A Formal Framework for Coupling Document Spanners with Ontologies

Domenico Lembo  
Dip. di Ingegneria Informatica, Automatica e Gestionale  
Sapienza Università di Roma  
Rome, Italy  
lembo@diag.uniroma1.it

Federico Maria Scafoglieri  
Dip. di Ingegneria Informatica, Automatica e Gestionale  
Sapienza Università di Roma  
Rome, Italy  
scafoglieri@diag.uniroma1.it

Abstract—A significant portion of information that is nowadays collected in enterprises and organizations resides in text documents, and thus is inherently unstructured. Turning it into a structured form is the aim of information extraction (IE). Depending on the approach followed, the output of an IE process can fill forms or populate relational tables, or can be presented through an ontology. This last approach is particularly interesting, since ontologies may facilitate the integration with other corporate and external data, and enable data management and governance at an abstract, conceptual level, as in Ontology-based Data Access (OBDA). To this aim, OBDA uses declarative mappings that specify the relation between the ontology and the database to be accessed. In OBDA, however, only mappings based on (e.g., [6], [7]), whereas rule-based approaches encode learning techniques to learn the probabilistic models they are based on (e.g., [8], [9], [10])

Recently, Fagin et al. have carried out a foundational study on rule-based IE, and introduced a formal framework based on the notion of (document) spanner ([13], [14]). In brief, a spanner extracts from an input string a relation over its spans, which are pairs of indices referring to substrings to be extracted from D. Fagin et al. study possible representations of spanners and analyze how the use of some algebraic operations on the relations extracted from strings influences the spanner expressiveness. In particular, they study spanners defined by regular expressions with capture variables (a.k.a. “regex formulas”) and relational algebra.

In this paper we construct on their results and propose a formal framework for coupling document spanners with ontologies. Our basic idea is to adapt the OBDA framework to enable ontologies to be mapped to documents. We thus introduce the notion of Ontology-based document spanning (OBDS) system, in which, an ontology (more precisely, its intensional component) is linked to text documents through what we call extraction assertions, which in OBDS act exactly as mapping assertions in OBDA. Intuitively, an extraction assertion associates a document spanner P to a query q

1For additional references and a discussion on the literature on IE we refer the reader to [11] and [12].
specified over the ontology, with the intended meaning that the tuples of substrings corresponding to the spans returned by \( P \) must be among the answers to \( q \). In our framework we establish queries over the ontology to be conjunctive queries (CQs), whereas document spanners are defined as regex formulas extended with the relational algebra operators union, projection, join and string selection (the class of such spanners is denoted with \([\text{RGX}^{(\cup,\pi,\cdot,\cdot^*\cdot)}]\)). We remark that CQs are the most expressive queries for which query answering over ontologies has been shown to be decidable, and spanners in \([\text{RGX}^{(\cup,\pi,\cdot,\cdot^*\cdot)}]\) are among the most expressive document spanners considered in [13]. Notice also that extraction assertions we define resemble Global-Local-As-View (GLAV) mapping assertions used in data integration and in OBDA, and that their semantics, we have sketched above, corresponds to the classical sound interpretation for mappings [15], [16], [17].

After defining our framework, we investigate the problem of query answering, i.e., how to compute the answers to user queries posed over the ontology. We focus on CQs and analyze the case of DL-Lite\(_R\) OBDS systems, i.e., when the ontology is specified in the Description Logic (DL) DL-Lite\(_R\) [18], a popular lightweight ontology language that is at the basis of OWL 2 QL, one of the tractable profiles of OWL 2 [19]. Interestingly, in OBDA, when the ontology is in DL-Lite\(_R\) and mappings are GLAV, CQ answering is first-order rewritable, i.e., it can be reduced to the evaluation of a first-order (i.e., an SQL) query over the underlying database [20]. We recall that an OBDA mapping associates a query over the database to a query over the ontology, and the above result holds even when the queries over the database in the mapping are arbitrary first-order queries. Interestingly, also the rewriting can be specified in first-order, i.e., it has the same expressiveness of database queries used in the mappings (this has a practical fallout, since its evaluation can be delegated to the underlying database engine). A natural question is now whether a similar behaviour shows up also in DL-Lite\(_R\) OBDS systems, i.e., whether we can reduce query answering to the execution of a document spanner of the same kind of the spanners used in the mapping. We positively answer this question, by providing an algorithm that rewrites every CQ issued over a DL-Lite\(_R\) OBDS system (i.e., over its ontology) into a spanner belonging to \([\text{RGX}^{(\cup,\pi,\cdot,\cdot^*\cdot)}]\), and thus with the same expressiveness of spanners used in the mapping. Since evaluating such spanners is polynomial in the size of the input document, we can also conclude that CQ answering in this setting is polynomial in data complexity.

