

# Ontological Query Answering over Semantic Data

Giorgos Stamou<sup>(✉)</sup> and Alexandros Chortaras

School of Electrical and Computer Engineering,  
National Technical University of Athens, 15780 Zografou, Athens, Greece  
`gstam@cs.ntua.gr`

## 1 Introduction

Modern information retrieval systems advance user experience on the basis of concept-based rather than keyword-based query answering. In particular, efficient user interfaces involve terminological descriptions of the domain of interest, expressed in formal knowledge representation formalisms. Ontological representation and reasoning based on Description Logics (DLs) [7, 9, 10] play an important role, providing expressive concept-level query languages with formal semantics and reasoning support. On the other hand, most real-life applications use huge amounts of data, consequently, efficient data storage and retrieval focuses on methodologies that take advantage of the physical storage using simple rather than sophisticated data models. Trying to combine the requirements for highly expressive queries and efficient data storage, ontology-based query answering is one of the widely used approaches, especially for web applications, involving data from different sources, in different formats [25, 27, 29, 31, 32, 34].

Here, we present methods for data integration, query rewriting and query answering based on both tractable and expressive Description Logics. Specifically, we focus on semantic data representation based on relational schemas to ontology mappings, ontology-based query rewriting for tractable Description Logics and approximate query answering techniques for expressive Description Logics.

The rest of the paper is structured as follows. Section 2 presents some basics of semantic data technologies. First, relational databases are introduced as a paradigm of disk-oriented data storage that misses a vocabulary-based semantic interpretation. Then, thing descriptions that are based on terminological assertions (ABoxes) are presented as a simple way to store and access semantic data. Finally, Sect. 2 concludes with a short presentation of semantic databases that are based on relational to terminology mappings, an important technology widely used in practice, especially in cases where systems already use relational database management systems. Section 3 provides the reader with a short introduction to Description Logics and how Description Logic ontologies can be used to extend the vocabulary of data descriptions, thus providing a formal terminological data access framework. Moreover, automated ontology reasoning problems introduce the reader to ontology-based data access that is the subject of Sect. 4. Starting

from standard reasoning and instance retrieval problems, the main technologies of semantic data access are presented, with emphasis to optimised query rewriting in tractable fragments of web ontology languages. Finally, Sect. 5 briefly describes the current technologies and standards that enable ontology based data access methods, discussed in the previous sections, to be used in real web applications, while Sect. 6 concludes the paper.

## 2 Semantic Data Representation

Data access in real life applications is usually based on storage oriented technologies that focus on the efficient retrieval of information from the disks, taking advantage of the specific technological restrictions of the physical layer. Sophisticated, analytical data modelling that represents the knowledge of the domain, is usually avoided for the sake of efficiency. A typical example is the relational database management model.

**Definition 1.** Let  $\Delta_{\mathcal{V}}$  be a value domain and  $\Delta_{\mathcal{F}}$  a name domain for subsets of the values in  $\Delta_{\mathcal{V}}$ . The  $n$ -tuple  $\mathcal{D} = \langle \mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_n \rangle$  is a data structure defined on  $\Delta_{\mathcal{V}}$ ;  $\mathcal{F}_i \in \Delta_{\mathcal{F}}$  ( $i \in \mathbb{N}_n$  (we write  $\mathbb{N}_n$  for the set  $\{1, 2, \dots, n\}$ )) are the fields of  $\mathcal{D}$ ;  $v = \langle v_1, v_2, \dots, v_n \rangle$ , with  $v_i \in \Delta_{\mathcal{V}}$  ( $i \in \mathbb{N}_n$ ), is a record of  $\mathcal{D}$ . A database is a tuple  $\mathcal{B} = \langle \mathcal{D}, \mathcal{V} \rangle$ , where  $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_m\}$  is a non-empty set of data structures and  $\mathcal{V}$  a set of their records. We say that  $\mathcal{D}$  is the database schema and  $\mathcal{V}$  is the data.

An expression of the form

```
SELECT fields
FROM   structures
WHERE  conditions
```

is an SQL query against the database  $\mathcal{B}$ , where **structures** are elements of  $\mathcal{D}$  (some  $\mathcal{D}_i$ ), **fields** are some fields of the **structures** (some  $\mathcal{F}_{ij}$ ) and **conditions** are conditions for the values of these fields. The answer to the SQL query is the set that contains all the **fields** value vectors of  $\mathcal{V}$ , that satisfy **conditions**.  $\square$

Definition 1 presents a simple form of relational databases and SQL queries, covering the basic ideas. More sophisticated relational models and query languages have been introduced in the literature (see for example [2, 3]).

*Example 1.* Table 1 summarises the database schema of an example database from the cinema domain. The schema consists of five data structures providing information for directors, movies and awards. For example, the data structure **DIRECTORS**, after a unique number for each record that is usually called *primary key* (here underlined), stores the name, place of birth and a short bio of a director, while her possible movies and awards are stored in the data structures **DIRECTOR-OF** and **AWARDED-WITH**, respectively.

Some data following this database schema is given in Table 2. It contains information about two directors and two movies; it is stored in different records of

the database. For example, the movie ‘Manhattan Murder Mystery’ is described (its title is stored in the field `Title` of the first record of the data structure `MOVIES`). The movie is a ‘Comedy’ (see the value of the field `Genre` of `MOVIES`), its duration is 104 min (field `Duration`), its director is ‘Woody Allen’ (we join the information from the first record of `DIRECTOR-OF` and the field `Name` of the first record of `DIRECTORS`).

With the following SQL query  $q$ , we may find all directors of comedies.

```
SELECT DIRECTORS.Name
FROM   DIRECTORS, MOVIES, DIRECTORS-OF
WHERE  DIRECTORS.DirID = DIRECTORS-OF.DirID
        MOVIES.MovID = DIRECTORS-OF.MovID
        MOVIES.Genre = "Comedy"
```

The query involves three structures of the database, as we can see in its `FROM` clause, namely the `DIRECTORS`, the `MOVIES` and the `DIRECTORS-OF`. The query answer returns only director names, from the field `DIRECTORS.Name` (see the `SELECT` clause), however finding the correct answer set involves condition checking that needs information from the three structures (see the `WHERE` clause). In particular, the first two conditions ensure that the movies of all directors will be checked against the third condition. Thus, the relevant set of tuples is constructed with appropriate joins of `DIRECTORS`, `MOVIES` and `DIRECTORS-OF`, and then only tuples that satisfy the condition “the movie is a comedy” are selected. In this case, only M1 has `M1.Genre = “Comedy”`, thus only D1 is “director of a comedy” and thus only D1 will be an answer of the query. Formally, we write  $\text{ans}(q) = \{\langle \text{WoodyAllen} \rangle\}$ .  $\square$

Collecting information from the data is not always a straight-forward process. It presupposes a good understanding of the database schema and the value domains, and involves conditions that are difficult to be expressed in the query language. In some cases, a sophisticated information extraction procedure may be needed to mine semantically rich information out of semi-structured or unstructured data, stored in some of the fields of the database. For instance (in Example 1) the field `ShortBio` of `DIRECTORS` may contain useful information in an unstructured form.

The syntax of relational databases is suitable for efficient data storage, on the other hand it does not provide rich semantic information. For example, the position of a symbol in a statement (schema, record, field, value) is not informative

**Table 1.** Database schema for Example 1

<code>DIRECTORS</code> ( <u>DirID</u> , Name, PlaceOfBirth, ShortBio)
<code>MOVIES</code> ( <u>MovID</u> , Title, Year, Duration, Genre)
<code>AWARDS</code> ( <u>AwID</u> , Type, Category)
<code>AWARDED-WITH</code> (DirID, AwID, Year, Type)
<code>DIRECTOR-OF</code> (DirID, MovID)

**Table 2.** Example 1 database values**DIRECTORS**

DirID	Name	PlaceOfBirth	ShortBio
D1	Woody Allen	New York, USA	ex/waBio.pdf
D2	Theo Angelopoulos	Athens, Greece	ex/taCV.pdf

**MOVIES**

MovID	Title	Year	Duration	Genres
M1	Manhattan Murder Mystery	1993	104	Comedy
M2	Eternity and a day	1998	137	Drama

**AWARDS**

AwID	Type	Category
A1	BAFTA Film Award	Best Actress in a Supporting Role
A2	Cannes Film Festival	Palme d'Or

**AWARDED-WITH**

MovID	AwID	Year	Type
M1	A1	1995	Nomination
M2	A2	1998	Win

**DIRECTOR-OF**

DirID	MovID
D1	M1
D2	M2

for the nature of the entity that the specific symbol stands for (individual, concept, property, relationship, constant or datatype). An alternative of relational modelling is the object-oriented one that focus on representing *thing descriptions* in a clear syntactic form of statements classifying things to categories and describing their properties and roles (for example *manhattan* is a feature film, a comedy, has director ‘Woody Allen’ etc.). The first ingredient of this modelling is the use of an extended set of names that is clearly distinguished into three subsets, the individual, the concept and the role names. It constitutes the *vocabulary* or the *terminology* of the data representation. The second ingredient is the use of very simple syntax rules: statements *classify* individuals to concepts, based on their properties or relations to other individuals. Then, data access is based on queries that use the vocabulary to formally describe conditions and bring ‘individuals that are members of a specific class’.

**Definition 2.** Let  $\mathcal{L} = \langle \text{IN}, \text{CN}, \text{RN} \rangle$  be a vocabulary, i.e. mutually disjoint sets of names for individuals, concepts and roles of the world, respectively. We call individual equality assertion the statement  $a \approx b$ , individual inequality assertion the statement  $a \not\approx b$ , concept assertion the statement  $A(a)$  and role assertion the statement  $r(a, b)$ , where  $a, b \in \text{IN}$ ,  $A \in \text{CN}$  and  $r \in \text{RN}$ . A set of (equality, inequality, concept or role) assertions is called assertion box or simply ABox. The set of names involved in the assertions of an ABox  $\mathcal{A}$  is the signature of  $\mathcal{A}$ , written as  $\text{Sig}(\mathcal{A})$ .

Let  $\mathbf{VN}$  be a set of variable names, taking values on  $\mathbf{IN}$ . An atomic query for the ABox  $\mathcal{A}$  is an expression of the following forms (the symbol  $|$  is used to summarise alternatives):

$$q = C(a) \mid r(a, b) \quad (1)$$

$$q(x) = C(x) \mid r(x, a) \mid r(a, x) \quad (2)$$

$$q(x, y) = r(x, y) \quad (3)$$

where  $C \in \mathbf{CN}$ ,  $r \in \mathbf{RN}$ ,  $a, b \in \mathbf{IN}$ ,  $x, y \in \mathbf{VN}$  are concept, role, individual and variable names, respectively. We refer to the individual names involved in the query as constants. We refer to the set of the variables of a query  $q$  with  $\mathbf{var}(q)$ . In case the query has no variable (form 1), it is called boolean.

A conjunctive query is an expression of the form:

$$q(\mathbf{x}) = \{q_1, \dots, q_n\}, \quad (4)$$

where  $q_i$ ,  $i \in \mathbb{N}_n$  are atomic queries.  $q(\mathbf{x})$  is the head and  $\{q_1, \dots, q_n\}$  is the body of the query. We say that  $\mathbf{x}$  is the variable vector of  $q$  and its elements are called answer variables. The set of answer variables is written as  $\mathbf{avar}(q)$ . Answer variables should be also in the body (in at least one  $q_i$ ). The variables appear in the body of  $q$  and not in its head are called free variables ( $\mathbf{fvar}(q)$  is the set of free variables). The free variables that appear at least twice are called existential join variables (the set of existential join variables is written as  $\mathbf{ejvar}(q)$ ). A conjunctive query with no answer variables is called boolean.

*Example 2.* [example 1 cont.] Table 3 presents an ABox representing information from the movies domain, also contained in the database example of the previous section (see Table 2). In particular, the set of assertions

$$\mathcal{A} = \{\alpha_1, \alpha_2, \dots, \alpha_8\}$$

describes individuals like `manhattan`, `woodyAllen` and their properties, for example it is stated that `manhattan` is a `Comedy` (assertion  $\alpha_1$ ). The same information can be extracted from the database (the first record of the structure `Movies`, that has title `manhattan`, has the value `Comedy` in the field `Genre`). Additionally, the Abox contains information for the individual interrelationships, for example the assertion `hasDirector(manhattan, woodyAllen)` states that ‘Woody Allen is a director of the movie ‘Manhattan Murder Mystery’’. It is not difficult to see that this information is also in the database, in a more complicated manner, specifically from the first record of the structure `DIRECTOR-OF` we find the keys and then we get the names from the structures `DIRECTORS` and `MOVIES`. Suppose now that we would like to find all directors of comedies (the same query as in Example 1), from the information of ABox. The conjunctive query

$$q(x) = \{\text{Director}(x), \text{isDirector}(x, y), \text{Comedy}(y)\}, \quad (5)$$

uses the ABox signature, in particular the concepts `Director`, `Comedy` and the role `isDirector`. In this case,  $x$  is an answer variable and  $y$  a free variable, specifically an

existential one. Intuitively, the answer that we would get is `woodyAllen`, knowing the meaning of the vocabulary names. However, looking more carefully, this is not the case, since it is not explicitly stated in the ABox that `woodyAllen` is a `director` (although from assertion  $\alpha_2$  we can conclude that since he has directed a movie, he obviously *is* a director). In the next sections, we will see how this problem can be handled by representing domain knowledge on the basis of terminological axioms.  $\square$

**Table 3.** ABox of Example 2

$\alpha_1$	<code>Comedy(manhattan)</code>
$\alpha_2$	<code>hasDirector(manhattan, woodyAllen)</code>
$\alpha_3$	<code>FeatureFilm(manhattan)</code>
$\alpha_4$	<code>nominatedFor(manhattan, baftaBestActressSupporting)</code>
$\alpha_5$	<code>hasAward(eternityAndAday, cannesPalmeDor)</code>
$\alpha_6$	<code>woodyAllen <math>\not\approx</math> theoAngelopoulos</code>

The development of efficient disk-oriented storage and retrieval of ABoxes has been an attractive area of research during the last years, especially in the framework of the Semantic Web. As a result, several systems, known as *triple stores* have been proposed in the literature, some of them really efficient. However, even state-of-the-art triple stores face difficulties when they try to scale to big data. Moreover, in several applications, existing systems use relational database management systems and it is difficult to swap to other technologies. Thus, some applications call for vocabulary-based, semantic information access on the one hand, with relational database storage on the other hand. This requirement can be achieved with *semantic databases*, that need to connect the terms of vocabularies with the information stored in the database.

