mathcal{M}$ over $\boldsymbol{I}$ [61] (this ABox is just a convenient presentational tool and does not have to be materialised by the system). We can now define *certain answers* to a CQ $\boldsymbol{q}$ over $\mathcal{T}$ linked by $\mathcal{M}$ to $\boldsymbol{I}$ as certain answers to $\boldsymbol{q}$ over $(\mathcal{T}, \mathcal{A}_{\boldsymbol{I},\mathcal{M}})$.

As an illustration, we consider a (simplified) database IMDb (`www.imdb.com/interfaces`), whose schema contains relations $\mathit{title}[m, t, y]$ with information about movies (ID, title, production year), and $\mathit{castinfo}[p, m, r]$ with information about movie casts (person ID, movie ID, person role). Thus, a data instance $\boldsymbol{I}$ of this schema may contain the tables

title

m	t	y
728	'Django Unchained'	2012

castinfo

p	m	r
n37	728	1
n38	728	1

The users are not supposed to know the structure of the database. Instead, they are given an ontology, say MO (`www.movieontology.org`), describing the application domain in terms of, for example, concepts *mo:Movie* and *mo:Person*, and roles *mo:cast* and *mo:year*:

$$\mathit{mo{:}Movie} \equiv \exists \mathit{mo{:}title}, \qquad \mathit{mo{:}Movie} \sqsubseteq \exists \mathit{mo{:}year},$$
$$\mathit{mo{:}Movie} \equiv \exists \mathit{mo{:}cast}, \qquad \exists \mathit{mo{:}cast}^- \sqsubseteq \mathit{mo{:}Person}.$$

A mapping $\mathcal{M}$ that relates the ontology terms to the database schema contains, for example, the following rules:

$$\mathit{mo{:}Movie}(m) \leftarrow \mathit{title}(m, t, y), \tag{15}$$
$$\mathit{mo{:}title}(m, t) \leftarrow \mathit{title}(m, t, y), \tag{16}$$
$$\mathit{mo{:}year}(m, y) \leftarrow \mathit{title}(m, t, y), \tag{17}$$
$$\mathit{mo{:}cast}(m, p) \leftarrow \mathit{castinfo}(p, m, r), \tag{18}$$
$$\mathit{mo{:}Person}(p) \leftarrow \mathit{castinfo}(p, m, r). \tag{19}$$

Then the virtual ABox $\mathcal{A}_{\boldsymbol{I},\mathcal{M}}$ for $\mathcal{M}$ over $\boldsymbol{I}$ consists of the ground atoms

$$\begin{array}{l} \mathit{mo{:}Movie}(728),\ \mathit{mo{:}title}(728, \text{`Django Unchained'}), \mathit{mo{:}year}(728, 2012), \\ \mathit{mo{:}Person}(\text{n37}),\ \mathit{mo{:}cast}(728, \text{n37}), \\ \mathit{mo{:}Person}(\text{n38}),\ \mathit{mo{:}cast}(728, \text{n38}). \end{array}$$

Given a CQ $\boldsymbol{q}$ and an ontology $\mathcal{T}$, one could first construct a rewriting $\boldsymbol{q}'$ of $\boldsymbol{q}$ and $\mathcal{T}$ over *arbitrary* ABoxes. The rewriting $\boldsymbol{q}'$ could then be *unfolded* into an SQL query by using *partial evaluation* [45], which exhaustively applies

SLD-resolution to $\boldsymbol{q}'$ and the mapping $\mathcal{M}$ and returns those rules whose bodies contain only database atoms. Consider the simple CQ $\boldsymbol{q}(x) = \mathit{mo{:}Movie}(x)$. An obvious rewiring of $\boldsymbol{q}$ and the TBox above (over arbitrary ABoxes) contains the following three rules:

$$\begin{align}
\boldsymbol{q}'(x) &\leftarrow \mathit{mo{:}Movie}(x), \tag{20}\\
\boldsymbol{q}'(x) &\leftarrow \mathit{mo{:}title}(x, y), \tag{21}\\
\boldsymbol{q}'(x) &\leftarrow \mathit{mo{:}cast}(x, y). \tag{22}
\end{align}$$

The unfolding applies the SLD-resolution procedure to these three rules and the mapping $\mathcal{M}$ and produces the rules:

$$\begin{align}
\boldsymbol{q}'(x) &\leftarrow \mathit{title}(x, t, y), \tag{20+15}\\
\boldsymbol{q}'(x) &\leftarrow \mathit{title}(x, t, y), \tag{21+16}\\
\boldsymbol{q}'(x) &\leftarrow \mathit{castinfo}(p, x, r). \tag{22+18}
\end{align}$$

The resulting union of Select-Project-Join queries could then be forwarded for execution to an RDBMS.