We finally observe that the use of ontologies in IE has been already widely considered in the literature (see [21] for a survey on the topic). However, none of the previous works on this matter has proposed a formal declarative framework, nor did they study the problem of query answering as we do in the present paper. Also, we believe that our framework paves the way for an in-depth investigation of the role of ontologies in IE, and in particular for the understanding of how reasoning over the ontology can help IE.

The rest of the paper is organized as follows: In Section II we give some preliminaries; in Section III we introduce our OBDS framework, in Section IV we provide our query rewriting algorithms for OBDSs systems equipped with DL-Lite\(_R\) ontologies; we conclude the paper in Section V.

II. PRELIMINARIES

A. Ontologies, Description Logics, and Queries

In the context of computer and information sciences, an ontology defines a set of representational primitives with which to model a domain of knowledge or discourse. Ontologies are defined in some formal language, which usually has its root in some kind of logic. Description Logics (DLs) are fragments of first-order logic that can be used to represent the knowledge of a domain of interest in a structured and formally well-understood way in order to reason upon it. DLs are thus well-suited to specify ontologies. DLs model the domain of interest in terms of objects, i.e., individuals, concepts, that are abstractions for sets of objects, and roles, that denote binary relations between objects [22]. DL ontologies are widely used in the context of the Semantic Web, and indeed are at the basis of OWL 2, the W3C standard for specifying ontologies on the web [23].

In this work we consider ontologies specified in DL-Lite\(_R\), which is a member of the class DL-Lite of tractable DLs, and is the logical basis of the profile OWL 2 QL [19]. DL-Lite\(_R\) expressions are given by the following syntax:

- Concept Expressions:

  \[ B ::= A \mid \exists Q \]
  \[ C ::= B \mid \neg B \]

- Role Expressions:

  \[ Q ::= P \mid P^\perp \]
  \[ R ::= Q \mid \neg Q \]

where \( A \) denotes an atomic concept (i.e., a named concept from the ontology signature), \( B \) a basic concept (i.e., an atomic role \( A \) or an existential restriction on a role \( \exists Q \)), \( C \) a general concept (i.e., a basic concept \( B \) or its negation \( \neg B \)), \( P \) an atomic role (i.e., a named role from the ontology signature), \( Q \) a basic role (i.e., an atomic role \( P \) or its inverse \( P^\perp \)), and \( R \) a general role (i.e., a basic role \( Q \) or its negation \( \neg Q \)).

A DL ontology \( O \) is a pair \( \langle T, A \rangle \) where:

- \( T \), called TBox, is the terminological component of \( O \), which contains statements representing intensional knowledge, and
- \( A \), called ABox, is the assertional component of \( O \), which represents extensional knowledge.

A TBox in DL-Lite\(_R\) is a finite set of assertions in form:

\[ B \sqsubseteq C \quad \text{(concept inclusion assertion)} \]
\[ Q \sqsubseteq R \quad \text{(role inclusion assertion)} \]

An ABox in DL-Lite\(_R\) is a finite set of membership assertions (i.e., facts) of the form:

\[ A(a) \quad \text{(concept membership assertion)} \]
\[ P(a,b) \quad \text{(role membership assertion)} \]
where \(a\) and \(b\) are constants (i.e., individual names). The formal semantic of DL language is given in terms of first-order (FOL) interpretations \(I\). An interpretation \(I = (\Delta^I, \cdot^I)\) consists of a nonempty interpretation domain \(\Delta^I\) and an interpretation function \(\cdot^I\) that assigns to each concept \(C\) a subset \(C^I\) of \(\Delta^I\), and to each role \(R\) a binary relation \(R^I\) over \(\Delta^I\), and to each individual name \(a\) and object \(o\) in \(\Delta^I\). Expressions and assertions in DL-Lite\(_R\) are interpreted as shown in the following table:

<table>
<thead>
<tr>
<th>Construct</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic Concept</td>
<td>(A)</td>
<td>(\Delta^I \subseteq \Delta^I)</td>
</tr>
<tr>
<td>Atomic Role</td>
<td>(P)</td>
<td>(P^I \subseteq \Delta^I \times \Delta^I)</td>
</tr>
<tr>
<td>Concept Negation</td>
<td>(\neg A)</td>
<td>(\Delta^I \setminus B^I)</td>
</tr>
<tr>
<td>Inverse Role</td>
<td>(P^{-})</td>
<td>\{(o', o)</td>
</tr>
<tr>
<td>Existential Rest.</td>
<td>(\exists \vec{y})</td>
<td>({ o</td>
</tr>
<tr>
<td>Role Negation</td>
<td>(\neg Q)</td>
<td>(\Delta^I \setminus \Delta^I \times \Delta^I \setminus Q^I)</td>
</tr>
<tr>
<td>Concept inclusion</td>
<td>(C_1 \subseteq C_2)</td>
<td>(C_1^I \subseteq C_2^I)</td>
</tr>
<tr>
<td>Role Inclusion</td>
<td>(Q \subseteq R)</td>
<td>(Q^I \subseteq R^I)</td>
</tr>
<tr>
<td>Membership Ass.</td>
<td>(A(a))</td>
<td>(\vec{a} \in A^I)</td>
</tr>
<tr>
<td>Membership Ass.</td>
<td>(P(a, b))</td>
<td>((\vec{a}, b^2) \in P^I)</td>
</tr>
</tbody>
</table>

An interpretation \(I\) is a model of an ontology \(O\) if it satisfies all assertions in \(T\) and \(A\). We denote with \(\text{Mod}(O)\) the set of all models of an ontology. We also say that \(O\) infers a sentence \(\psi\) if \(\psi^I\) evaluates to true in every \(I \in \text{Mod}(O)\).

A query is an open formula of function free first-order logic (FOL), i.e., a formula of the form:

\[
\{ \vec{x} | \exists \vec{y}\cdot \phi(\vec{x}, \vec{y}) \}
\]

where \(\exists \vec{y}\cdot \phi(\vec{x}, \vec{y})\) is a FOL formula with free variables \(\vec{x}\) and existentially quantified variables \(\vec{y}\), possibly containing constants. The number of variables in \(\vec{x}\) is the arity of the query. Among FOL queries, we in particular consider conjunctive queries (CQs), i.e., FOL queries in which \(\exists \vec{y}\cdot \phi(\vec{x}, \vec{y})\) is a conjunction of the form \(\exists \vec{y}\cdot p_1(\vec{x}, \vec{y}_1) \land \ldots \land p_n(\vec{x}, \vec{y}_n)\), where each \(p_i(\vec{x}, \vec{y}_i)\) is an atom, \(\vec{x} = \cup_{i=1}^n \vec{x}_i\) and \(\vec{y} = \cup_{i=1}^n \vec{y}_i\). When queries are specified over an ontology, each predicate \(p_i\) in every atom is either an atomic concept or an atomic role from the ontology signature. An extension of CQs are unions of conjunctive queries. A union of conjunctive queries (UCQ) is a FOL query of the form \(\{ \vec{x} | \exists \vec{y}_1\cdot \phi_1(\vec{x}, \vec{y}_1) \lor \ldots \lor \exists \vec{y}_n\cdot \phi_n(\vec{x}, \vec{y}_n) \}\), such that each \(\{ \vec{x} | \exists \vec{y}_i\cdot \phi_i(\vec{x}, \vec{y}_i) \}\) is a CQ. To simplify notation, throughout the paper we can write UCQ as sets of CQs. Query answering over an ontology amount to computing the so-called certain answers, i.e., those answers that hold in all models of the ontology. Formally, given a query \(q\) of the form (1) over an ontology \(O = (T, A)\), a tuple \(\vec{c}\) of constants is a certain answer to \(q\) if \(O \models \exists \vec{y}\cdot \phi(\vec{c}, \vec{y})\). In the following, we can write \(q(\vec{x})\) to denote a query of the form (1) with free variables \(\vec{x}\). Also, given a tuple of constants \(\vec{c}\), we can write \(q(\vec{c})\) to denote \(\exists \vec{y}\cdot \phi(\vec{c}, \vec{y})\).

### B. Document Spanners

We recall below the framework of document spanners studied by Fagin et al. in [13, 14].

1) Strings and spans: We fix a finite alphabet \(\Sigma\) of symbols, which we assume totally ordered. In particular, in the following examples \(\Sigma\) is composed by the lower and capital letters of English alphabet, the full stop ("."), and the underscore ("_"), which stands for the whitespace.

We denote by \(\Sigma^+\) the set of all finite strings over \(\Sigma\), and by \(\Sigma^*\) the set of all finite strings of length at least one over \(\Sigma\). A language over \(\Sigma\) is a subset of \(\Sigma^*\). A document \(D\) is simply a finite string over \(\Sigma\), i.e., \(D = \sigma_1 \ldots \sigma_n\) with \(n \geq 0\) and \(\sigma_i \in \Sigma\) for \(i = 1, \ldots, n\). In other terms, \(D \in \Sigma^*\).