Consider the first record of the structure `MOVIES` in Table 2, that stores the fact that the movie with ID `M1` and title ‘Manhattan Murder Mystery’ is an instance of the concept `Comedy` (defined in the vocabulary). The same information is given in the ABox of Table 3 (assertion  $\alpha_1$ ):

$$\text{Comedy}(\text{manhattan}). \quad (6)$$

In order to represent the same information by simply connecting the individual described in the database with the term `Comedy`, we need to first *identify* individuals that are described in the structure `MOVIES` and then *filter* only those individuals that are instances of the class `Comedy`.

**Definition 3.** Let  $\mathcal{B}$  be a database and  $\mathcal{F}$  the set of fields of all structures of  $\mathcal{B}$ . An object identifier is a function  $\text{id}$  of any order  $n \leq |\mathcal{F}|$  defined as:

$$\text{id}(v_1, v_2, \dots, v_n) = a, \quad (7)$$

where  $v_1, v_2, \dots, v_n$  values form  $n$  fields of  $\mathcal{F}$  and  $a \in \text{IN}$  an individual name.

Similarly, a concept classifier is a function  $\text{ccl}$  of order  $m \leq |\mathcal{F}|$  defined as:

$$\text{ccl}(v_1, v_2, \dots, v_m) = C, \quad (8)$$

where  $v_1, v_2, \dots, v_m$  form  $m$  fields of  $\mathcal{F}$  and  $C \in \text{CN}$  an individual name.

Finally, a role classifier is a function  $\text{rcl}$  of order  $k \leq |\mathcal{F}|$  defined as:

$$\text{rcl}(v_1, v_2, \dots, v_k) = r, \quad (9)$$

where  $v_1, v_2, \dots, v_k$  form  $k$  fields of  $\mathcal{F}$  and  $r \in \text{RN}$  an individual name.

**Definition 4.** Let  $\mathcal{B}$  be a database and  $\mathcal{L}$  a vocabulary, with  $\text{IN}$ ,  $\text{CN}$  and  $\text{RN}$  the set of names, concepts and roles, respectively. Let also  $p(\mathbf{x})$  be an SQL query for  $\mathcal{B}$ ,  $\text{id}$ ,  $\text{ccl}$  and  $\text{rcl}$ , individual, concept and role identifier, respectively,  $\mathbf{x}$  a nonempty variable vector on  $\mathcal{V}$ , and  $q(\mathbf{y})$  an instance query. An expression of the form:

$$p(\mathbf{x}) \xrightarrow{(\text{id}, \text{ccl}, \text{rcl})} q(\mathbf{y}) \quad (10)$$

is a semantic mapping from the database  $\mathcal{B}$  to the vocabulary  $\mathcal{L}$ . A set of semantic mappings,  $\mathcal{M}$ , is a semantic mapping box or *MBox*.

The triple  $\mathcal{S} = \langle \mathcal{L}, \mathcal{B}, \mathcal{M} \rangle$  is a semantic database.

The intuitive meaning of the identifiers  $\text{id}$ ,  $\text{ccl}$  and  $\text{rcl}$  is the following. They are used to define fresh names for individuals, concept and roles respectively, if they are not already in the vocabulary. For practical reasons, these names should be intuitive for humans (i.e. informative enough for humans to refer to the specific entity) and uniquely identify the entity.  $\text{id}$ ,  $\text{ccl}$  and  $\text{rcl}$  are necessary in practice, since database IDs are not always appropriate as entity identifiers. Indeed, database IDs do not fulfil the first requirement (they are not informative for humans) and moreover, the entities described in the knowledge base are not always formally identified in the database schema (see for example the award category).

*Example 3.* [example 2 cont.] Following Examples 1 and 2, an identifier  $\text{dir}(\text{DIRECTORS.Name})$  can be defined as a function of order 1, that takes as input the value of the field **Name** of the structure **DIRECTORS** (see Table 2) and gives output object names, as:

$$\text{dir}(\text{Woody Allen}) = \text{woodyAllen}.$$

Moreover, the function  $\text{mov}$  can be defined similarly, as a function of order 1, taking as input the title of a movie (or parts of it for simplicity reasons) to define movie names, giving at the output for example **manhattan** as an movie identifier for the movie ‘Manhattan Murder Mystery’. If the context suggest for more information in the name to identify, the function  $\text{mov}$  could do so by concatenating the title and the first release date:

$$\text{mov}(\text{MOVIES.Title}, \text{MOVIES.Year})$$

giving the output:

`mov(Manhattan Murder Mystery, 1993) = manhattanMurderMystery1993.`

Finally, we define the semantic mapping

$$\begin{aligned}
 m : & \text{ SELECT DIRECTORS.Name} \\
 & \text{ FROM DIRECTORS, MOVIES, DIRECTORS-OF} \\
 & \text{ WHERE DIRECTORS.DirID = DIRECTORS-OF.DirID} \\
 & \text{ MOVIES.MovID = DIRECTORS-OF.MovID} \\
 & \text{ MOVIES.Genre = "Comedy"} \\
 \mapsto & \text{ DirectorOfComedy(dir}(x)),
 \end{aligned} \tag{11}$$

that maps to the concept `DirectorOfComedy`. In this case, we can get the assertion

`DirectorOfComedy(woodyAllen),`

since Woody Allen is the only answer to the SQL query of the mapping (11).  $\square$

Definition 4 presents a simple form of semantic mappings. More general mapping frameworks, especially mapping relational databases to terminologies have been studied in the literature, especially in the framework of data integration [4–6].

### 3 Ontological Data Descriptions

Information retrieval using vocabularies and semantic data forms a basis for user friendly systems, however it does not meet *all* user requirements. Users sometimes expect that the system will employ logical procedures during the retrieval process in order to be more precise and effective. For example, users expect that ‘directors’ should be answers to a query that asks for ‘creators’, simply because ‘all directors are creators’. Formal knowledge representation can be very helpful within this context, enriching the vocabularies with additional terms (not directly mapped to the data, but connected with other entities that are mapped), and expressing the restrictions of the domain that are helpful during the data retrieval process. Ontologies expressed in Description Logics play an important role here, as a rich terminological knowledge representation framework, supported by efficient automated reasoning services [7–12].

**Definition 5.** Let  $IN$ ,  $CN$  and  $RN$  be mutually disjoint sets of individual, concept and role names, respectively.

A role  $r \in RN$  is a named role expression or atomic role. Let  $r, s$  be atomic roles. The expressions  $r^{-}$ ,  $r \circ s$ , recursively defined using the role constructors  $^{-}$  (inverse role constructor) and  $\circ$  (role composition constructor), are role expressions or complex roles or simply roles. Moreover, we use the symbol  $U$  for the universal role.



A concept  $C \in \text{CN}$  is a named concept expression or atomic concept. Let  $C, D$  be atomic concepts,  $r$  an atomic role,  $a$  an individual name and  $n$  a natural number. The expressions  $\neg C$ ,  $C \sqcap D$ ,  $C \sqcup D$ ,  $\exists r.C$ ,  $\forall r.C$ ,  $\geq nr.C$ ,  $\leq nr.C$ ,  $\{a\}$ , recursively defined using the concept constructors  $\neg$  (negation),  $\sqcap$  (conjunction),  $\sqcup$  (disjunction),  $\exists$  (existential),  $\forall$  (universal),  $\geq n$  (at-least number restrictions),  $\leq n$  (at-most number restrictions),  $\{ \}$  (nominal), are called concept expressions or complex concepts or simply concepts. Moreover,  $\top$  (named Top) and  $\perp$  (named Bottom) are concepts. Finally, Self can be used in expressions of the form  $\exists r.\text{Self}$ .

An expression of the form  $C \sqsubseteq D$  ( $C \equiv D$ ) is a concept subsumption axiom (concept equivalence axiom). Similarly, an expression of the form  $r \sqsubseteq s$  ( $r \equiv s$ ) is a role subsumption axiom (role equivalence axiom).

A set of concept or role subsumption or equivalence axioms is a terminological box or TBox or ontology. The individual, concept and role names used in the axioms of a TBox  $\mathcal{T}$  is the signature of  $\mathcal{T}$ , written as  $\text{Sig}(\mathcal{T})$ .

The tuple  $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ , where  $\mathcal{T}$  is a TBox and  $\mathcal{A}$  an ABox, with  $\text{Sig}(\mathcal{T}), \text{Sig}(\mathcal{A}) \subseteq \text{IN} \cup \text{CN} \cup \text{RN}$  is a knowledge base or simply knowledge, with signature  $\text{Sig}(\mathcal{K}) = \text{Sig}(\mathcal{T}) \cup \text{Sig}(\mathcal{A})$ .

*Example 4.* The set of axioms  $\mathcal{T} = \{\tau_1, \tau_2, \dots, \tau_{13}\}$ , where

- $\tau_1.$  Director  $\sqsubseteq$  Creator,
- $\tau_2.$  Movie  $\equiv$  Film,
- $\tau_3.$  Director  $\sqcap$  Movie  $\sqsubseteq \perp$ ,
- $\tau_4.$  Movie  $\equiv$  ShortFilm  $\sqcup$  FeatureFilm,
- $\tau_5.$  FeatureFilm  $\equiv$  Film  $\sqcap$  LongFilm,
- $\tau_6.$  FeatureFilm  $\equiv$  Film  $\sqcap \neg \text{ShortFilm}$ ,
- $\tau_7.$  Director  $\equiv \exists \text{isDirector.Movie}$ ,
- $\tau_8.$  Movie  $\sqsubseteq \forall \text{hasDirector.Director}$ ,
- $\tau_9.$  MultiAwardWinning  $\equiv \geq 3 \text{hasAward.MajorAward}$ ,
- $\tau_{10}.$   $\top \sqsubseteq \forall \text{hasDirector.Director}$ ,
- $\tau_{11}.$  hasDirector  $\sqsubseteq$  hasCreator,
- $\tau_{12}.$  isDirector  $\equiv \text{hasDirector}^-$ ,
- $\tau_{13}.$  hasCollaboration  $\sqsubseteq \text{isDirector} \circ \text{hasActor}$ ,

is a TBox, with signature

$$\text{Sig}(\mathcal{T}) = \{\text{Director}, \text{Creator}, \text{Movie}, \text{Film}, \text{ShortFilm}, \text{FeatureFilm}, \text{LongFilm}, \text{MultiAwardWinning}, \text{MajorAward}, \text{isDirector}, \text{hasDirector}, \text{hasAward}, \text{hasCreator}, \text{hasCollaboration}, \text{hasActor}, \text{hasRunningTime}\}.$$

Axioms  $\tau_1, \tau_3, \tau_8$  and  $\tau_{10}$  are concept inclusion axioms,  $\tau_2, \tau_4$ – $\tau_7$  and  $\tau_9$  are concept equivalence axioms,  $\tau_{11}$  and  $\tau_{13}$  are role inclusion axioms and  $\tau_{12}$  is a role equivalence axiom.  $\square$

Ontologies and knowledge bases are practically useful because reasoning services can extract logical entailments of their axioms, by applying simple semantic

rules. For example, based on the TBox of Example 4 we can conclude using simple reasoning rules that if an individual  $a$  is director of an individual  $b$  that has actor an individual  $c$ , then  $a$  is a director,  $b$  is a movie and  $a$  has a collaboration with  $c$ . Consequences like the above, are based on formal semantics of axioms and assertions.