One can achieve the same result by using the tree-witness rewriting $\boldsymbol{q}_{\mathsf{tw}}$ of $\boldsymbol{q}$ and $\mathcal{T}$ over H-complete ABoxes introduced in Section 3. An obvious way to construct H-complete ABoxes is to take the composition of $\mathcal{M}$ and the inclusions in $\mathcal{T}$, that is, a mapping $\mathcal{M}^{\mathcal{T}}$ given by

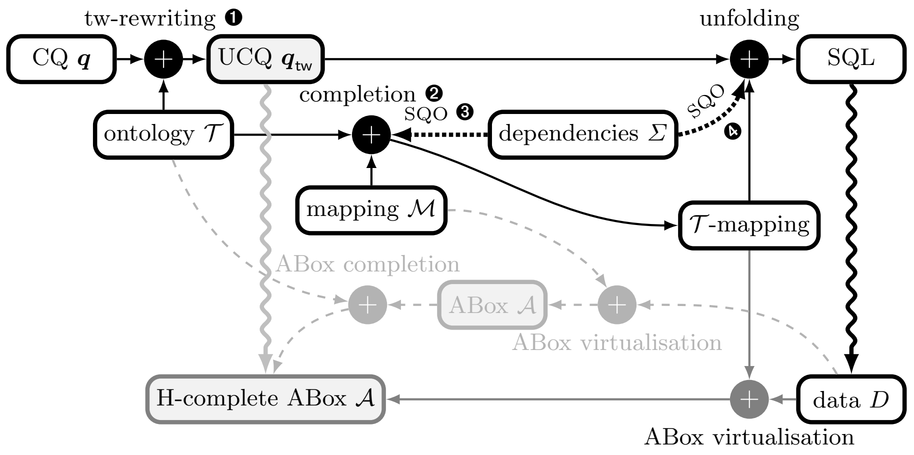
$$\begin{align*}
A(x) &\leftarrow \varphi(x, \boldsymbol{z}), && \text{if } A'(x) \leftarrow \varphi(x, \boldsymbol{z}) \in \mathcal{M} \text{ and } \mathcal{T} \models A' \sqsubseteq A,\\
A(x) &\leftarrow \varphi(x, y, \boldsymbol{z}), && \text{if } R(x, y) \leftarrow \varphi(x, y, \boldsymbol{z}) \in \mathcal{M} \text{ and } \mathcal{T} \models \exists R \sqsubseteq A,\\
P(x, y) &\leftarrow \varphi(x, y, \boldsymbol{z}), && \text{if } R(x, y) \leftarrow \varphi(x, y, \boldsymbol{z}) \in \mathcal{M} \text{ and } \mathcal{T} \models R \sqsubseteq P.
\end{align*}$$

(Recall that we do not distinguish between $P^-(y, x)$ and $P(x, y)$.) Thus, for any $\boldsymbol{I}$ and any tuple $\boldsymbol{a}$ of individuals in $\mathcal{A}_{\boldsymbol{I},\mathcal{M}}$, we have:

$$(\mathcal{T}, \mathcal{A}_{\boldsymbol{I},\mathcal{M}}) \models \boldsymbol{q}(\boldsymbol{a}) \quad \text{iff} \quad \mathcal{A}_{\boldsymbol{I},\mathcal{M}^{\mathcal{T}}} \models \boldsymbol{q}_{\mathsf{tw}}(\boldsymbol{a}). \tag{23}$$

So, to compute the answers to $\boldsymbol{q}$ over $\mathcal{T}$ linked by $\mathcal{M}$ to $\boldsymbol{I}$, one can unfold the tree-witness rewriting $\boldsymbol{q}_{\mathsf{tw}}$ over H-complete ABoxes with the help of the composition $\mathcal{M}^{\mathcal{T}}$. However, the resulting query will produce duplicating answers if the ontology axioms express the same properties of the application domain as the integrity constraints of the database [60]. For example, the IMDb schema contains a foreign key: movie ID in *castinfo* references movie ID in *title*, and therefore the unfolded rewriting above will return the same movie many times—once from *title* and once for each of the cast members of the movie in *castinfo*. Such a duplication is clearly an undesirable feature of this straightforward approach.