A span identifies a substring of \(D\) by specifying its bounding indices. Formally a span of \(D\) has the form \([i, j]\), where \(1 \leq i \leq j \leq n+1\). If \([i, j]\) is a span of \(D\), then \(D\{i, j\}\) denotes the substring \(\sigma_i \ldots \sigma_{j-1}\). Note that \(D\{i, i\}\) is the empty string, and that \(D\{i, n+1\}\) is \(D\). Two spans \([i, j]\) and \([i', j']\) are equal if and only if \(i = i'\) and \(j = j'\).

We denote by \(\text{Spans}(D)\) the set of all the spans of \(D\).

**Example 1.** Consider the document \(D_{\text{ex}}\) given in Figure 1, and the span \([11, 19]\). It identifies the substring Einstein, i.e., \(D_{[11,19]} = \text{Einstein}\).

We fix an infinite set \(\text{SVars}\) of (span) variables; spans may be assigned to these variables. The sets \(\Sigma^+\) and \(\text{SVars}\) are disjoint. For a finite set \(V \subseteq \text{SVars}\) of variables and document \(D \in \Sigma^*\), a \((V, D)\)-tuple is a mapping \(\mu : V \rightarrow \text{Spans}(D)\) that assigns a span of \(D\) to each variable in \(V\). A \((V, D)\)-relation is a set of \((V, D)\)-tuples. A document spanner (or just spanner for short) is a function \(P\) over a finite set \(V \subseteq \text{SVars}\) of variables, that maps a document \(D\) to a \((V, D)\)-relation.

In the following, we use \(\text{SVars}(P)\) to refer to the set of variables of a spanner \(P\). We may also use \(P(v_1, \ldots, v_n)\) to denote a spanner \(P\) over variables \(V = \{v_1, \ldots, v_n\}\). Furthermore, given a document \(D\), we write \(\text{eval}(P, D)\) to denote the \((V, D)\)-relation that maps \(\text{SVars}(P)\) to \(D\).

**Example 2.** In Figure 2 we provide an example of \((V, D)\)-relation, for the spanner \(\gamma_{\text{tok}}\), such that \(\text{SVars}([\gamma_{\text{tok}}]) = \{x\}\). \((V, D)\)-tuples listed in this figure correspond to the words of the document \(D_{\text{ex}}\) in Figure 1.

### C. Spanner Representation

By a spanner representation system we refer collectively to any manner of specifying spanners through finite objects. In the next subsections we recall the regex formula (a primitive spanner representation) and the spanner algebra (an extension of regex formula with relational algebra operators).

1) Regex Formulas: As defined in [13], a variable regex is an extension of a regular expression with capture variables. Its grammar is defined as follows:

\[
\gamma := \emptyset | \epsilon | \sigma | (\gamma \lor \gamma) | (\gamma \cdot \gamma) | \gamma^* | x\{\gamma\}
\] (2)
The symbol ∅ defines the empty set, ε is the empty string, and σ ∈ Σ. The ∨, ·, and * symbols denote disjunction, concatenation, and the Kleene-star operators, respectively. x(σ) instead indicates that the match obtained through the variable regex γ is mapped (in the form of a span) to the variable x ∈ SVars. Parenthesis may be used in a variable regex in the usual way to specify precedence between operators. We denote by SVars(γ) the set of variables that occur in γ. We use γ as abbreviations γγ, and [σi − σj] as a shortcut for the disjunction of all characters σ ∈ Σ such that σi ≤ σ ≤ σj.

In this paper we consider only variable regex expressions that are union compatible, i.e., a regex formula assigning to x ∈ SVars. Parenthesis may be used in a variable regex in the usual way to specify precedence between operators. We denote by SVars(γ) the set of variables that occur in γ. We use γ as abbreviations γγ, and [σi − σj] as a shortcut for the disjunction of all characters σ ∈ Σ such that σi ≤ σ ≤ σj.

Example 3. Consider the following (simplified) regex formulas system:

- γtok = (ε ∨ (Σ∗ · _)) · x[A−zA−Z]+· (_ · _), i.e., a regex formula assigning to x the words in a document (that is, every non-empty sequence of alphabetic characters preceded by either a space or an empty string, and followed by either a fullstop or a space);
- γcap = (ε ∨ (Σ∗ · _)) · x[A−Z]· Σ∗· (_ · _), i.e., a regex formula assigning to x the words in a document that begin with a capital letter;
- γ aft_prof = (Professor · _) · xΣ+· (_ · _), i.e., a regex formula assigning to x the words in a document that follow the word Professor (plus a white space).