**Definition 6.** Let  $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$  be a knowledge base, with signature  $\text{Sig}(\mathcal{K}) \subseteq \text{IN} \cup \text{CN} \cup \text{RN}$ , where  $\text{IN}$ ,  $\text{CN}$ ,  $\text{RN}$  mutually disjoint sets of individual, concept and role names, respectively. Interpretation of the knowledge, is a tuple  $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ , where  $\Delta^{\mathcal{I}}$  a nonempty (possibly infinite) set of objects, called domain and  $\mathcal{I}$  the interpretation function, that maps elements of  $\mathcal{K}$  to  $\Delta^{\mathcal{I}}$  structures as follows:

- Individuals are interpreted as elements of  $\Delta^{\mathcal{I}}$ , i.e. if  $a \in \text{IN}$ , then  $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ .
- Atomic concepts are interpreted as subsets of  $\Delta^{\mathcal{I}}$ , i.e. if  $A \in \text{CN}$ , then  $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ .
- Atomic roles are interpreted as subsets of  $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ , i.e. if  $r \in \text{RN}$ , then  $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ .
- Complex roles are interpreted as subsets of  $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$  ( $(x, y) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ ) recursively on their structure, as follows:
  - For every  $x, y \in \Delta^{\mathcal{I}}$  it is  $(x, y) \in \text{U}^{\mathcal{I}}$ .
  - $(x, y) \in (r^-)^{\mathcal{I}}$  if and only if  $(y, x) \in r^{\mathcal{I}}$ .
  - $(x, y) \in (r \circ s)^{\mathcal{I}}$  if and only if there exists  $z \in \Delta^{\mathcal{I}}$  such that  $(x, z) \in r^{\mathcal{I}}$  and  $(z, y) \in s^{\mathcal{I}}$ .
- Complex concepts are interpreted as subsets of  $\Delta^{\mathcal{I}}$ , recursively on their structure, as follows:
  - For every  $x \in \Delta^{\mathcal{I}}$  it is  $x \in \top^{\mathcal{I}}$ .
  - There does not exist  $x \in \Delta^{\mathcal{I}}$  such that  $x \in \perp^{\mathcal{I}}$ .
  - $x \in (\neg C)^{\mathcal{I}}$  if and only if  $x \notin C^{\mathcal{I}}$ .
  - $x \in (C \sqcap D)^{\mathcal{I}}$  if and only if  $x \in C^{\mathcal{I}}$  and  $x \in D^{\mathcal{I}}$ .
  - $x \in (C \sqcup D)^{\mathcal{I}}$  if and only if  $x \in C^{\mathcal{I}}$  or  $x \in D^{\mathcal{I}}$ .
  - $x \in (\exists r.C)^{\mathcal{I}}$  if and only if there exists  $y \in \Delta^{\mathcal{I}}$ , such that  $(x, y) \in r^{\mathcal{I}}$  and  $y \in C^{\mathcal{I}}$ .
  - $x \in (\forall r.C)^{\mathcal{I}}$  if and only if for every  $y \in \Delta^{\mathcal{I}}$  with  $(x, y) \in r^{\mathcal{I}}$ , it is  $y \in C^{\mathcal{I}}$ .
  - $x \in (\geq n \, r.C)^{\mathcal{I}}$  if and only if there exist at least  $n$  different elements  $y_1, \dots, y_n$  of  $\Delta^{\mathcal{I}}$  such that  $(x, y_i) \in r^{\mathcal{I}}$  and  $y_i \in C^{\mathcal{I}}$ ,  $i \in \mathbb{N}_n$ .
  - $x \in (\leq n \, r.C)^{\mathcal{I}}$  if and only if there exist at most  $n$  different elements  $y_1, \dots, y_n$  of  $\Delta^{\mathcal{I}}$ , such that  $(x, y_i) \in r^{\mathcal{I}}$  and  $y_i \in C^{\mathcal{I}}$ ,  $i \in \mathbb{N}_n$ .
  - $x \in (\exists r.\text{Self})^{\mathcal{I}}$  if and only if  $(x, x) \in r^{\mathcal{I}}$ .
  - $x \in \{a\}^{\mathcal{I}}$  if and only if  $x = a^{\mathcal{I}}$ .

The interpretation  $\mathcal{I}$  of the knowledge  $\mathcal{K}$ , satisfies:

- a concept assertion  $C(a)$  of  $\mathcal{A}$  if and only if  $a^{\mathcal{I}} \in C^{\mathcal{I}}$ ,
- a role assertion  $r(a, b)$  of  $\mathcal{A}$  if and only if  $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ ,
- an individual equality  $a \approx b$  of  $\mathcal{A}$  if and only if  $a^{\mathcal{I}} = b^{\mathcal{I}}$ ,
- a concept inequality  $a \not\approx b$  of  $\mathcal{A}$  if and only if  $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ .

An interpretation  $\mathcal{I}$  satisfies an ABox  $\mathcal{A}$  if and only if it satisfies all of its assertions. In this case, we say that  $\mathcal{I}$  is a model of  $\mathcal{A}$ .

An interpretation  $\mathcal{I}$  of a knowledge  $\mathcal{K}$ , satisfies:

- a concept subsumption axiom  $C \sqsubseteq D$  if and only if  $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ ,
- a concept equivalence axiom  $C \equiv D$  if and only if  $C^{\mathcal{I}} = D^{\mathcal{I}}$ ,
- a role subsumption axiom  $r \sqsubseteq s$  if and only if  $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$ ,
- a role equivalence axiom  $r \equiv s$  if and only if  $r^{\mathcal{I}} = s^{\mathcal{I}}$ .

The interpretation  $\mathcal{I}$  satisfies the TBox  $\mathcal{T}$  if and only if it satisfies all of its axioms. Then, we say that  $\mathcal{I}$  is a model of  $\mathcal{T}$ . Additionally,  $\mathcal{I}$  satisfies a concept  $C$  of  $\mathcal{T}$ , if and only if  $C^{\mathcal{I}}$  is nonempty.

Finally, the interpretation  $\mathcal{I}$  satisfies the knowledge  $\mathcal{K}$  (is a model of  $\mathcal{K}$ ), if and only if it is a model of both its Abox and Tbox. We say that  $\mathcal{K}$  is satisfiable if there exists a model of  $\mathcal{K}$ .

**Table 4.** Complex concept and role semantics

Constructor	Syntax	Semantics
Top	$\top$	$\Delta^{\mathcal{I}}$
Bottom	$\perp$	$\emptyset$
Negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Existential	$\exists r.C$	$\{x \mid \exists y \text{ such that } r(x, y)\}$
For-all	$\forall r.C$	$\{x \mid \forall y \text{ with } r(x, y) \text{ it is } C(y)\}$
At-least $n$	$\geq n.C$	$\{x \mid \exists y_1, \dots, y_n \text{ with } y_i \neq y_j, r(x, y_i), i, j \in \mathbb{N}_n\}$
At-most $n$	$\leq n.C$	$\{x \mid \nexists y_1, \dots, y_{n+1} \text{ with } y_i \neq y_j, r(x, y_i), i, j \in \mathbb{N}_{n+1}\}$
Reflexivity	$\exists r.\text{Self}$	$\{x \mid \text{it is } r(x, x)\}$
Nominals	$\{a\}$	$a^{\mathcal{I}}$
Universal role	$\mathbf{U}$	$\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Inverse role	$r^{-}$	$\{(x, y) \mid \text{it is } r(y, x)\}$
Role composition	$r \circ s$	$\{(x, y) \mid \exists z \text{ such that } r(x, z) \text{ and } s(z, y)\}$

A great advantage of DLs is that the expressive power of concept and role constructors can be used in a pay-as-you-go manner. The more expressive the language that is used in the axioms, the more difficult the problem of automated reasoning. Less expressive DLs are supported by very efficient reasoning services and are used in applications that need fast response, while more expressive ones are used in applications where sophisticated reasoning is needed. We say that the former DLs are of *low expressivity*, while the latter are *very expressive*. An example of a very expressive DL is *SRQIQ* [16] underpinning the Web

**Table 5.** Semantics of concept and role axioms

Axiom	Syntax	Model condition
Concept assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
Role assertion	$r(a, b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$
Individual equality	$a \approx b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$
Individual inequality	$a \not\approx b$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$
Concept subsumption	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
Concept equivalence	$C \equiv D$	$C^{\mathcal{I}} = D^{\mathcal{I}}$
Role subsumption	$r \sqsubseteq s$	$r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$
Role equivalence	$r \equiv s$	$r^{\mathcal{I}} = s^{\mathcal{I}}$

Ontology Language (OWL 2) [56], that uses all the constructors shown in Table 4. Examples of tractable DLs underpinning some tractable fragments of OWL 2, are DL-Lite [25, 26],  $\mathcal{ELHI}$  [27], and the DLP [17].

Automated reasoning for DLs has been studied by many researchers over the past 20 years [9, 13–15]. The work mainly focused to the development of sophisticated algorithms and optimised systems for standard reasoning problems, directly following the semantics. Finally, several systems have been implemented for DL reasoning [18–21].

**Definition 7.** Let  $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$  be a knowledge base with signature  $\text{Sig}(\mathcal{K}) \subseteq \text{IN} \cup \text{CN} \cup \text{RN}$ , where IN, CN, RN mutually disjoint sets of individual, concept and role names, respectively. Let  $C \in \text{CN}$ ,  $\alpha$  an assertion with  $\text{Sig}(\alpha) \subseteq \text{Sig}(\mathcal{K})$  and  $\tau$  an axiom with  $\text{Sig}(\tau) \subseteq \text{Sig}(\mathcal{K})$ .

- Concept satisfiability The concept  $C$  is satisfiable in  $\mathcal{T}$ , if and only if  $C$  is satisfied in some model of  $\mathcal{T}$ .
- Logical entailment of axioms The axiom  $\tau$  is a logical entailment of  $\mathcal{T}$  (we write  $\mathcal{T} \models \tau$ ), if and only if  $\tau$  is satisfied in every model of  $\mathcal{T}$ .
- ABox consistency The ABox  $\mathcal{A}$  is consistent w.r.t. the TBox  $\mathcal{T}$ , if and only if there exists a model of  $\mathcal{T}$  that is also a model of  $\mathcal{A}$ .
- Logical entailment of assertions The assertion  $\alpha$  is a logical entailment of  $\mathcal{K}$  (we write  $\mathcal{K} \models \alpha$ ), if and only if  $\alpha$  is satisfied in every model of  $\mathcal{K}$ .

The above problems are not independent. An algorithm solving one of them can be used to solve others, as suggested by the following proposition.

**Proposition 1.** Let  $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$  be a knowledge base,  $C, D$  two concepts and  $a$  an individual of  $\mathcal{K}$ .

1.  $C$  is satisfiable in  $\mathcal{T}$ , if and only if  $\mathcal{T} \models C \sqsubseteq \perp$ .
2. It is  $\mathcal{T} \models C \sqsubseteq D$ , if and only if the concept  $C \sqcap \neg D$  is non-satisfiable in  $\mathcal{T}$ .
3. It is  $\mathcal{K} \models C(a)$ , if and only if the ABox  $\mathcal{A} \cup \{\neg C(a)\}$  is inconsistent w.r.t.  $\mathcal{T}$ .

4.  $\mathcal{T}$  entails that  $C$  is satisfiable in  $\mathcal{T}$ , if and only if the ABox  $\{C(b)\}$  is consistent w.r.t. the TBox  $\mathcal{T}$ , where  $b$  is a fresh individual name.

The reasoning problems mentioned above are useful in applications where data retrieval is based on terminological description of the domain. Among them, the problem of logical entailment of assertions is of great importance. Indeed, it is not difficult to imagine a (naive) algorithm that, given a concept  $C \in \text{CN}$  checks for every individual  $a \in \text{IN}$  whether  $\mathcal{K} \models C(a)$  holds or not, thus collecting all instances of  $C$ . This is a simple way to solve a problem that is called *instance retrieval*, which is actually semantic query answering for very simple query languages (only atomic concepts or roles). In the next, we will see how this problem can be efficiently solved, depending on the DL expressivity, as well as how it can be extended to conjunctive query answering over DL terminologies.

## 4 Semantic Data Access

Consider a semantic database  $\mathcal{S} = \langle \mathcal{L}, \mathcal{B}, \mathcal{M} \rangle$ , where  $\mathcal{L}$  is a vocabulary with  $\text{IN}$ ,  $\text{CN}$ ,  $\text{RN}$  the sets of individuals, concept and role names respectively,  $\mathcal{B}$  a database, and  $\mathcal{M}$  a semantic mapping box. Assume also that we have a TBox  $\mathcal{T}$  with  $\text{sig}(\mathcal{T}) \subseteq \text{IN} \cup \text{CN} \cup \text{RN}$  and that we pose a conjunctive query of the form

$$q(\mathbf{x}) = \{q_1, \dots, q_n\},$$

where  $q_1, \dots, q_n$  are atomic queries for concepts and roles of  $\text{CN}$  and  $\text{RN}$ . Since no explicit ABox is given, intuitively, to answer this query we need to find the elements in  $\mathcal{B}$  that satisfy the constraints of query  $q$  and TBox  $\mathcal{T}$ , according to the mappings defined in  $\mathcal{M}$ . Thus, implicitly we assume that we have a knowledge base  $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$  with an ABox  $\mathcal{A}$ , implicitly encoded in  $\mathcal{S}$ .

For answering such conjunctive queries in practice, two different strategies have been suggested. The first approach, following the above intuition, tries to solve the problem by *converting the semantic database into a knowledge base*, i.e. by computing explicitly the missing  $\mathcal{A}$  from  $\mathcal{S}$  using a forward-chaining procedure. In this case, the problem of answering a query over the semantic database can be solved as an instance retrieval problem. The second approach tries to solve the problem in a backward-chaining manner by *converting the conjunctive query to an SQL query*, using the TBox and the mappings. In this case, the data retrieval problem is solved as a database access problem of answering SQL queries. Both approaches have advantages and disadvantages, have been studied extensively in the literature and have been used in several systems.

The process of converting a semantic database into a knowledge base is relatively simple. Intuitively it can be described as follows: Starting from the mappings, we execute all SQL queries contained in the mappings, and record each answer in an ABox  $\mathcal{A}$ , specifically generated for this purpose. This conversion can be performed efficiently, and it can be proved that it does not affect the soundness and completeness of the answering system.