For this reason, before applying $\mathcal{M}^{\mathcal{T}}$ to unfold the tree-witness rewriting, *Ontop* optimises the mapping using the database integrity constraints Σ. This allows us to (a) reduce redundancy in answers, and (b) substantially shorten the SQL queries, which makes the OBDA system more efficient. The process of query rewriting and unfolding in *Ontop* with all optimisations is shown in the picture below (the dashed lines illustrate processes that do *not* take place):



The key ingredients of the architecture of *Ontop* are as follows:

- ❶ the tree-witness rewriting q_{tw} assumes the virtual ABoxes to be H-complete; it separates the topology of q from the taxonomy defined by $\mathcal{T}$, is fast in practice and produces short UCQs as demonstrated for real-world ontologies and queries [62];
- ❷ the $\mathcal{T}$-mapping combines the system mapping $\mathcal{M}$ with the taxonomy of $\mathcal{T}$ to ensure H-completeness of virtual ABoxes;
- ❸ the $\mathcal{T}$-mapping is simplified using the Semantic Query Optimisation (SQO) technique and SQL features; the $\mathcal{T}$-mapping is constructed and optimised for the given $\mathcal{T}$ and Σ only once, and is used for unfolding all rewritings q_{tw};
- ❹ the unfolding algorithm uses SQO to produce small and efficient SQL queries.

We illustrate the last three items in the remainder of this section.

6.1 $\mathcal{T}$-mappings

We say that a mapping $\mathcal{M}$ is a *$\mathcal{T}$-mapping over dependencies Σ* if the ABox $\mathcal{A}_{\boldsymbol{I},\mathcal{M}}$ is H-complete with respect to $\mathcal{T}$, for any data instance $\boldsymbol{I}$ satisfying Σ. The composition $\mathcal{M}^{\mathcal{T}}$ defined above is trivially a $\mathcal{T}$-mapping over any Σ. *Ontop* starts with $\mathcal{M}^{\mathcal{T}}$ and then applies a series of optimisations to construct a simpler $\mathcal{T}$-mapping.

Inclusion Dependencies. Suppose $\mathcal{M} \cup \{S(\boldsymbol{x}) \leftarrow \psi_1(\boldsymbol{x}, \boldsymbol{z})\}$ is a $\mathcal{T}$-mapping over Σ. If there is a more specific rule than $S(\boldsymbol{x}) \leftarrow \psi_1(\boldsymbol{x}, \boldsymbol{z})$ in $\mathcal{M}$, then $\mathcal{M}$ itself will also be a $\mathcal{T}$-mapping. To discover such 'more specific' rules, we run the standard query containment check (see, e.g., [1]) taking account of the inclusion dependencies. For example, since $\mathcal{T} \models \exists$*mo:cast* $\sqsubseteq$ *mo:Movie* in our running example, the composition $\mathcal{M}^{\mathrm{MO}}$ of the mapping $\mathcal{M}$ given above and MO contains the following rules for *mo:Movie*:

$$\begin{aligned} \textit{mo:Movie}(m) &\leftarrow \textit{title}(m,t,y), \\ \textit{mo:Movie}(m) &\leftarrow \textit{castinfo}(p,m,r). \end{aligned}$$

The latter rule is redundant since IMDb contains the foreign key (an inclusion dependency)

$$\forall m\, \big(\exists p, r\, \textit{castinfo}(p,m,r) \to \exists t, y\, \textit{title}(m,t,y)\big).$$