A regex formula γ is naturally viewed as representing a spanner, and by [γ] we denote the spanner that is represented by γ.

2) Algebra Over Spanner: The class \( RGX^{(∪, π, δ, ε)} \) denotes the class of dynamic spanners extended with the three basic operators of relation algebra: union (\( ∪ \)), projection (\( π \)), and join (\( δ \)) plus the string selection operator (\( ε \)). When we apply these symbols on one or more spanners, the results will be a new spanner. More precisely, let \( P_1 \) and \( P_2 \) be three spanners, such operators are defined as follows [13]:

- **Union.** The union \( P_1 ∪ P_2 \) is defined when \( P_1 \) and \( P_2 \) are union compatible, that is, \( SVars(P_1) = SVars(P_2) \). In that case, \( SVars(P_1 ∪ P_2) = SVars(P_1) \) and \( eval(P_1 ∪ P_2, D) = eval(P_1, D) ∪ eval(P_2, D) \).
- **Projection.** If \( θ ⊆ SVars \), then \( π_θ(P) \) is the spanner such that \( SVars(π_θ(P)) = θ \) and \( eval(π_θ(P), D) \) is obtained from \( eval(P, D) \) by restricting the domain of each \( (V, D) \)-tuple to \( θ \).
- **Join.** The join between spanners is defined as \( P_1 ⋈ P_2 \). It holds that \( SVars(P_1 ⋈ P_2) = SVars(P_1) \cup SVars(P_2) \), and \( eval(P_1 ⋈ P_2, D) \) consists of all \( (V, D) \)-tuples \( µ \) that agree with some \( µ_1 ∈ eval(P_1, D) \) and \( µ_2 ∈ eval(P_2, D) \).
- **String selection.** Let \( R \) be a k-ary string relation. The string-selection operator \( ez \) is parametrized by k variables \( x_1, \ldots, x_k \) in \( SVars(P) \), and may then be written as \( ez_{x_1,\ldots,x_k} \). If \( Py \) is \( ez_{x_1,\ldots,x_k} P \), then the span relation \( eval(P, D) \) is the restriction of \( eval(P, D) \) to those \( (V, D) \)-tuples \( µ \) such that \( (D_µ(x_1), \ldots, D_µ(x_k)) ∈ R \).

Let \( ρ \) be a regex formula, \([ρ] \) the spanner represented by \( ρ \), and \( x = x_1 \ldots x_n \) the sequence of n distinct variables in \( SVars([ρ]) \). Let \( y = y_1 \ldots y_n \) be a sequence of n distinct variables. The unary operator rename applied to \([ρ] \) returns a new spanner \([ρ[\hat{x} → y]] \), written also as \([ρ[x/y]] \), obtained by replacing every occurrence of \( x_i \) in \( [ρ] \), with \( y_i \).

Example 4. Using the regex formula defined in the previous section we can define, using the spanner algebra, the following more expressive and complex \( RGX^{(∪, π, δ, ε)} \)-spanner:

- \( [π_{prof}] = [γ_{cap}] ∞ [γ_{aft_prof}] \), i.e., the spanner represented by a regex formula that assigns to the variable x each word that both begins with a capital letter and follows the string Professor. The result of applying \( [π_{prof}] \) to the document \( D_{ex} \) in Figure 1 is shown in Figure 3. The extracted span is \([11,19] \) corresponding to the substring Einstein.

![Figure 1. Document D_{ex}](image1)

<table>
<thead>
<tr>
<th>eval([γ_{tok}], D_{ex})</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>µ1</td>
<td>[1,10]</td>
</tr>
<tr>
<td>µ2</td>
<td>[11,19]</td>
</tr>
<tr>
<td>µ3</td>
<td>[20,26]</td>
</tr>
<tr>
<td>µ4</td>
<td>[27,34]</td>
</tr>
<tr>
<td>µ5</td>
<td>[35,38]</td>
</tr>
<tr>
<td>µ6</td>
<td>[39,48]</td>
</tr>
<tr>
<td>µ7</td>
<td>[49,52]</td>
</tr>
<tr>
<td>µ8</td>
<td>[53,54]</td>
</tr>
<tr>
<td>µ9</td>
<td>[55,60]</td>
</tr>
</tbody>
</table>

![Figure 2. Spanner [γ_{tok}] applied to the document in Figure 1](image2)
In this section we introduce our formal framework. An OBDS System $\mathcal{E}$ is a pair $\langle \mathcal{T}, \mathcal{R} \rangle$, where