The second approach is based on converting the conjunctive query, that is expressed in terms of the TBox, into an SQL query, expressed in terms of the underlying database schema. In particular, using the mappings defined in  $\mathcal{M}$ , we check if each atomic query of the conjunctive query  $q$  is the righthand side of some element in  $\mathcal{M}$ . If this is the case, then the respective SQL query is transformed, so that its **SELECT** part returns the identifiers of the objects of the variables of the database structure that correspond to query answer variables. To complete the construction of the final SQL query from the individual SQL queries specified in  $\mathcal{M}$ , we join their **FROM** and **WHERE** clauses, and add in the **WHERE** clause any necessary additional conditions for the joining, in the case the same variable belongs to two different atomic queries. This approach does not explicitly construct the ABox  $\mathcal{A}$ .

In both approaches, we will not get the full, sound and complete, solution to the semantic data retrieval problem, if we limit ourselves to retrieving simply the instances of the atomic queries (in the first case) or to answering the SQL query (in the second case). This is because, this process does not take into account the information of the axioms in the TBox  $\mathcal{T}$ , which encode additional knowledge both explicit and implicit. If we take into account also  $\mathcal{T}$ , a right answer to the query should be compatible with a model of the implicit knowledge base  $\langle \mathcal{T}, \mathcal{A} \rangle$ . However, unlike relational databases, a knowledge base may in general have many models. So, we can consider as right answers to the query those answers that depend only on the information contained in  $\mathcal{T}$ , i.e. those that are obtained by evaluating the query over a database compatible with  $\mathcal{T}$ , but independently of which is the actually chosen database [34]. This leads to the following definition:

**Definition 8.** Let  $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$  be a knowledge base and  $q(\mathbf{x})$  a conjunctive query for the particular knowledge base, where  $\mathbf{x}$  (of size  $n$ ) are the answer variables. Let also  $\mathcal{I}$  be an interpretation for knowledge base  $\mathcal{K}$ . An answer  $q^{\mathcal{I}}$  to the conjunctive query  $q$  in  $\mathcal{I}$  is the set of the individual vectors  $\mathbf{a}$ , of size  $n$ , for which we have that  $\mathcal{I} \models q(\mathbf{a})$ .

A vector of individuals  $\mathbf{c}$  of size  $n$  is a certain answer to  $q$  for  $\mathcal{K}$ , if and only if for each model  $\mathcal{I}$  of  $\mathcal{K}$  we have  $\mathbf{c} \in q^{\mathcal{I}}$ . The set of certain answers to the conjunctive query  $q$  is denoted by  $\text{cert}(q, \mathcal{K})$ .

Based on the above definition, in order to solve the query answering problem, we need to find the set of certain answers to  $q$  for  $\langle \mathcal{T}, \mathcal{A} \rangle$ , i.e., not only the answers obtained from the ABox directly derivable from  $\mathcal{S}$ , but also the answers that are obtained though the assertions that are consequences of the ABox, using the axioms in TBox. Finding all these assertions is a reasoning problem, which can be solved in two ways:

The first method, called *materialization*, or *saturation* is query-independent and is performed as a data preprocessing step. In particular, it uses the TBox  $\mathcal{T}$  to extend the ABox  $\mathcal{A}$  that has been derived directly from  $\mathcal{S}$ , making thus explicit all the implicit knowledge that can be derived from  $\mathcal{S}$  and  $\mathcal{T}$ . In this way, when answering the query, the TBox is not needed any more, since its contribution has already been recorded by extending the ABox.

The second approach, called *query rewriting*, follows a different strategy and does not modify the database. Instead, it starts from the query and extends it using the TBox, trying to encode in this extension all the implicit knowledge related with the atomic queries that appear in the query. Then, the extended query is executed against the database, without needing the TBox. Of particular practical significance are the cases where the extended query can be expressed as an SQL query, so that it can be directly executed over the underlying relational database  $\mathcal{B}$ .

#### 4.1 Implicit Knowledge Materialization

In the materialization approach introduced above, the contribution of the TBox to answering a query is determined through the expansion of the ABox, i.e. through computing and recording all relevant assertions. For this process to be effective, it should be guaranteed that all of the implicit knowledge are converted to explicit knowledge. In this case, the TBox is not any more necessary for finding the certain answers to a query, and the query answering process can be performed by simply retrieving the relevant individuals from the ABox. Materialisation is very effective in some Description Logics, but impossible to be applied on others [22]. In general, materialization applies a set of rules that encode the consequences of the TBox axioms. These rules, depending on the axiom expressions, add assertions for the ABox individuals, and if necessary, add also new individuals in the ABox.

*Example 5.* Consider a TBox  $\mathcal{T}$  that contains only the axiom

$$\tau_1. \text{Director} \sqsubseteq \text{Creator},$$

that for the database  $\mathcal{B}$  of Table 2  $\mathcal{M}$  contains only the mapping

$$\begin{aligned} m_1: & \text{SELECT DIRECTORS.Name} \\ & \text{FROM DIRECTORS} \\ & \mapsto \text{Director}(\text{dir}(x)), \end{aligned}$$

where  $\text{dir}$  is the object identity function, and that we want to answer the query

$$q(x) = \{\text{Creator}(x)\}.$$

In order to answer  $q$ , we will first construct an ABox  $\mathcal{A}$  that corresponds to the materialization of  $\mathcal{B}$  using  $\mathcal{M}$ . This results in  $\mathcal{A} = \{\alpha_1, \alpha_2\}$ , where  $\alpha_1$  is  $\text{Director}(\text{woodyAllen})$  and  $\alpha_2$  is  $\text{Director}(\text{theoAngelopoulos})$ . Next, we have to extend  $\mathcal{A}$ , using the axioms of  $\mathcal{T}$ . This results in adding to  $\mathcal{A}$  the assertions  $\alpha_3$ .  $\text{Creator}(\text{woodyAllen})$  and  $\alpha_4$ .  $\text{Creator}(\text{theoAngelopoulos})$ ; these are the results of applying  $\tau_1$  on  $\alpha_1$  and  $\alpha_2$ . In this way, the certain answers for the new knowledge base  $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}' \rangle$ , where  $\mathcal{A}' = \{\alpha_1, \dots, \alpha_4\}$ , are the following:  $\text{cert}(q) = \{\langle \text{woodyAllen} \rangle, \langle \text{theoAngelopoulos} \rangle\}$ .

Next, assume that we add to  $\mathcal{T}$  the axioms

$$\begin{aligned}\tau_2. \text{ Director} &\sqsubseteq \exists \text{ isDirector.Movie} \\ \tau_3. \text{ Movie} &\sqsubseteq \exists \text{ hasDirector.Director}\end{aligned}$$

and that we want to answer the same query  $q$ . In this case, when trying to expand the initial  $\mathcal{A}$ , we face the following problem: When we try to apply the axiom  $\tau_2$  on **woodyAllen**, which according to  $\alpha_1$  is an instance of **Director**, we have to add to the knowledge a new individual, say **mov1**, which will be an instance of **Movie** and will be connected to **woodyAllen** through the role **isDirector**. Thus, the following assertions will be added:  $\alpha_5$ . **Movie(mov1)** and  $\alpha_6$ . **isDirector(woodyAllen, mov1)**. Then we can apply the axiom  $\tau_3$  on **mov1**, and obtain the assertions  $\alpha_7$ . **Director(dir1)** and  $\alpha_8$ . **hasDirector(mov1, dir1)**, after adding the new individual **dir1**, in a way similar to **mov1**. After the addition of the new assertions, we can apply axiom  $\tau_2$  on the new individual **dir1**. This leads to the addition of a new individual **mov2** for which we will then have to apply again axiom  $\tau_3$  to add a new individual **dir2**, etc. Hence, the process will not terminate.

As the above example shows, materialization cannot be applied to expressive Description Logics because it does not always terminate. Problems arise also in ontology languages with disjunction, which leads to alternative ABoxes, which are difficult to handle. For these reasons, materialization can be applied more successfully to Description Logics that do not allow representation of disjunctive knowledge and in which no references to new individuals are needed. Languages that do not allow representation of disjunctive knowledge have been studied extensively within the first-order logic framework and are known as Horn Logic. Accordingly, the Description Logics exhibiting similar properties are called Horn Description Logics and play an important role in developing practical semantic retrieval systems.

In such Description Logics, the inference procedure needed to perform the materialization can be encoded in a set of inference rules: the initial ABox is saturated by repeatedly applying the rules to the available data until no fresh data are derived. Optimizations can be applied on this naive approach, so as to avoid redundant derivations. The OWL 2 RL profile is a subset of OWL 2 designed specifically to allow reasoning using such a rule-based implementation.

A rule language that is commonly used to capture the consequences of TBox axioms is *datalog* [2], which is particularly useful because it underlies deductive databases.

**Definition 9.** *[adapted from [2]] A datalog rule is an expression of the form*

$$R_1(\mathbf{u}_1) \leftarrow R_2(\mathbf{u}_2), \dots, R_n(\mathbf{u}_n),$$

where  $n \geq 1$ ,  $R_1, \dots, R_n$  are relation names and  $\mathbf{u}_1, \dots, \mathbf{u}_n$  are tuples of appropriate arities. Each variable occurring in  $\mathbf{u}_1$  must occur in at least one of  $\mathbf{u}_2, \dots, \mathbf{u}_n$ . A datalog program is a finite set of datalog rules.



Using datalog, the TBox is converted into a datalog program that is executed to generate the inferred assertions. There are several systems that perform rule-based reasoning and materialization over OWL 2 RL knowledge bases using datalog or other rule-based methodologies, such as Apache Jena, Oracle 11g, GraphDB (formerly OWLIM) [24] and RDFox [23].

## 4.2 Query Rewriting

The query rewriting approach is based on the premise that the answers to a conjunctive query are affected by the axioms of the TBox that are related in some way with the query. Since the conjunctive query is a set of atomic queries, it is obvious, as a starting point, that any axiom that involves a concept or a role that is used in the conjunctive query is related with the query. Of course, other axioms may also affect the answers to the query. Hence, if we could encode in some way in the query itself, or to an expansion of it, the way that all relevant axioms affect the answers to the query, then we could possibly ignore the TBox.

*Example 6.* Consider the TBox  $\mathcal{T} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$  where

- $\tau_1.$  Director  $\sqsubseteq$  Creator,
- $\tau_2.$  Movie  $\sqsubseteq$  Film,
- $\tau_3.$  isDirector  $\sqsubseteq$  isCreator,
- $\tau_4.$  MovieDirector  $\sqsubseteq$   $\exists$ isDirector.Movie.

First, assume that we want to answer the query  $q_1(x) = \{\text{Creator}(x)\}$ . Since  $\tau_1$  tells us that all Directors are Creators, the axiom is relevant to the query. In order to ‘encode’ it into the query, we need to produce the additional query  $q_{1a}(x) = \{\text{Director}(x)\}$ , by ‘replacing’ Creator with its subconcept Director, so as to guarantee that we will retrieve also the individuals that have been declared to be directors but not creators. So, we can consider as answers to the original query  $q_1$  the answers to both  $q_1$  and  $q_{1a}$ , i.e. the answers to the query set  $Q = \{q_1, q_{1a}\}$ . Given  $Q$ , we do not need to consider any more  $\tau_1$  when answering the query. The encoding of  $\tau$  into  $q_1$  gave rise to the set of conjunctive queries  $Q$ , which is called *union of conjunctive queries*.

Now, assume that we want to answer the query  $q_2(x) = \{\text{isCreator}(x, y), \text{Film}(y)\}$ . By applying the same idea, we see that now  $\tau_2$  and  $\tau_3$  are relevant to the query and we can use them to extend the initial query, to produce the queries  $q_{2a}(x) = \{\text{isDirector}(x, y), \text{Film}(y)\}$ ,  $q_{2b}(x) = \{\text{isCreator}(x, y), \text{Movie}(y)\}$  and  $q_{2c}(x) = \{\text{isDirector}(x, y), \text{Movie}(y)\}$ . Similarly to the first query, we constructed the new queries by ‘replacing’ the concepts and roles of the original query with their subconcepts and roles; in this case, however, we have to take also their combinations.

At this point, we note that  $\tau_4$ , that tells us that a MovieDirector is a director of movies, is now also relevant to the query, and we can use it in conjunction with the query  $q_{2c}$  to obtain the additional query  $q_{2d}(x) = \{\text{MovieDirector}(x)\}$ . In this case we did not just ‘replace’ a concept or role of the original query with

a subconcept or subrole, but we had to combine a concept and a role into a new concept, using a TBox axiom. Thus, in this case the answers to  $q_2$  can be obtained by answering the union of conjunctive queries  $\{q_2, q_{2a}, q_{2b}, q_{2c}, q_{2d}\}$ .