Disjunctions in SQL. Another way to reduce the size of a $\mathcal{T}$-mapping is to identify pairs of rules whose bodies are equivalent up to *filters with respect to constant values*. This optimisation deals with the rules introduced due to the so-called type (discriminating) attributes [20] in database schemas. For example, the mapping $\mathcal{M}$ for IMDb and MO contains six rules for sub-concepts of *mo:Person*:

$$\begin{gathered} \textit{mo:Actor}(p) \leftarrow \textit{castinfo}(c,p,m,r), (r = 1), \\ \dots \\ \textit{mo:Editor}(p) \leftarrow \textit{castinfo}(c,p,m,r), (r = 6). \end{gathered}$$

Then the composition $\mathcal{M}^{\text{MO}}$ contains six rules for *mo:Person* that differ only in the last condition $(r = k)$, $1 \leq k \leq 6$. These can be reduced to a single rule:

$$\textit{mo:Person}(p) \leftarrow \textit{castinfo}(c,p,m,r), (r = 1) \lor \dots \lor (r = 6).$$

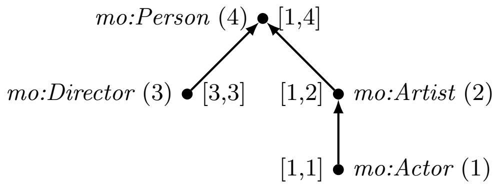
Note that such disjunctions lend themselves to efficient evaluation by RDBMSs.

Materialised ABoxes and Semantic Index. In addition to working with proper relational data sources, *Ontop* also supports ABox storage in the form of structureless *universal tables*: a binary relation $CA[\textit{id}, \textit{concept-id}]$ and a ternary relation $RA[\textit{id}_1, \textit{id}_2, \textit{role-id}]$ represent concept and role assertions. The universal tables give rise to trivial mappings, and *Ontop* implements a technique, the *semantic index* [61], that takes advantage of SQL features in $\mathcal{T}$-mappings for this scenario. The key observation is that, since the IDs in the universal tables CA and RA can be chosen by the system, each concept and role in the TBox $\mathcal{T}$ can be assigned a numeric *index* and a set of numerical *intervals* in such a way that the resulting $\mathcal{T}$-mapping contains simple SQL queries with interval filter conditions. For example, in IMDb, we have

$$\begin{aligned} \textit{mo:Actor} &\sqsubseteq \textit{mo:Artist}, \\ \textit{mo:Artist} &\sqsubseteq \textit{mo:Person}, \\ \textit{mo:Director} &\sqsubseteq \textit{mo:Person}, \end{aligned}$$

so we can choose indexes and intervals for these concepts as in the table below:

concept	index	interval
mo:Actor	1	[1,1]
mo:Artist	2	[1,2]
mo:Director	3	[3,3]
mo:Person	4	[1,4]



It can be seen that these intervals respect the concept inclusions of the TBox: e.g., [1,1] for *mo:Actor* is a subset of [1,2] for *mo:Artist*. This will generate a $\mathcal{T}$-mapping with

$$\begin{aligned}
\mathit{mo{:}Actor}(p) &\leftarrow \mathit{CA}(p, \mathit{concept\text{-}id}), (\mathit{concept\text{-}id} = 1),\\
\mathit{mo{:}Artist}(p) &\leftarrow \mathit{CA}(p, \mathit{concept\text{-}id}), (1 \leq \mathit{concept\text{-}id} \leq 2),\\
\mathit{mo{:}Director}(p) &\leftarrow \mathit{CA}(p, \mathit{concept\text{-}id}), (\mathit{concept\text{-}id} = 3),\\
\mathit{mo{:}Person}(p) &\leftarrow \mathit{CA}(p, \mathit{concept\text{-}id}), (1 \leq \mathit{concept\text{-}id} \leq 4).
\end{aligned}$$

Thus, by choosing appropriate concept and role IDs, we effectively construct H-complete ABoxes *without* the expensive forward chaining procedure (and the need to store large amounts of derived assertions). On the other hand, the semantic index $\mathcal{T}$-mappings are based on range expressions that can be evaluated efficiently by RDBMSs using standard B-tree indexes [20].