- $\mathcal{T}$ is a DL TBox,
- $\mathcal{R}$ is a set of extraction assertions of the form

$$P(v_1, \ldots, v_n) \rightsquigarrow \Psi(v_1, \ldots, v_n) \quad (3)$$

where

- $P(v_1, \ldots, v_n)$ (the left-hand side of the assertion) is a $\text{[RGX}\{\text{\underline{\ell}}, \pi, \kappa, \xi, \zeta\}\text{]}$-spanner.
- $\Psi(v_1, \ldots, v_n)$ (the right-hand side of the assertion) is a CQ with free variables $v_1, \ldots, v_n$. Atoms of this CQ are built as usual over $v_1, \ldots, v_n$, and possibly over other existentially quantified variables and/or constants, but may also use terms of the form $f(v_1, \ldots, v_m)$, where $f$ is a function symbol. The use of terms of this form allows us to denote individuals “constructed” from the spans in $\text{eval}(P, D)$.

An instance of OBDS system is given in Example 5. By focusing on the mapping, we note the presence of terms of the form $f(v_1, \ldots, v_m)$ (as $\text{prof}(x_1)$). Such terms are useful in all those cases in which the identifiers of individuals that are instances of the ontology do not appear in the underlying documents, but have to be constructed starting from the strings extracted from them. Note that a similar mechanism is adopted in OBDA, and in the W3C standard R2RML for mapping relational tables to RDF datasets, which adopts templates to construct IRIs denoting individuals².

The semantic of an OBDS system $\mathcal{E} = \langle \mathcal{T}, \mathcal{R} \rangle$ is defined with respect to a document $D$. Given such document, an interpretation $I$ is a model for $\mathcal{E}$ with respect to $D$ if:

- $I$ is a model for $\mathcal{T}$, and
- $\Psi(D_{\mu}(v_1), \ldots, D_{\mu}(v_n))$ evaluates to true in $I$ for each mapping $\mu \in \text{eval}(P, D)$.

We use $\text{Mod}(\mathcal{E}, D)$ to denote the set of models of $\mathcal{E}$ with respect to $D$. The notion of logical implication naturally extends to OBDS systems, i.e., given a sentence $\phi$ we write that $\langle \mathcal{E}, D \rangle \models \phi$ if $\phi^I$ for every $I \in \text{Mod}(\mathcal{E}, D)$.

Example 5. Let $\mathcal{E} = \langle \mathcal{T}, \mathcal{R} \rangle$ be a OBDS system where $\mathcal{T}$ is constituted by the following intensional axioms:

$$
\begin{align*}
t_1 & : \quad \text{Course} \sqsubseteq \neg \text{Person} \\
t_2 & : \quad \text{Professor} \sqsubseteq \text{Person} \\
t_3 & : \quad \exists \text{teaches}^{-} \sqsubseteq \text{Course} 
\end{align*}
$$

Such a TBox states that a course is not a person ($t_1$), every professor is a person ($t_2$) and that only course can be taught ($t_3$). Let $D_{\text{ex}}$ be the document in Figure 1, and let $[p_{\text{prof}}[x_1/x]]$ and $[\gamma_{\text{teaches}}]$ be two spanner programs. The former is represented by a regex formula that is obtained by a simple renaming of a regex formula given in Example 4 (the renaming will be useful for the examples of next section), whereas the regex formula representing the latter is given below:

- $\gamma_{\text{teaches}} = \_ : x_2 \{ \Sigma^+ \} \cdot (\_ \text{ taught} \_ ) ; y_2 \{ \Sigma^+ \} \cdot (\_ \_ \_ )$, i.e., a regex assigning to $x_2$ the words before the word taught, and to $y_2$ the words after taught.

$\mathcal{R}$ is as follows:

$$
\begin{align*}
M_1 & : \ [p_{\text{prof}}](x_1) \rightsquigarrow \text{Professor}(\text{prof}(x_1)) \\
M_2 & : \ [\gamma_{\text{teaches}}](x_2, y_2) \rightsquigarrow \text{teaches}(\text{prof}(x_2), \text{course}(y_2))
\end{align*}
$$

Notice that both $\text{prof}$ and $\text{course}$ are function symbols of arity 1 used to construct individuals from the string returned by the spanners. For instance, if the spanner in $M_1$ applied to a document returns the string Einstein, the assertions $M_1$ constructs the individual $\text{prof}(\text{Einstein})$ as an instance of $\text{Professor}$.