Building on the idea illustrated in the above example, the query rewriting approach to query answering over semantic databases is to transform the query  $q$  using the TBox  $\mathcal{T}$  into a set of sentences  $\mathcal{R}$ , which is called *rewriting*, such that for any Abox  $\mathcal{A}$  the answers to  $q$  w.r.t.  $\mathcal{A}$  and  $\mathcal{T}$  coincide with the answers to  $q$  w.r.t.  $\mathcal{A}$  and  $\mathcal{R}$  discarding  $\mathcal{T}$  [25, 27]. More formally [30]:

**Definition 10.** *Let  $q$  be a conjunctive query, and  $\mathcal{T}$  a TBox. A rewriting  $\mathcal{R}$  of  $q$  w.r.t.  $\mathcal{T}$  is a datalog program whose rules can be partitioned into two disjoint sets  $\mathcal{R}_D$  and  $\mathcal{R}_q$ , such that  $\mathcal{R}_D$  does not mention  $q$ ,  $\mathcal{R}_q$  is a union of conjunctive queries with query predicate  $q$ , and where for each  $\mathcal{A}$  consistent w.r.t.  $\mathcal{T}$  and using only predicates from  $\mathcal{T}$  we have:*

$$\text{cert}(q, \mathcal{T} \cup \mathcal{A}) = \text{cert}(\mathcal{R}_q, \mathcal{R}_D \cup \mathcal{A}).$$

In Example 6, the rewriting of  $q$  was a union of conjunctive queries, and  $\mathcal{R}_D = \emptyset$ . In this case the rewriting can be answered over a semantic database  $\langle \mathcal{L}, \mathcal{B}, \mathcal{M} \rangle$  and a TBox  $\mathcal{T}$ , for a relational database  $\mathcal{B}$ , directly by using the mappings in  $\mathcal{M}$  to transform the union of conjunctive queries into an SQL query that can be executed directly over  $\mathcal{B}$ . As the above definition states, however, in the general case the rewriting  $\mathcal{R}$  is a datalog program. In this case, a scalable deductive database system capable of executing the datalog part of the rewriting is needed on top of database  $\mathcal{B}$ .

Most of the current query rewriting systems use resolution-based calculi to compute rewritings. In this approach, the TBox axioms are first transformed into a set of Horn clauses, which, together with the query, are then saturated using resolution to derive new clauses. (A *Horn clause* is a clause that has at most one positive literal, and hence can be written as a logic rule with one head atom).

*Example 7.* The axioms in the TBox of Example 6 can be transformed into the following first-order clauses:

$$\begin{aligned} \pi_1. \quad & \text{Creator}(x) \leftarrow \text{Director}(x), \\ \pi_2. \quad & \text{Film}(x) \leftarrow \text{Movie}(x), \\ \pi_3. \quad & \text{isCreator}(x, y) \leftarrow \text{isDirector}(x, y), \\ \pi_{4a}. \quad & \text{isDirector}(x, f(x)) \leftarrow \text{MovieDirector}(x), \\ \pi_{4b}. \quad & \text{Movie}(f(x)) \leftarrow \text{MovieDirector}(x). \end{aligned}$$

The conjunctive query  $q_2(x)$  can be rewritten as the clause

$$q_2. \quad q(x) \leftarrow \text{isCreator}(x, y) \wedge \text{Film}(y)$$

Resolving  $q_2$  with  $\pi_3$  and  $\pi_2$  we get queries  $q_{2a}$  and  $q_{2b}$ , respectively, and by resolving  $q_{2a}$  with  $\pi_2$  we get  $q_{2c}$ , which in clause form is

$$q_{2c}. \quad q(x) \leftarrow \text{isDirector}(x, y) \wedge \text{Movie}(y).$$

Proceeding and resolving  $q_{2c}$  with  $\pi_{4a}$  and  $\pi_{4b}$ , respectively, we get the clauses

$$\begin{aligned} q_{2c1}. q(x) &\leftarrow \text{MovieDirector}(x) \wedge \text{Movie}(f(x)) \\ q_{2c2}. q(x) &\leftarrow \text{isDirector}(x, f(x)) \wedge \text{MovieDirector}(x) \end{aligned}$$

Resolving either  $q_{2c1}$  with  $\pi_{4b}$  or  $q_{2c1}$  with  $\pi_{4a}$ , we get  $q(x) \leftarrow \text{MovieDirector}(x, y)$ , i.e. query  $q_{2d}$ .

The above example illustrates some of the key ideas in using resolution-based calculi for computing query rewritings. First, in the clausification step, i.e. in the conversion of the TBox to first-order clauses, the resulting clauses may either contain function terms or be function-free. Because of this, during the resolution process, intermediate clauses containing function terms may be derived, which should not included in the rewriting; the final rewriting consists only of function-free clauses. The number of such intermediate clauses that are produced and then discarded, but are potentially necessary to derive other clauses of the output rewriting, may be large and may even contain compositions of function terms. An intermediate non function-free clause may even not contribute at all to the derivation of new function-free clauses, as would be the case in the above example for clause  $q_{2c1}$  if  $\pi_{4b}$  was not part of  $\mathcal{T}$ .

Second, the size of the rewriting may be large, since it is necessary to perform all resolution steps in order to derive all clauses that contain all possible combinations of the subconcepts and subroles that can take the place of the atomic queries in the original query. In our particular example, with two atomic queries in the original query and two subconcept/subrole axioms we obtained four conjunctive queries in the rewriting. In general, the size of the resulting rewriting may be exponentially larger than the size of the initial query and the TBox.

Third, some of the intermediate clauses and output conjunctive queries may be computed several times, through different resolution chains. In the above example,  $q_{2d}$  has been obtained twice after resolving  $q_{2c1}$  with  $\pi_{4b}$  and  $q_{2c2}$  with  $\pi_{4a}$ . This means that during the resolution process many redundant recomputations may take place that do not contribute anything to the final rewriting. This negatively affects the efficiency of the rewriting process.

Finally, the exhaustive application of the resolution rule may create long derivations of clauses that are eventually subsumed by other clauses, and hence do not need to be included in the output rewriting. E.g. if in the above example the initial query was  $q(x) \leftarrow \text{Film}(x) \wedge \text{Movie}(x)$ , the resolution process would produce  $q(x) \leftarrow \text{Movie}(x)$ , which subsumes the initial query; a compact rewriting should include the latter, but not the initial query.

The existence of a rewriting and whether it is possible to compute a rewriting that is a pure union of conjunctive queries part depends on the expressivity of the TBox. Computing rewritings has been studied for various ontology languages, in particular for ontologies expressed in  $\mathcal{ELHI}$ , Horn- $\mathcal{SHIQ}$  and in the DL-Lite family of languages. The DL-Lite family of languages introduced in [25] is essentially the maximal language fragment exhibiting the desirable computational properties that allows the construction of rewritings that can be expressed as

unions of conjunctive queries, and hence the direct delegation of query answering to a relational database engine. In general, for the  $\mathcal{ELHI}$  and Horn- $SHIQ$  languages, rewritings containing a datalog part are produced, and hence a deductive database system is needed on top of the underlying relational database.

The first algorithm for computing rewritings for the DL-Lite family was proposed in [25] and implemented in the QuOnto system, which later evolved to Presto [26] and Mastro [5]. The algorithm encodes the TBox axioms as a set of custom rewriting rules, that are applied backwards on the query, and systematically replace concepts and roles in the query by concepts and roles that imply them. QuOnto does not produce a compact rewriting; the size of the produced rewriting can be very large and include many clauses subsumed by other clauses, which do not contribute anything to the query answers. Presto avoids this problem and generates a non-recursive datalog program instead of a union of conjunctive queries as a rewriting for DL-Lite ontologies. This essentially hides the exponential size of the rewriting inside the datalog rules. Requiem [27] proposed a resolution based approach, which rewrites the initial conjunctive query to a union of conjunctive queries which is, generally, smaller in size than the rewriting produced by QuOnto because systematic subsumption checking is applied to remove redundant clauses and produce a compact rewriting. Requiem supports also  $\mathcal{ELHI}$  ontologies, for which it produces datalog rewritings. Rapid [29,30], which we will discuss in detail later, carefully applies an optimized resolution-based rewriting technique only in cases that lead to the generation of useful, non-redundant conjunctive queries, avoiding in this way many query subsumption tests. Of course, although the optimization techniques reduce the size of the rewriting in many practical cases by avoiding redundancies, the size of the rewriting remains worst case exponential in the size of the original query and the Tbox. Rapid supports also  $\mathcal{ELHI}$  TBoxes, for which it produces datalog rewritings. Several other practical rewriting system have been developed for DL-Lite, including Quest [28], Nyaya [35], IQAROS [36], and Ontop [37]. Query rewriting techniques have also been developed for the more expressive Horn- $SHIQ$  language in the Clipper system [31].

### 4.3 The Rapid Query Rewriting System

Rapid is an optimized resolution-based query rewriting system, which tries to avoid or to minimize the effects of the problems inherent in the general resolution process outlined above. The general idea on which it is based is to constrain the resolution process in such a way so as to avoid the production of clauses that will later be discarded and not included in the final rewriting, either because they are non-function-free or because they are subsumed by others. Avoiding the production of such clauses early, helps to avoid redundant resolution steps later in the process. Originally it was developed to support DL-Lite<sub>R</sub> ontologies [29], subsequently it has been extended to support  $\mathcal{ELHI}$  ontologies [30], and currently it is being extended for Horn- $SHIQ$  ontologies.

**Rapid for DL-Lite<sub>R</sub>.** We will start the exposition of Rapid, discussing the query rewriting algorithm for DL-Lite<sub>R</sub>, a DL-Lite family language. As in any resolution-based query rewriting algorithm, the first part is the conversion of the TBox axioms to first-order clauses. Table 6 shows the axioms that are allowed in DL-Lite<sub>R</sub> and their corresponding first-order clause form, which we call DL-Lite<sub>R</sub> clauses. Note that for each occurrence of a concept of the form  $\exists R.B$  in the righthand side of a concept subsumption axiom, a distinct function symbol is used in the respective clauses. Thus, the same function symbol can occur at most twice in the clasified TBox.

**Table 6.** DL-Lite<sub>R</sub> axioms and their translation to clauses

Axiom	Clause
$B \sqsubseteq A$	$A(x) \leftarrow B(x)$
$\exists R \sqsubseteq A$	$A(x) \leftarrow R(x, y)$
$\exists R^- \sqsubseteq A$	$A(x) \leftarrow R(y, x)$
$A \sqsubseteq \exists R.B$	$R(x, f(x)) \leftarrow A(x)$
	$B(f(x)) \leftarrow A(x)$
$A \sqsubseteq \exists R^-.B$	$R(f(x), x) \leftarrow A(x)$
	$B(f(x)) \leftarrow A(x)$
$P \sqsubseteq R$	$R(x, y) \leftarrow P(x, y)$
$P \sqsubseteq R^-$	$R(x, y) \leftarrow P(y, x)$

The core part of Rapid implements a controlled resolution-based derivation process over the clasified TBox and a given conjunctive query. The crucial difference between Rapid and other resolution-based systems, such as Requiem, is that Rapid implements the resolution step by using as main premise always the initial conjunctive query or a query derived from it, and as side premise, or premises, clauses of the original clasified TBox  $\mathcal{T}$ , and not clauses that are resolvents of clauses of the clasified  $\mathcal{T}$  (i.e. of the saturation of the clasified  $\mathcal{T}$ ). The obvious benefit is that the clasified TBox is much smaller than its saturation. Moreover, Rapid performs in a controlled way resolution with the clauses of the clasified TBox that contain function symbols, taking advantage of the fact that the clasified TBox contains at most two clauses with the same function symbol. Finally, Rapid never produces, as resolvents, clauses than contain function symbols.

Formally, Rapid for DL-Lite<sub>R</sub> employs the  $\mathcal{I}_{lite}$  resolution-based inference system, given in the following definition (all definitions in this section are adapted from [30]):

**Definition 11.** *Let  $q$  be a conjunctive query.  $\mathcal{I}_{lite}$  is the inference system that consists of the following inference rules:*

– *Unfolding*:

$$\frac{q \quad \mathcal{C}}{q'\sigma} \quad \text{where}$$

1. the side premise  $\mathcal{C}$  is a  $DL\text{-}Lite_R$  clause,
2.  $q'\sigma$  is function-free resolvent of  $q$  and  $\mathcal{C}$ , and
3. if  $x \mapsto f(y) \in \sigma$ , then  $x \notin \text{ejvar}(q)$ .

– *Shrinking*:

$$\frac{q \quad \mathcal{C}_1 \quad [\mathcal{C}_2]}{q'\sigma} \quad \text{where}$$

1. the side premises  $\mathcal{C}_1$  and the optional  $\mathcal{C}_2$  are  $DL\text{-}Lite_R$  clauses,
2.  $q'\sigma$  is a function-free resolvent of  $q$ ,  $\mathcal{C}_1$  and  $\mathcal{C}_2$ , and
3. there exists some  $x \mapsto f(y) \in \sigma$  such that  $x \in \text{ejvar}(q)$ .

The rewriting of a conjunctive query  $q$  w.r.t to a  $DL\text{-}Lite_R$  TBox  $\mathcal{T}$  is the set of all the (function-free) clauses derivable from  $q$  and the clausified  $\mathcal{T}$  by  $\mathcal{I}_{lite}$ .

The unfolding rule represents standard resolution inferences with a function-free resolvent. Since this rule essentially ‘replaces’ an atomic query of  $q$  with a subconcept or a subrole, the resolvent has the same size with the main premise of the query (unless the replacement already exists in the main premise). Hence this rule ‘unfolds’ the main premise.