6.2 Unfolding with Semantic Query Optimisation

Ontop applies the Semantic Query Optimisation (SQO) [14] to rules obtained at the intermediate steps of unfolding. In particular, this eliminates redundant JOIN operations caused by reification of database relations by means of concepts and roles. Consider, for example, the CQ

$$\boldsymbol{q}(t, y) \leftarrow \mathit{mo{:}Movie}(m), \mathit{mo{:}title}(m, t), \mathit{mo{:}year}(m, y), (y > 2010).$$

It has no tree witnesses, and so $\boldsymbol{q}_{\mathsf{tw}} = \boldsymbol{q}$. By straightforwardly applying the unfolding to $\boldsymbol{q}_{\mathsf{tw}}$ and the $\mathcal{T}$-mapping $\mathcal{M}$ above, we obtain the query

$$\boldsymbol{q}'_{\mathsf{tw}}(t, y) \leftarrow \mathit{title}(m, t_0, y_0), \mathit{title}(m, t, y_1), \mathit{title}(m, t_2, y), (y > 2010),$$

which requires two (potentially) expensive JOIN operations. However, by using the primary key m of *title*:

$$\begin{aligned}
&\forall m \forall t_1 \forall t_2 \left(\exists y\, \mathit{title}(m, t_1, y) \land \exists y\, \mathit{title}(m, t_2, y) \to (t_1 = t_2)\right),\\
&\forall m \forall y_1 \forall y_2 \left(\exists t\, \mathit{title}(m, t, y_1) \land \exists t\, \mathit{title}(m, t, y_2) \to (y_1 = y_2)\right)
\end{aligned}$$

(a functional dependency with determinant m), we reduce two JOIN operations in the first three atoms of $\boldsymbol{q}'_{\mathsf{tw}}$ to a single atom $\mathit{title}(m, t, y)$:

$$\boldsymbol{q}''_{\mathsf{tw}}(t, y) \leftarrow \mathit{title}(m, t, y), (y > 2010).$$

Note that these two JOIN operations were introduced to reconstruct the ternary relation from its reification by means of the roles *mo:title* and *mo:year*.

The role of SQO in OBDA systems appears to be much more prominent than in conventional RDBMSs, where it was initially proposed to optimise SQL queries. While some of SQO techniques reached industrial RDBMSs, it never had a strong impact on the database community because it is costly compared to statistics- and heuristics-based methods, and because most SQL queries are written by highly-skilled experts (and so are nearly optimal anyway). In OBDA scenarios, in contrast, SQL queries are generated automatically, and so SQO becomes the only tool to avoid redundant and expensive JOIN operations.

6.3 Why Does It Work?

The techniques above prove to be very efficient in practice. Moreover, they often automatically produce queries that are similar to those written by human experts. To understand why, we briefly review the process of designing database applications [20]. It starts with conceptual modelling which describes the application domain in such formalisms as ER, UML or ORM. The conceptual model gives the vocabulary of the database and defines its semantics by means of hierarchies, cardinality restrictions, etc. The conceptual model is turned into a relational database by applying a series of *standard* procedures that *encode* the semantics of the model into a relational schema. These procedures include:

– amalgamating many-to-one and one-to-one attributes of an entity to a single n-ary relation with a primary key identifying the entity (e.g., *title* with *mo:title* and *mo:year*);
– using foreign keys over attribute columns when a column refers to the entity (e.g., *title* and *castinfo*);
– using type (discriminating) attributes to encode hierarchical information (e.g., *castinfo*).

As this process is universal, the $\mathcal{T}$-mappings created for the resulting databases are dramatically simplified by the *Ontop* optimisations, and the resulting UCQs are usually of acceptable size and can be executed efficiently by RDBMSs.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.
2. N. Alon and R. Boppana. The monotone circuit complexity of Boolean functions. *Combinatorica*, 7(1):1–22, 1987.
3. S. Arora and B. Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, New York, NY, USA, 1st edition, 2009.
4. A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev. The DL-Lite family and relations. *Journal of Artificial Intelligence Research* (*JAIR*), 36:1–69, 2009.
5. A. Artale, V. Ryzhikov, and R. Kontchakov. DL-Lite with attributes and datatypes. In *Proc. of the 20th European Conf. on Artificial Intelligence* (*ECAI 2012*), volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 61–66. IOS Press, 2012.
6. M. Bienvenu, M. Ortiz, M. Šimkus, and G. Xiao. Tractable queries for lightweight description logics. In *Proc. of the 23 Int. Joint Conf. on Artificial Intelligence* (*IJCAI 2013*), 2013.
7. M. Bienvenu, B. ten Cate, C. Lutz, and F. Wolter. Ontology-based data access: A study through disjunctive Datalog, CSP and MMSNP. arXiv:1301.6479, 2013.
8. A. Borodin, J. von zur Gathen, and J. Hopcroft. Fast parallel matrix and gcd computations. In *Proc. of the 23rd Annual Symposium on Foundations of Computer Science* (*FOCS'82*), pages 65–71, 1982.