In an OBDS system $\mathcal{E} = \langle \mathcal{T}, \mathcal{R} \rangle$, computing the certain answers to a query $q$ with respect to a document $D$, denoted by $\text{cert}(q, \mathcal{E}, D)$, amounts to finding the query answers to $q$ that hold in all models for $\mathcal{E}$ with respect to $D$. More formally, given a query $q$ of the form (1), a tuple $\vec{c}$ is a certain answer to $q$ with respect to $D$ if $\langle \mathcal{E}, D \rangle \models q(\vec{c}, \vec{y})$.

IV. QUERY REWRITING IN $\text{DL-Lite}_R$ OBDS SYSTEMS

In this section we consider $\text{DL-Lite}_R$ OBDS systems, i.e., systems in which the ontology is expressed in $\text{DL-Lite}_R$, and devise an algorithm to compute the certain answers to a CQ over one such system with respect to a given document $D$.

To this aim we resort to rewriting techniques used in the context of OBDA, which we slightly adapt to our setting. First of all, we recall the formal definition of OBDA. An OBDA system $\mathcal{J}$ is a triple $\langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$, where $\mathcal{T}$ is a DL TBox, $\mathcal{S}$ is the relational schema of the source database, and $\mathcal{M}$ is a mapping between $\mathcal{T}$ and $\mathcal{S}$. The mapping $\mathcal{M}$ is a set of assertions of the form

$$
\Phi(x_1, \ldots, x_n) \rightsquigarrow \Psi(x_1, \ldots, x_n)
$$

where $\Psi(x_1, \ldots, x_n)$ (the right-hand side of the assertion) is exactly as for an extraction assertion in an OBDS system (cf. assertion (3)), whereas $\Phi(x_1, \ldots, x_n)$ (the left-hand side of the assertion) is a FOL query expressed over the schema $\mathcal{S}$.

The semantics of OBDA systems is similar to the semantics of OBDS systems, but is defined with respect to a database instance for $\mathcal{S}$. More precisely, given one such database DB, a model for $\mathcal{J}$ is any interpretation $I$ that satisfies $\mathcal{T}$ and such that for every tuple $(c_1, \ldots, c_n)$ in the evaluation of the query $\Phi(x_1, \ldots, x_n)$ over DB, $\Psi(c_1, \ldots, c_n)$ evaluates to true in $I$.

It is well-known that CQ answering over an OBDA system in which the TBox is expressed in $\text{DL-Lite}_R$ is FOL-rewritable, i.e., it can be reduced to the evaluation of a FOL query over the source database [18], [20]. That is, it can be reduced to

²https://www.w3.org/TR/r2rml/
the evaluation of a query written in the same language of the query used in the left-hand side of mapping assertions, which are the queries that the DBMS managing the source database is able to process.

In the following, by exploiting the similarities between the OBDA and OBDS frameworks, we show that CQ answering in DL-Lite\(_R\) OBDS systems \(\mathcal{J}\) is reducible to the evaluation of a \([\text{RGX}^{\{\cup, \text{\&}, \mathcal{R}\}}]\)-spanner over a document \(\mathcal{D}\), i.e., a spanner of the same expressiveness of those allowed in the left-hand side of the extraction assertions in \(\mathcal{J}\).

To maintain the treatment simple, from now on we consider extraction assertions that do not use terms of the form \(f(x_1, \ldots, x_n)\). Intuitively, this corresponds to the assumption that the identifiers of objects that are instances of the ontologies are extracted directly from the documents over which spanners are executed. Our algorithms however can be adapted to manage the presence of terms of this form by using the same techniques showed in [16]. Thus our results apply straightforwardly to general OBDS extraction assertions.

Below we first consider the case in which extraction assertions do not allow for existential variables in their right-hand side, i.e., assertions do not allow for existential variables in their right-hand side of each extraction rule. The extraction assertions are specified in the previous “splitted” form. We thus assume in the following that GA\(V\) extraction assertions are equivalent to the set of assertions

\[
P(\vec{v}) \leadsto p_1(\vec{v}_1) \land \ldots \land p_k(\vec{v}_k)
\]

where each \(p_i\) is either an atomic concept or an atomic role in the ontology (notice that in the former case \(\vec{v}_i\) is a single variable, in the latter contains two variables), and \(\cup_{i=1}^k \vec{v}_i \subseteq \vec{v}\). It is easy to see that the previous extraction assertion is equivalent to the set of assertions\(^3\)

\[
P(\vec{v}) \leadsto p_1(\vec{v}_1) \\
\ldots \\
P(\vec{v}) \leadsto p_k(\vec{v}_k)
\]

We thus assume in the following that GA\(V\) extraction assertions are specified in the previous “splitted” form.