The shrinking rule represents a controlled resolution process involving clauses with function symbols that eventually leads to a function-free resolvent. In particular, it represents a chain of inferences  $q, q_1, \dots, q_n, q'$ , where  $q$  is a function-free conjunctive query,  $q_1$  contains a function symbol, and the subsequent inferences eliminate all occurrences of the function symbol until the function-free conjunctive query  $q'$  is obtained. Because a function symbol  $f$  can occur in at most two  $DL\text{-}Lite_R$  clauses, these inferences can involve at most the two different side premises that mention  $f$  (which according to Table 6 can only be of the form  $R(x, f(x)) \leftarrow B(x)$  and  $A(f(x)) \leftarrow B(x)$ ). Furthermore, since the resolvent is function-free, the variable of the query  $q$  that has been bound to the function symbol is eventually eliminated, and since  $DL\text{-}Lite_R$  clauses do not introduce new variables in their bodies, the resolvent has fewer variables than the main premise. Hence this rule ‘shrinks’ the main premise.

$\mathcal{I}_{lite}$  produces a union of conjunctive queries since all queries in the rewriting are clauses having the query predicate as head. It can be proved that  $\mathcal{I}_{lite}$  is correct, in the sense that it terminates and that it computes indeed a rewriting for obtaining the certain answers of  $q$ .

*Example 8.* In Example 7, the resolution of  $q_2$  and  $\pi_3$  to derive  $q_{2a}$  is an application of the unfolding rule:

$$\frac{q(x) \leftarrow \text{isCreator}(x, y) \wedge \text{Film}(y) \quad \text{isCreator}(x', y') \leftarrow \text{isDirector}(x', y')}{q(x) \leftarrow \text{isDirector}(x, y) \wedge \text{Film}(y)}$$

with  $\sigma = \{x' \mapsto x, y' \mapsto y\}$ . Similarly we obtain  $q_{2b}$  from  $q_2$  and  $\pi_2$ . The resolution of  $q_{2a}$  and  $\pi_2$  to derive  $q_{2c}$  is also an application of the unfolding rule:

$$\frac{q(x) \leftarrow \text{isDirector}(x, y) \wedge \text{Film}(y) \quad \text{Film}(x') \leftarrow \text{Movie}(x')}{q(x) \leftarrow \text{isDirector}(x, y) \wedge \text{Movie}(y)}$$

with  $\sigma = \{x' \mapsto y\}$ .

Then, conjunctive query  $q_{2d}$  can be derived from  $q_{2c}$  using the shrinking rule and  $\pi_{4a}$  and  $\pi_{4b}$  as side premises:

$$\frac{q(x) \leftarrow \text{isDirector}(x, y) \wedge \text{Movie}(y) \quad \begin{array}{l} \text{isDirector}(x', f(x')) \leftarrow \text{MovieDirector}(x') \\ \text{Movie}(f(x')) \leftarrow \text{MovieDirector}(x') \end{array}}{q(x) \leftarrow \text{MovieDirector}(x)}$$

with  $\sigma = \{y \mapsto f(x), x' \mapsto x\}$ , and  $y$  is the variable that is eliminated from  $q_{2c}$ .

The application of the shrinking step on clause  $q_{2c}$  to derive directly  $q_{2d}$  corresponds to a double saving, since it does not only avoid the production of the intermediate clauses  $q_{2c1}$  and  $q_{2c2}$ , but also produces  $q_{2d}$  only once.

Although  $\mathcal{I}_{lite}$  avoids many eventually unneeded resolution steps, it does not always avoid the production of redundant queries, i.e. of queries that are subsumed by others and do not need to be included in the final rewriting. Checking queries for subsumption after all queries have been generated, in order to produce an as compact rewriting as possible, is very expensive and may lead to poor rewriting times. Redundant queries can be produced by the unfolding rule because different chains of applications of the unfolding rule on different atoms of the main premise may produce the same clause. Moreover, an unfolding may replace an atom of the main premise with an atom that already exists in the main premise, hence reducing the size of the query. In such a case the resolvent subsumes all queries of greater size that are supersets of the resolvent. Finally, queries produced using the shrinking rule are likely to subsume queries produced by the unfolding rule, since they ‘shrunk’ queries are shorter.

To efficiently address some of the above issues, the practical implementation of Rapid does not explicitly construct the queries that should normally be produced using the unfolding rule. Instead, it computes a structure that holds all the information that is needed to construct all queries derivable by the unfolding rule from the same starting conjunctive query. After this structure has been constructed, a redundant-free set of the queries unfoldings can automatically be generated.

**Definition 12.** Let  $\mathcal{T}$  be a DL-Lite TBox,  $q$  a conjunctive query, and  $A$  an atom of  $q$ . Let  $q_A$  be the query having  $A$  as body and  $\text{avar}(q_A) = \text{var}(A) \cap \text{ejvar}(q)$ . The unfolding set of  $A$  w.r.t.  $q$  and  $\mathcal{T}$  is the set that contains all atoms that are bodies of the queries derivable from the clausified  $\mathcal{T}$  and  $q_A$  using only the unfolding rule, including the atom  $A$ .

For a query  $q$ , the unfolding sets of its atoms fully represent the queries that can be derived from  $q$  by unfolding. Indeed, all such queries can be constructed

by taking all possible combinations of atoms from the respective unfolding sets. However, the unfolding sets allow the generation of the minimum number of queries that represent all unfoldings of  $q$ , i.e. the generation only of the unfoldings that are not subsumed by other unfoldings and hence are redundant. This can be done by scanning the respective unfolding sets and cross checking for the presence of identical atoms (up to variable renamings) in the unfolding sets corresponding to different atoms of the query body, while producing the set of unfoldings. This process essentially performs subsumption checking in a much more efficient way because it does not check for subsumption whole clauses, but carefully recasts the problem to comparing individual atoms.

Clearly, depending on the particular TBox and query, this step can still result in an exponential number of queries in the rewriting. As in other query rewriting systems, this exponential behaviour can be hidden by producing a datalog rewriting instead of a union of conjunctive queries rewriting. Rapid provides also this option; in this case the rules of the datalog program essentially encode the non explicitly generated unfoldings.

*Example 9.* For query  $q_2$  of Example 7, Rapid calculates unfolding sets for atoms  $\text{isCreator}(x, y)$  and  $\text{Film}(y)$ , which are the sets  $\{\text{isCreator}(x, y), \text{isDirector}(x, y)\}$  and  $\{\text{Film}(y), \text{Movie}(y)\}$ , respectively. These are then used to actually generate the queries  $q_2$ ,  $q_{2a}$ ,  $q_{2b}$  and  $q_{2c}$ . Recall from Example 6 that the rewriting of  $q_2$  as a union of conjunctive queries is  $\{q_2, q_{2a}, q_{2b}, q_{2c}, q_{2d}\}$ . If we choose not to explicitly perform the unfoldings and produce a datalog program instead, the rewriting would be  $\{q_2, \pi_2, \pi_3, q_{2d}\}$ .

**Rapid for  $\mathcal{ELHI}$ .** Moving from the DL-Lite family of languages to the  $\mathcal{ELHI}$  language, we loose the property that the rewriting can always be a union of conjunctive queries. Table 7 shows the permissible axioms in  $\mathcal{ELHI}$  and their clausifications, which we call  $\mathcal{ELHI}$  clauses.

**Table 7.**  $\mathcal{ELHI}$  axioms and their translation to clauses

Axiom	Clause
$B \sqsubseteq A$	$A(x) \leftarrow B(x)$
$B \sqcap C \sqsubseteq A$	$A(x) \leftarrow B(x) \wedge C(x)$
$\exists R \sqsubseteq A$	$A(x) \leftarrow R(x, y)$
$\exists R^- \sqsubseteq A$	$A(x) \leftarrow R(y, x)$
$A \sqsubseteq \exists R.B$	$R(x, f(x)) \leftarrow A(x)$
	$B(f(x)) \leftarrow A(x)$
$A \sqsubseteq \exists R^-.B$	$R(f(x), x) \leftarrow A(x)$
	$B(f(x)) \leftarrow A(x)$
$\exists R.C \sqsubseteq A$	$A(x) \leftarrow R(x, y) \wedge C(y)$
$\exists R^-.C \sqsubseteq A$	$A(x) \leftarrow R(y, x) \wedge C(y)$
$P \sqsubseteq R$	$R(x, y) \leftarrow P(x, y)$
$P \sqsubseteq R^-$	$R(x, y) \leftarrow P(y, x)$



The essential difference with respect to  $\text{DL-Lite}_R$  is the permission of axioms of the form  $\exists R.C \sqsubseteq A$ , whose clausification takes the form  $A(x) \leftarrow R(x, y) \wedge C(y)$ . The distinguishing property of such clauses, which are called  $RA$ -clauses, is that when used as side premises of an inference they can produce resolvents containing more variables than the main premise. If there is a cyclicity in the axioms, this can lead to termination problems. E.g. rewriting the query  $q(x) \leftarrow A(x)$  using the clause  $A(x) \leftarrow R(x, y) \wedge A(y)$ , produces the queries  $q(x) \leftarrow R(x, y) \wedge A(y)$ ,  $q(x) \leftarrow R(x, y) \wedge A(y) \wedge R(y, z) \wedge A(z)$ , etc. Hence, to guarantee termination, the calculus for  $\mathcal{ELHI}$  should not allow  $RA$ -clauses as side premises, but still produce the clauses that are derivable by  $RA$ -clauses. It turns out that to achieve this, the restriction not to perform resolution using clauses of the TBox as main premises must be lifted.

Consider e.g. the clausified TBox  $c_1. C(x) \leftarrow S(x, y) \wedge D(y)$ ,  $c_2. S(f(x), x) \leftarrow B(x)$  and  $c_3. K(x) \leftarrow S(y, x) \wedge C(y)$ , and the query  $q_1. q(x) \leftarrow K(x)$  (the example is adapted from [30]). By resolving  $c_1$  with  $c_2$  we get  $c_4. C(f(x)) \leftarrow B(x) \wedge D(x)$ , by resolving  $c_3$  with  $c_2$  we get  $c_5. K(x) \leftarrow B(x) \wedge C(f(x))$ , by resolving  $c_5$  with  $c_4$  we get  $c_6. K(x) \leftarrow B(x) \wedge D(x)$ , and finally by resolving  $q_1$  with  $c_6$  we get  $q_{1a}. q(x) \leftarrow B(x) \wedge D(x)$ . The crucial clause here is  $c_4$  which is needed to derive the rewriting, but can be produced only by performing resolution on clauses of the TBox. To account for this, the Rapid calculus for  $\mathcal{ELHI}$  ontologies includes a new inference rule, called function rule, which can produce clauses like  $c_4$  from  $RA$ -clauses and clauses of the form of  $c_2$ . It also extends the unfolding and shrinking rules so as to allow also  $RA$ -clauses as main premises, in order to be able to compute, e.g. clause  $c_6$  from  $c_2$ ,  $c_3$ , and  $c_4$ . Because the function rule can produce clauses with function symbols in the head, there can now be more than two clauses mentioning the same function symbol  $f$ . Hence, the extended shrinking rule allows for arbitrary number of side premises.

**Definition 13.** Let  $\Upsilon$  be either a conjunctive query or an  $RA$ -clause. With  $\mathcal{I}_{\mathcal{EL}}$  we denote the inference system consisting of the following rules:

– *Unfolding:*

$$\frac{\Upsilon \quad C}{\Upsilon'\sigma} \text{ where}$$

1. the side premise  $C$  is an  $\mathcal{ELHI}$ , non  $RA$ -clause,
2.  $\Upsilon'\sigma$  is a function-free resolvent of  $\Upsilon$  and  $C$ , and
3. if  $x \mapsto f(y) \in \sigma$  then  $x \notin \text{ejvar}(\Upsilon)$ .

– *n-Shrinking:*

$$\frac{\Upsilon \quad C_1 \ [C_2 \ \dots \ C_n]}{\Upsilon'\sigma} \text{ where}$$

1. the side premises  $C_1, \dots, C_n$ ,  $n \geq 1$  are  $\mathcal{ELHI}$ , non  $RA$ -clauses,
2.  $\Upsilon'\sigma$  is a function-free resolvent of  $\Upsilon$  and all  $C_1, \dots, C_n$  for  $n \geq 1$ , and
3. some  $x \mapsto f(y) \in \sigma$  exists such that  $x \in \text{ejvar}(\Upsilon)$ .

– *Function:*

$$\frac{B(x) \leftarrow R(x, y) \wedge [C(y)] \quad R(f(x), x) \leftarrow A(x)}{B(f(x)) \leftarrow A(x) \wedge [C(x)]} \quad \text{or}$$

$$\frac{B(x) \leftarrow R(y, x) \wedge [C(y)] \quad R(x, f(x)) \leftarrow A(x)}{B(f(x)) \leftarrow A(x) \wedge [C(x)]}$$

where  $[C(y)]$  denotes an optional conjunction of atoms, all with argument  $y$ .

The rewriting of a conjunctive query  $q$  w.r.t a  $\mathcal{ELHI}$  TBox  $\mathcal{T}$  is defined as the set of all function-free clauses derivable from  $q$  and the clausified  $\mathcal{T}$  by  $\mathcal{I}_{\mathcal{EL}}$ .

The function rule is not expected to ‘fire’ often in practice since it models a rather complex interaction between a clause containing  $R(x, f(x))$  or  $R(f(x), x)$  and an  $RA$ -clause containing the inverse  $R(y, x)$ , or  $R(x, y)$ , respectively. In practice, the application of the  $\mathcal{I}_{\mathcal{EL}}$  calculus on a conjunctive query  $q$  and a TBox  $\mathcal{T}$  can be performed in two steps. The first saturates  $\mathcal{T}$  using  $\mathcal{I}_{\mathcal{EL}}$  and only the  $RA$ -clauses as main premises. Then, the second step collects all non  $RA$ -clauses from the clausified  $\mathcal{T}$  and those produced in the previous step and uses them as side premises in the unfolding and shrinking rules with main premises only query clauses.