9. A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *Journal of Web Semantics*, 14:57–83, 2012.
10. A. Calì, G. Gottlob, and A. Pieris. Towards more expressive ontology languages: The query answering problem. *Artificial Intelligence*, 193:87–128, 2012.
11. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *Journal of Automated Reasoning*, 39(3):385–429, 2007.
12. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. 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*), pages 260–270, 2006.
13. D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. View-based query answering in description logics: Semantics and complexity. *Journal of Computer and System Sciences*, 78(1):26–46, 2012.
14. U. S. Chakravarthy, D. H. Fishman, and J. Minker. *Semantic query optimization in expert systems and database systems.* Benjamin-Cummings Publishing Co., Inc., 1986.
15. C. Chekuri, and A. Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci. 239,* 2, 211–229, 2000.
16. A. Chortaras, D. Trivela, and G. Stamou. Optimized query rewriting for OWL 2 QL. In *Proc. of the 23rd Int. Conf. on Automated Deduction* (*CADE-23*), volume 6803 of *LNCS*, pages 192–206. Springer, 2011.
17. J. Dolby, A. Fokoue, A. Kalyanpur, L. Ma, E. Schonberg, K. Srinivas, and X. Sun. Scalable grounded conjunctive query evaluation over large and expressive knowledge bases. In *Proc. of the 7th Int. Semantic Web Conf.* (*ISWC 2008*), volume 5318 of *LNCS*, pages 403–418. Springer, 2008.
18. T. Eiter, M. Ortiz, M. Šimkus, T.-K. Tran, and G. Xiao. Query rewriting for Horn-SHIQ plus rules. In *Proc. of the 26th AAAI Conf. on Artificial Intelligence* (*AAAI 2012*). AAAI Press, 2012.
19. T. Eiter, M. Ortiz, and M. Šimkus. Conjunctive query answering in the description logic SH using knots. *Journal of Computer and System Sciences*, 78(1):47–85, 2012.
20. R. Elmasri and S. Navathe. *Fundamentals of Database Systems.* Addison-Wesley, 6th edition, 2010.
21. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.
22. B. Glimm, C. Lutz, I. Horrocks, and U. Sattler. Conjunctive query answering for the description logic SHIQ. *Journal of Artificial Intelligence Research* (*JAIR*), 31:157–204, 2008.
23. G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization. In *Proc. of the 27th Int. Conf. on Data Engineering* (*ICDE 2011*), pages 2–13. IEEE Computer Society, 2011.
24. G. Gottlob and T. Schwentick. Rewriting ontological queries into small nonrecursive datalog programs. In *Proc. of the 13th Int. Conf. on Principles of Knowledge Representation and Reasoning* (*KR 2012*). AAAI Press, 2012.
25. M. Grigni and M. Sipser. Monotone separation of logarithmic space from logarithmic depth. *Journal of Computer and System Sciences*, 50(3):433–437, 1995.
26. M. Grohe, T. Schwentick, and L. Segoufin. When is the evaluation of conjunctive queries tractable? In *Proc. on 33rd Annual ACM Symposium on Theory of Computing* (*STOC'01*), pages 657–666. ACM, 2001.