*Example 10.* Consider the TBox  $\mathcal{T} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$  where

$$\begin{aligned} \tau_1. & \text{ Director} \sqsubseteq \text{Creator}, \\ \tau'_2. & \text{ Movie} \equiv \text{Film}, \\ \tau_3. & \text{ isDirector} \sqsubseteq \text{isCreator}, \\ \tau'_4. & \text{ MovieDirector} \equiv \exists \text{ isDirector.Movie}. \end{aligned}$$

The difference w.r.t. the TBox of Example 6 is that  $\tau_2$  and  $\tau_4$  have been replaced by  $\tau'_2$  and  $\tau'_4$ , respectively, where the subconcept relations have been replaced by equivalence relations. These axioms can be transformed into the following  $\mathcal{ELHI}$  clauses:

$$\begin{aligned} \pi_1. & \text{ Creator}(x) \leftarrow \text{Director}(x), \\ \pi'_{2a}. & \text{ Film}(x) \leftarrow \text{Movie}(x), \\ \pi'_{2b}. & \text{ Movie}(x) \leftarrow \text{Film}(x), \\ \pi_3. & \text{ isCreator}(x, y) \leftarrow \text{isDirector}(x, y), \\ \pi'_{4a}. & \text{ isDirector}(x, f(x)) \leftarrow \text{MovieDirector}(x), \\ \pi'_{4b}. & \text{ Movie}(f(x)) \leftarrow \text{MovieDirector}(x) \\ \pi'_{4c}. & \text{ MovieDirector}(x) \leftarrow \text{isDirector}(x, y) \wedge \text{Movie}(y). \end{aligned}$$

Consider that we want to answer the query  $q_3 = \{\text{MovieDirector}(x)\}$ , which in clause form is  $q_3(x) \leftarrow \text{MovieDirector}(x)$ . To apply  $\mathcal{I}_{\mathcal{EL}}$ , we need to saturate first the clausified TBox using only  $RA$ -clauses as main premises. The only  $RA$ -clause is  $\pi'_{4c}$ , for which we can apply the unfolding rule with  $\pi'_{2b}$  as side premise:

$$\frac{\text{MovieDirector}(x) \leftarrow \text{isDirector}(x, y) \wedge \text{Movie}(y) \quad \text{Movie}(x') \leftarrow \text{Film}(x')}{\text{MovieDirector}(x) \leftarrow \text{isDirector}(x, y) \wedge \text{Film}(y)}$$

with  $\sigma = \{x' \mapsto y\}$ . Let  $\pi_5$  be the resulting clause. The shrinking rule is also applicable on  $\pi'_{4c}$  with  $\pi'_{4a}$  and  $\pi'_{4b}$ , but it produces a tautology which is immediately discarded. No other resolution is possible using RA-clauses as main premise, so we proceed by applying the rules using only query clauses as main premise. Because no rule is applicable on  $q_3(x) \leftarrow \text{MovieDirector}(x)$  with a non RA-clause as side premise, the final rewriting is  $\{q_3, \pi'_{4c}, \pi_5\}$ , where the datalog part is  $\{\pi'_{4c}, \pi_5\}$ . Note that if we did not perform explicitly the unfolding, we could produce the equivalent rewriting  $\{q_3, \pi'_{4c}, \pi'_{2a}\}$ . Note also that, since no structural circularity exists in the TBox axioms, the rewriting could also be expanded into a union of conjunctive queries. In this case, the rewriting would be  $\{q_3, q_{3a}, q_{3b}\}$ , where  $q_{3a}(x) = \{\text{isDirector}(x, y), \text{Movie}(y)\}$  and  $q_{3b}(x) = \{\text{isDirector}(x, y), \text{Film}(y)\}$ .

## 5 Semantic Data Representation in the Web

This section provides a brief overview of the current technologies that allow the use of the techniques discussed in the previous sections in real applications, in particular the construction of functional knowledge bases (i.e. of ABoxes and TBoxes) that possibly use data from an underlying relational database and support query answering. These technologies have been developed as part of the Semantic Web and are now standards of the W3C (World Wide Web Consortium).

### 5.1 RDF

In Semantic Web applications ABoxes are represented using RDF [50], which is a general framework for making statements about resources. An RDF statement always has the structure  $\langle \text{subject} \rangle \langle \text{predicate} \rangle \langle \text{object} \rangle$ , where the subject and the object represent the two resources being related and the predicate represents the type of the relationship. RDF statements are called (*RDF*) *triples*.

To represent an ABox as a set of RDF statements we need statements for expressing concept and role assertions of the form  $C(a)$  and  $r(a, b)$ , respectively, where  $C$  is a concept,  $r$  a role, and  $a, b$  individuals. In RDF each concept, role or individual is a resource having a IRI. Thus, assuming some namespace  $\text{ns}$ , a concept assertion  $C(a)$  is represented by the triple

$$\text{ns:C rdf:type ns:a}$$

where  $\text{rdf:type}$  is a special property defined by the RDF standard (which can also be shorthand as  $\text{a}$ ), and the role assertion  $r(a, b)$  is represented by the triple

$$\text{ns:a ns:r ns:b}$$

An RDF dataset, i.e. a set of RDF statements, is an *RDF graph*. RDF graphs are stored in triple stores, which are databases specifically build for storing and retrieving RDF triples.

For writing down RDF graphs there exist several serializations, such as N-Triples, Turtle, and RDF/XML. In our continuing example, assuming a namespace *cine* corresponding e.g. to `<http://image.ece.ntua.gr/cinemaOntology/>`, the triples stating that Woody Allen is a director and has directed the movie Manhattan can be written in Turtle syntax as following:

```
@prefix cine: <http://image.ece.ntua.gr/cinemaOntology/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

cine:woodyAllen
  cine:isDirector cine:manhattan;
  rdf:type cine:Director.
```

## 5.2 OWL 2

In Semantic Web applications, concepts, roles, individuals and axioms between them can be modeled using the OWL 2 language [56]. OWL 2 provides structures for expressing all constructors and axioms of Tables 4 and 5. Because OWL 2 is a very expressive ontology language (c.f. Sect. 3) and the use of its full expressivity in real applications poses computational problems, OWL 2 provides three profiles, which reflect compromises between expressivity and desirable computational properties. These profiles are OWL 2 QL, OWL 2 EL, and OWL 2 RL. OWL 2 QL is based on the DL-Lite family of languages [25] and hence can be used for answering conjunctive queries by translating them to SQL queries using query rewriting, as discussed in Sect. 4.2 and avoiding materialization. OWL 2 EL is based on the  $\mathcal{ELHI}$  language and OWL 2 RL on DLP [17]. As discussed in Sect. 4.1, OWL 2 RL consequences can be modeled using rule-based techniques.

OWL 2 axioms can be written down as RDF triples and stored in a triple store, along with other RDF triples corresponding to ABox assertions. In this way, the triple store can represent a knowledge base of the form  $\langle \mathcal{T}, \mathcal{A} \rangle$ . However, in order for to guarantee that a set of OWL 2 axioms constitutes a TBox under the model theoretic-semantics of Sect. 3, some additional syntactic conditions must be imposed on the OWL 2 structures. These conditions are specified by the OWL 2 Direct Semantics, and guarantee that the knowledge base is compatible with the  $\mathcal{SROIQ}$  language. OWL 2 ontologies that satisfy these syntactic conditions are called OWL 2 DL ontologies.

There are several serializations of OWL 2, the more reader friendly of which is the functional syntax. In this syntax, the TBox of Example 6 is written as follows:

```
Prefix(cine:=<http://image.ece.ntua.gr/cinemaOntology/>)
Ontology(<http://image.ece.ntua.gr/cinemaOntology/>
  SubClassOf(cine:Director cine:Creator)
  SubClassOf(cine:Movie cine:Film)
  SubObjectPropertyOf(cine:isDirector cine:isCreator)
  SubClassOf(cine:MovieDirector
    ObjectSomeValuesFrom(cine:isDirector cine:Movie) )
)
```

### 5.3 R2RML

R2RML [57] is a language for expressing mappings from relational databases to RDF datasets. Every R2RML mapping is constructed for a specific database schema and target vocabulary. The input to an R2RML mapping is a relational database that conforms to the database schema. The output is an RDF dataset, that uses roles and concepts from the target vocabulary. R2RML mappings are expressed as RDF graphs and written down in Turtle syntax.

R2RML allows us to model the semantic mappings defined in Sect. 4:  $\mathcal{L}$  corresponds to the target vocabulary,  $\mathcal{B}$  is the input relational database, and the semantic mapping box  $\mathcal{M}$  corresponds to the actual mappings defined in an R2RML document. Note, however, that unlike Definition 4, R2RML mappings cannot contain object identifier, and concept or role classifier functions, other than the functions allowed by the SQL language supported by the underlying database.

The mappings defined in an R2RML document is conceptual; An R2RML implementation may either materialize the mappings, or access directly the underlying database when answering queries. These two strategies correspond to the options of explicitly converting the semantic database into an RDF dataset representing the ABox, or of converting the queries to SQL queries without explicitly constructing the ABox, that we have discussed in Sect. 4.

An R2RML mapping is defined as a set of mappings from logical tables to sets of RDF triples. A logical table may be a database table, a view, or a valid SQL query. Each individual mapping, which is called a triples map, is a rule consisting of two main parts: (a) a subject map that generates the subject of the RDF triples that will be generated from each logical table row, and (b) one or more predicate-object maps that in turn consist of predicate and object maps, which specify the predicates and the objects of the RDF triples that will be generated for the respective subject. In the context of ABoxes, subjects, predicates and objects should be IRIs, and the subject, predicate and object maps should provide instructions on how to generate them. An example of a mapping is the following:

1. Use the template `http://image.ece.ntua.gr/cinemaOntology/{DirID}` to generate the subject IRI from the `DirID` column of the `DIRECTOR-OF` table.
2. Use the constant IRI `cine:isDirector` as predicate.
3. Use the template `http://image.ece.ntua.gr/cinemaOntology/{MovID}` to generate the object IRI from the `MovID` column.

Expressed in R2RML the above mapping is the following:

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix cine: <http://image.ece.ntua.gr/cinemaOntology/>.

<#TriplesMap1>
  rr:logicalTable [ rr:tableName "DIRECTOR-OF" ];
  rr:subjectMap [
```

```

rr:template "http://image.ece.ntua.gr/cinemaOntology/{DirID}";
rr:class cine:Director;
];
rr:predicateObjectMap [
  rr:predicate cine:isDirector;
  rr:objectMap [
    rr:template "http://image.ece.ntua.gr/cinemaOntology/{MovID}";
  ];
].

```

The above R2RML document implements the mapping described above, and in addition specifies that for each subject IRI a triple determining the subject as being of type `cine:Director` should also be created. Another example is the following R2RML mapping, which generates ABox assertions for the concept `DirectorOfComedy` using an SQL query as logical table.

```

<#TriplesMap2>
rr:logicalTable [rr:sqlQuery "
  SELECT DIRECTOR-OF.DirID,
  FROM MOVIES, DIRECTOR-OF
  WHERE MOVIES.MovID = DIRECTOR-OF.MovID AND
  MOVIES.Genre = 'Comedy'";
rr:subjectMap [
  rr:template "http://image.ece.ntua.gr/cinemaOntology/{DirID}";
  rr:class cine:DirectorOfComedy;
].

```

## 5.4 SPARQL

The set of RDF triples that are stored in a triple store can be queried using SPARQL [53]. A typical SPARQL query consists of two parts: a **SELECT** clause that identifies the query answer variables, and a **WHERE** clause that provides the basic graph pattern to be matched against the underlying RDF graph. The basic graph pattern is a set of triple patterns. Triple patterns are like RDF triples but the subject, the predicate or the object may be variables. Variables are names preceded by a question mark. A simple SPARQL query is the following:

```

PREFIX cine: <http://image.ece.ntua.gr/cinemaOntology/>
SELECT ?s ?o
WHERE { ?s cine:isDirector ?o }

```

which will return the subject and the object of all triples having `isDirector` as predicate. It corresponds to the conjunctive query  $q(x, y) = \{isDirector(x, y)\}$ . This query example consists of a single triple pattern with the variable `?s` in the subject position and the variable `?o` in the object position.

To express conjunctive queries, a basic graph pattern consisting of more than one triple patterns can be used. E.g. the query

```

PREFIX cine: <http://image.ece.ntua.gr/cinemaOntology/>
PREFIX <rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?s
WHERE { ?s cine:isDirector ?o .
        ?s rdf:type cine:Actor }

```

will return all directors of some film that are also actors and corresponds to the conjunctive query  $q(x) = \{\text{isDirector}(x, y), \text{Actor}(x)\}$ .

SPARQL allows also disjunctive queries through the UNION keyword. E.g. the query

```

PREFIX cine: <http://image.ece.ntua.gr/cinemaOntology/>
SELECT ?s
WHERE {{ ?s cine:isDirector ?o } UNION { ?s cine:isProducer ?o } }

```

will return everyone that has directed or produced a movie.

SPARQL includes many more facilities, such as negation, property paths, assignment, aggregation, multiple graphs, federated querying, filtering, sorting and limiting answers. Several of these facilities are used with more general triple stores than ABoxes and more general queries than conjunctive queries we are interested in this paper.

By default, a triple store answers a SPARQL query by matching the query pattern with the RDF graph of the data it holds. If the triple store is a pure ABox  $\mathcal{A}$ , this is all that is needed. As we have discussed in detail in Sect. 4, however, if the triple store is a knowledge base  $\langle \mathcal{T}, \mathcal{A} \rangle$ , some answers to the query may not be explicitly present in  $\mathcal{A}$ , but need to be inferred using the axioms in  $\mathcal{T}$ . To allow for such inference processes to be performed by triples stores, SPARQL provides the so-called *entailment regimes* [54]. For OWL 2 TBoxes (OWL 2 DL ontologies), the OWL 2 Direct Semantics Entailment Regime is provided.

Due to the computational difficulties in reasoning under the full expressivity of OWL 2, several practical implementation of triple stores usually provide some limited support of the full OWL 2 entailment, limiting themselves to supporting an OWL 2 profile, usually the OWL 2 RL profile, in which, as we have seen in Sect. 4.1, inferences can be computed using materialization rule-based techniques.

## 6 Conclusions

The paper discusses how ontological descriptions can be used as a basis for semantic data access. Specifically, we saw how we can build efficient user interfaces that provide the user with the ability to express queries in terms of a rich vocabulary relevant to the domain of interest, rather than queries employing the technical terminology of the database schemas.

We started our paper (Sect. 2) by describing technologies used to store semantic data, either by directly using the vocabulary, or by ‘semantifying’ the information stored in the database, using the vocabulary terminology. Then, in Sect. 3

we saw how the use of ontological knowledge representation languages (description logics), that are supported by automated reasoning, can advance the level of semantic data description, enriching the vocabulary by adding new terms, or by expressing formal restrictions and constraints of the domain of interest. In Sect. 4 we saw that the use of ontology reasoning, specifically the use of ontology-based data access technologies, can support realistic scenarios of concept-based data access systems for pragmatic applications, with a lot of advantages. Finally, in Sect. 5 we described technologies and standards for the representation and use of semantic data in the web.

## References

1. Abiteboul, S., Manolescu, I., Rousset, M.-C., Senellart, P.: *Web Data Management*. Cambridge University Press, New York (2011)
2. Abiteboul, S., Hull, R., Vianu, V.: *Foundations of Databases*. Addison-Wesley, Chicago (1995)
3. Garcia-Molina, H., Ullman, J.D., Widom, J.: *Database Systems - The Complete Book*, 2nd edn. Pearson Education, Harlow (2009)
4. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. *J. Data Semant.* **10**, 133–173 (2008)
5. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., Savo, D.F.: The MASTRO system for ontology-based data access. *Semant. Web* **2**(1), 43–53 (2011)
6. Jiménez-Ruiz, E., et al.: BootOX: practical mapping of RDBs to OWL 2. In: Arenas, M., et al. (eds.) *ISWC 2015*. LNCS, vol. 9367, pp. 113–132. Springer, Cham (2015). doi:[10.1007/978-3-319-25010-6\\_7](https://doi.org/10.1007/978-3-319-25010-6_7)
7. Rudolph, S.: Foundations of description logics. In: Polleres, A., d’Amato, C., Arenas, M., Handschuh, S., Kroner, P., Ossowski, S., Patel-Schneider, P. (eds.) *Reasoning Web 2011*. LNCS, vol. 6848, pp. 76–136. Springer, Heidelberg (2011). doi:[10.1007/978-3-642-23032-5\\_2](https://doi.org/10.1007/978-3-642-23032-5_2)
8. Krötzsch, M., Simančík, F., Horrocks, I.: A description logic primer, CoRR abs/1201.4089 (2012)
9. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: *The Description Logic Handbook: Theory, Implementation, and Applications*, 2nd edn. Cambridge University Press, Cambridge (2007)
10. van Harmelen, F., Lifschitz, V., Porter, B.: *Handbook of Knowledge Representation*. Foundations of Artificial Intelligence. Elsevier Science, New York (2008)
11. Hitzler, P., Krötzsch, M., Rudolph, S.: *Foundations of Semantic Web Technologies*. Chapman and Hall/CRC, Boca Raton (2009)
12. Staab, S., Studer, R.: *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, Heidelberg (2010)
13. Baader, F., Sattler, U.: An overview of tableau algorithms for description logics. *Studia Logica* **69**(1), 5–40 (2001)
14. Motik, B., Sattler, U.: A comparison of reasoning techniques for querying large description logic ABoxes. In: Hermann, M., Voronkov, A. (eds.) *LPAR 2006*. LNCS (LNAI), vol. 4246, pp. 227–241. Springer, Heidelberg (2006). doi:[10.1007/11916277\\_16](https://doi.org/10.1007/11916277_16)



15. De Giacomo, G., Lenzerini, M.: TBox and ABox reasoning in expressive description logics. In: *Proceedings of the 1996 International Workshop on Description Logics*, 2–4 November, 1996, Cambridge, MA, USA, pp. 37–48 (1996)
16. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible SROIQ. In: *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006)*, pp. 57–67. AAAI Press (2006)
17. Grosz, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: combining logic programs with description logic. In: *Proceedings of the 12th International Conference on World Wide Web (WWW 2003)*, pp. 48–57 (2003)
18. Glimm, B., Horrocks, I., Motik, B., Stoilos, G., Wang, Z.: HermiT: an OWL 2 reasoner. *J. Autom. Reasoning* **53**(3), 245–269 (2014)
19. Haarslev, V., Hidde, K., Möller, R., Wessel, M.: The RacerPro knowledge representation and reasoning system. *Semant. Web J.* **3**(3), 267–277 (2012)
20. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: a practical OWL-DL reasoner. *J. Web Semant.* **5**(2), 51–53 (2011)
21. Steigmiller, A., Liebig, T., Glimm, B.: Konclude: system description. *Web Semant. Sci. Serv. Agents World Wide Web* **27–28**, 78–85 (2014)
22. Volz, R., Staab, S., Motik, B.: Incrementally maintaining materializations of ontologies stored in logic databases. *J. Data Semant.* **2**, 1–34 (2005)
23. Motik, B., Nenov, Y., Piro, R., Horrocks, I., Olteanu, I.: Parallel OWL 2 RL materialisation in centralised, main-memory RDF systems. In: *Bienvenu, M., Ortiz, M., Rosati, R., Simkus, M. (eds.) Informal Proceedings of the 27th International Workshop on Description Logics, Vienna, Austria, 17–20 July, 2014*, vol. 1193 of *CEUR Workshop Proceedings*, pp. 311–323. CEUR-WS.org (2014)
24. Bishop, B., Kiryakov, A., Ognyanoff, D., Peikov, I., Tashev, Z., Velkov, R.: OWLIM: a family of scalable semantic repositories. *Semant. Web* **2**(1), 33–42 (2011)
25. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: the DL-Lite family. *J. Autom. Reasoning* **39**(3), 385–429 (2007)
26. Rosati, R., Almatelli, A.: Improving query answering over dl-lite ontologies. In: *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, 9–13 May, 2010* (2010)
27. Pérez-Urbina, H., Motik, B., Horrocks, I.: Tractable query answering and rewriting under description logic constraints. *J. Appl. Logic* **8**(2), 186–209 (2010)
28. Rodríguez-Muro, M., Calvanese, D.: High performance query answering over dl-lite ontologies. In: *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference (KR 2012), Rome, Italy, 10–14 June, 2012* (2012)
29. Chortaras, A., Trivela, D., Stamou, G.: Optimized query rewriting for OWL 2 QL. In: *Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS (LNAI)*, vol. 6803, pp. 192–206. Springer, Heidelberg (2011). doi:[10.1007/978-3-642-22438-6\\_16](https://doi.org/10.1007/978-3-642-22438-6_16)
30. Trivela, D., Stoilos, G., Chortaras, A., Stamou, G.: Optimising resolution-based rewriting algorithms for OWL ontologies. *J. Web Semant.* **33**, 30–49 (2015)
31. Eiter, T., Ortiz, M., Simkus, M., Tran, T., Xiao, G.: Query rewriting for Horn-SHIQ plus rules. In: *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 22–26 July, 2012, Toronto, Ontario, Canada (2012)

32. Lutz, C., Toman, D., Wolter, F.: Conjunctive query answering in the description logic EL using a relational database system. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009), Pasadena, California, USA, 11–17 July, 2009, pp. 2070–2075 (2009)
33. Hustadt, U., Motik, B., Sattler, U.: Reasoning in description logics by a reduction to disjunctive datalog. *J. Autom. Reasoning* **39**(3), 351–384 (2007)
34. Calvanese, D., Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R.: Ontologies and databases: the DL-Lite approach. In: Tessaris, S., Franconi, E., Eiter, T., Gutierrez, C., Handschuh, S., Rousset, M.-C., Schmidt, R.A. (eds.) Reasoning Web 2009. LNCS, vol. 5689, pp. 255–356. Springer, Heidelberg (2009). doi:[10.1007/978-3-642-03754-2\\_7](https://doi.org/10.1007/978-3-642-03754-2_7)
35. Orsi, G., Pieris, A.: Optimizing query answering under ontological constraints. *J. Very Large Database (VLDB) Endow.* **11**(4), 1004–1015 (2011)
36. Venetis, T., Stoilos, G., Stamou, G.: Query extensions and incremental query rewriting for OWL 2 QL ontologies. *J. Data Semant.* **3**(1), 1–23 (2014)
37. Rodriguez-Muro, M., Kontchakov, R., Zakharyashev, M.: Query rewriting and optimisation with database dependencies in ontop. In: Proceedings of the 26th International Workshop on Description Logics (DL 2013) (2013)
38. Trivela, D., Stoilos, G., Chortaras, A., Stamou, G.: Query rewriting in Horn-SHIQ. In: Proceedings of the 28th International Workshop on Description Logics, Athens, Greece, 7–10 June, 2015 (2015)
39. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyashev, M.: The combined approach to query answering in DL-Lite. In: Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference (KR 2010), Toronto, Ontario, Canada, 9–13 May, 2010 (2010)
40. Stefanoni, G., Motik, B., Horrocks, I.: Introducing nominals to the combined query answering approaches for EL. In: Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, 14–18 July, 2013, Bellevue, Washington, USA (2013)
41. Zhou, Y., Grau, B.C., Nenov, Y., Horrocks, I.: PAGOdA: pay-as-you-go ABox reasoning. In: Calvanese, D., Konev, B. (eds.), Proceedings of the 28th International Workshop on Description Logics, Athens, Greece, 7–10 June, 2015, vol. 1350 of CEUR Workshop Proceedings. CEUR-WS.org (2015)
42. Glimm, B., Kazakov, Y., Kollia, I., Stamou, G.: Lower and upper bounds for SPARQL queries over OWL ontologies. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI 2015). AAAI Press (2015)
43. Stoilos, G., Stamou, G.: Hybrid query answering over OWL ontologies. In: Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014) (2014)
44. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. *Sci. Am.* 96–101(2001)
45. Horrocks, I., Patel-Schneider, P.F.: The Generation of DAML+OIL. In: Proceedings of the 2001 Description Logic Workshop (2001)
46. Fensel, D., van Harmelen, F., Horrocks, I., McGuinness, D., Patel-Schneider, P.F.: OIL: an ontology infrastructure for the semantic web. *IEEE Intell. Syst.* **16**(2), 38–45 (2001)
47. Horrocks, I.: Ontologies and the semantic web. *Commun. ACM* **51**(12), 58–67 (2008)
48. Cuenca Grau, B., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P., Sattler, U.: OWL 2: the next step for OWL. *J. Web Semant.* **6**(4), 309–322 (2008)
49. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From SHIQ and RDF to OWL: the making of a web ontology language. *J. Web Semant.* **1**, 7–26 (2003)

50. RDF 1.1 concepts and abstract syntax. W3C Recommendation (2014). <https://www.w3.org/TR/rdf11-concepts/>
51. RDF Schema 1.1. W3C Recommendation (2014). <https://www.w3.org/TR/2014/PER-rdf-schema-20140109/>
52. SPARQL Query Language for RDF. W3C Recommendation (2008). <http://www.w3.org/TR/rdf-sparql-query/>
53. SPARQL 1.1 Query Language. W3C Recommendation (2013). <http://www.w3.org/TR/sparql11-query/>
54. SPARQL 1.1 1.1 Entailment Regimes. W3C Recommendation (2013). <https://www.w3.org/TR/sparql11-entailment/>
55. OWL Web Ontology Language document overview. W3C Recommendation (2004). <http://www.w3.org/TR/owl-features/>
56. OWL 2 Web Ontology Language document overview (second edition). W3C Recommendation (2012). <http://www.w3.org/TR/owl2-overview/>
57. R2RML: RDB to RDF Mapping Language. W3C Recommendation (2012). <https://www.w3.org/TR/r2rml/>

Reasoning Web. Semantic Interoperability on the Web  
13th International Summer School 2017, London, UK,  
July 7-11, 2017, Tutorial Lectures

Ianni, G.; Lembo, D.; Bertossi, L.; Faber, W.; Glimm, B.;  
Gottlob, G.; Staab, S. (Eds.)

2017, XI, 347 p. 63 illus., Softcover

ISBN: 978-3-319-61032-0