27. S. Heymans, L. Ma, D. Anicic, Z. Ma, N. Steinmetz, Y. Pan, J. Mei, A. Fokoue, A. Kalyanpur, A. Kershenbaum, E. Schonberg, K. Srinivas, C. Feier, G. Hench, B. Wetzstein, and U. Keller. Ontology reasoning with large data repositories. In *Ontology Management, Semantic Web, Semantic Web Services, and Business Applications*, volume 7 of *Semantic Web And Beyond Computing for Human Experience*, pages 89–128. Springer, 2008.
28. U. Hustadt, B. Motik, and U. Sattler. Reasoning in description logics by a reduction to disjunctive datalog. *Journal of Automated Reasoning*, 39(3):351–384, 2007.
29. N. Immerman. *Descriptive Complexity.* Springer, 1999.
30. S. Jukna. *Boolean Function Complexity — Advances and Frontiers*, volume 27 of *Algorithms and combinatorics.* Springer, 2012.
31. S. Kikot, R. Kontchakov, V. Podolskii, and M. Zakharyaschev. Query rewriting over shallow ontologies. In *Proc. of the 2013 International Workshop on Description Logics* (*DL 2013*), 2013.
32. S. Kikot, R. Kontchakov, and M. Zakharyaschev. On (In)Tractability of OBDA with OWL 2 QL. In *Proc. of the 2011 International Workshop on Description Logics* (*DL 2011*), volume 745 of *CEUR-WS*, 2011.
33. S. Kikot, R. Kontchakov, and M. Zakharyaschev. Conjunctive query answering with OWL 2 QL. In *Proc. of the 13th Int. Conf. on Principles of Knowledge Representation and Reasoning* (*KR 2012*). AAAI Press, 2012.
34. S. Kikot, D. Tsarkov, M. Zakharyaschev, and E. Zolin. Query answering via modal definability with FaCT++: First blood. In *Proc. of the 2013 International Workshop on Description Logics* (*DL 2013*), 2013.
35. S. Kikot, R. Kontchakov, V. V. Podolskii, and M. Zakharyaschev. Exponential lower bounds and separation for query rewriting. In *Proc. of the 39th International Colloquium on Automata, Languages, and Programming* (*ICALP 2012,*), volume 7391 of *LNCS*, pages 263–274. Springer, 2012.
36. P. G. Kolaitis and M. Y. Vardi. Conjunctive-query containment and constraint satisfaction. In *Proc. of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'98).* ACM Press, 205–213.
37. M. König, M. Leclère, M.-L. Mugnier, and M. Thomazo. 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*), volume 7497 of *LNCS*, pages 122–138. Springer, 2012.
38. R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The combined approach to query answering in DL-Lite. In *Proc. of the 12th Int. Conf. on Principles of Knowledge Representation and Reasoning* (*KR 2010*). AAAI Press, 2010.
39. R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The combined approach to ontology-based data access. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence* (*IJCAI-2011*), pages 2656–2661. AAAI Press, 2011.
40. D. Kozen. *Theory of Computation.* Springer, 2006.
41. A. Krisnadhi and C. Lutz. Data complexity in the EL family of description logics. In *Proc. of the 2007 International Workshop on Description Logics* (*DL 2007*), volume 250 of *CEUR-WS*, pages 88–99, 2007.
42. O. Lachish and R. Raz. Explicit lower bound of $4.5n - o(n)$ for Boolean circuits. In *Proc. on 33rd Annual ACM Symposium on Theory of Computing* (*STOC'01*), pages 399–408. ACM, 2001.
43. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (*PODS'02*), pages 233–246. ACM, 2002.

44. L. Libkin. *Elements Of Finite Model Theory.* Springer, 2004.
45. J.W. Lloyd and J.C. Shepherdson. Partial Evaluation in Logic Programming. *The Journal of Logic Programming*, 11(3-4):217–242, 1991.
46. C. Lutz, D. Toman, and F. Wolter. Conjunctive query answering in the description logic EL using a relational database system. In *Proc. of the 21st Int. Joint Conf. on Artificial Intelligence* (*IJCAI 2009*), pages 2070–2075, 2009.
47. C. Lutz. The complexity of conjunctive query answering in expressive description logics. In *Proc. of the 4th Int. Joint Conf. on Automated Reasoning* (*IJCAR 2008*), number 5195 in *LNAI*, pages 179–193. Springer, 2008.
48. C. Lutz, I. Seylan, D. Toman, and F. Wolter. The combined approach to OBDA: taming role hierarchies using filters. In *Proc. of the Joint Workshop on Scalable and High-Performance Semantic Web Systems* (*SSWS+HPCSW 2012*), volume 943 of *CEUR-WS*, 2012.
49. C. Lutz and F. Wolter. Non-uniform data complexity of query answering in description logics. In *Proc. of the 13th Int. Conf. on Principles of Knowledge Representation and Reasoning* (*KR 2012*). AAAI Press, 2012.
50. A. R. Meyer and M. J. Fischer. Economy of description by automata, grammars, and formal systems. In *Proc. of the 12th Annual Symposium on Switching and Automata Theory* (*SWAT'71*), pages 188–191, 1971.
51. M. Ortiz, D. Calvanese, and T. Eiter. Data complexity of query answering in expressive description logics via tableaux. *Journal of Automated Reasoning*, 41(1):61–98, 2008.
52. M. Ortiz, S. Rudolph, and M. Simkus. Query answering in the Horn fragments of the description logics SHOIQ and SROIQ. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence* (*IJCAI-2011*), pages 1039–1044. AAAI Press, 2011.
53. C. H. Papadimitriou and M. Yannakakis. On the complexity of database queries. In *Proc. of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (*PODS 97*), pages 12–19. ACM Press, 1997.
54. H. Pérez-Urbina, B. Motik, and I. Horrocks. A comparison of query rewriting techniques for DL-lite. In *Proc. of the 2009 International Workshop on Description Logics* (*DL 2007*), volume 477 of *CEUR-WS*, 2009
55. H. Pérez-Urbina, E. Rodríguez-Díaz, M. Grove, G. Konstantinidis, and E. Sirin. Evaluation of query rewriting approaches for OWL 2. In *Proc. of the Joint Workshop on Scalable and High-Performance Semantic Web Systems* (*SSWS+HPCSW 2012*), volume 943 of *CEUR-WS*, 2012.
56. A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *Journal on Data Semantics*, X:133–173, 2008.
57. R. Raz and P. McKenzie. Separation of the monotone NC hierarchy. In *Proc. of the 38th Annual Symposium on Foundations of Computer Science* (*FOCS'97*), pages 234–243, 1997.
58. R. Raz and A. Wigderson. Monotone circuits for matching require linear depth. *Journal of the ACM*, 39(3):736–744, 1992.
59. A. Razborov. Lower bounds for the monotone complexity of some Boolean functions. *Dokl. Akad. Nauk SSSR*, 281(4):798–801, 1985.
60. M. Rodríguez-Muro. *Tools and Techniques for Ontology Based Data Access in Lightweight Description Logics.* PhD thesis, KRDB Research Centre for Knowledge and Data, Free University of Bozen-Bolzano, 2010.
61. M. Rodríguez-Muro and D. Calvanese. Dependencies: Making ontology based data access work. In *Proc. of the 5th A. Mendelzon Int. Workshop on Foundations of Data Management* (*AMW 2011*), volume 749 of *CEUR-WS*, 2011.

62. M. Rodríguez-Muro, R. Kontchakov, and M. Zakharyaschev. Ontop at work. In *Proc. of the 10th OWL: Experiences and Directions Workshop* (*OWLED 2013*), 2013.
63. R. Rosati. Prexto: Query rewriting under extensional constraints in DL-Lite. In *Proc. of EWSC 2012*, volume 7295 of *LNCS*, pages 360–374. Springer, 2012.
64. R. Rosati and A. Almatelli. Improving query answering over DL-Lite ontologies. In *Proc. of the 12th Int. Conf. on Principles of Knowledge Representation and Reasoning* (*KR 2010*). AAAI Press, 2010.
65. R. Rosati. On conjunctive query answering in EL. In *Proc. of the 2007 International Workshop on Description Logics* (*DL 2007*), volume 250 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011
66. O. Savkovic and D. Calvanese. Introducing datatypes in DL-Lite. In *Proc. of the 20th European Conf. on Artificial Intelligence* (*ECAI 2012*), volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 720–725. IOS Press, 2012.
67. C. E. Shannon. The synthesis of two-terminal switching circuits. *Bell Systems Technical Journal*, 28:59–98, 1949.