Given a GA\(V\) DL-Lite\(_R\) OBDS system \(\mathcal{E} = \langle \mathcal{T}, \mathcal{R} \rangle\) and a query \(q\) over \(\mathcal{E}\), we rewrite \(q\) in two steps, which we call ontology-based rewriting and extraction rules-based rewriting. The first step is aimed at compiling the TBox into the query. The second is aimed at rewriting the query obtained in the first step (which is still a query expressed over the ontology) according to the assertions in \(\mathcal{R}\), so that the final result is a document spanner. For the first step we adopt the algorithm PerfectRef presented [18]. According to this algorithm, the positive inclusion assertions in the TBox are used as rewriting rules, from right to left, to repeatedly rewrite atoms in the query. When an atom is rewritten a new CQ is added to the result, as long as a fix point is reached. The final rewriting is indeed a Union of CQs. For example, given a TBox assertion \(B \subseteq A\), and a query \(\{x \mid A(x)\}\) the atom \(A(x)\) is rewritten into \(B(x)\) and the query \(\{x \mid B(x)\}\) is added to the result. Notice however that for an atom to be rewritten according to an inclusion assertion in \(\mathcal{T}\) its terms must respect some syntactic conditions [18]. Moreover, when atoms in the query unify, PerfectRef performs such unification, which may then trigger some further atom rewriting. An example of execution of PerfectRef is given in Example 6.

The second step is through an unfolding method, which, roughly, substitutes, in all possible ways, each atom \(a\) in the query returned by PerfectRef with the spanners occurring in the left-hand side of extraction assertions referring to \(a\). To this aim, we use the procedure Unfolding, which takes as input a UCQ \(\mathcal{Q}\) and a set of extraction rules \(\mathcal{R}\). This procedure, for each CQ \(q \in \mathcal{Q}\), each atom \(p_i(\vec{t}_i)\) in \(q\) (where \(\vec{t}_i\) is a tuple of terms, i.e., variables and/or constants), and each extraction rule \(P(\vec{v}) \leadsto p_i(\vec{v}_i)\), computes a unification (if it exists) between \(p_i(\vec{t}_i)\) and \(p_i(\vec{v}_i)\), and substitutes \(p_i(\vec{t}_i)\) with \(P(\vec{v})\), modulo the application of the unifier. Notice that only queries having all atoms that unify with at least one extraction assertions are completely unfolded and returned by Unfolding. Finally, Unfolding expresses projections, selections and joins in the queries according to the \([\text{RGX}^{\{\cup, \text{\&}, \mathcal{R}\}}]\) syntax. An example of unfolding is given in Example 6.

The rewriting algorithm for the GA\(V\) case is given below.

**Algorithm OBDS\_Rewriting\_GA\(V\)(\(\mathcal{E}, q\))**

**Input:** OBDS \(\mathcal{E} = \langle \mathcal{T}, \mathcal{R} \rangle\), such that \(\mathcal{T}\) is a DL-Lite\(_R\) TBox and \(\mathcal{R}\) is a set of GA\(V\) extraction assertions, CQ \(q\)

**Output:** \(P \in [\text{RGX}^{\{\cup, \text{\&}, \mathcal{R}\}}]\)

begin

\(Q = \text{PerfectRef}(\mathcal{T}, q)\)

\(P = \text{Unfolding}(Q, \mathcal{R})\)

return \(P\)

end

We can compute the certain answers to CQs over GAV DL-Lite\(_R\) OBDS systems by means of the algorithm OBDS\_Rewriting\_GAV, as established by the following theorem.

**Theorem 1.** Let \(\mathcal{E} = \langle \mathcal{T}, \mathcal{R} \rangle\) be an OBDS system such that \(\mathcal{T}\) is a DL-Lite\(_R\) TBox and \(\mathcal{R}\) is GAV, let \(\mathcal{D}\) be a document, let \(q\) be a CQ of arity \(n\) over \(\mathcal{E}\), and let \(P(v_1, \ldots, v_n)\) be the spanner returned by OBDS\_Rewriting\_GAV(\(\mathcal{E}, q\)). Then, a tuple of constants \((t_1, \ldots, t_n) \in \text{cert}(q, \mathcal{E}, \mathcal{D})\) if and only if there exists \(\mu \in \text{eval}(P, \mathcal{D})\) such that \(t_i = D_{\mu(v_i)}\) for each \(i \in 1, \ldots, n\). Furthermore \(P \in [\text{RGX}^{\{\cup, \text{\&}, \mathcal{R}\}}]\).

**Proof (sketch).** The result follows from the following facts:

(i) PerfectRef\((q, \mathcal{T})\) returns the perfect rewriting of a CQ \(q\)
with respect to a DL-Lite$_R$ $\mathcal{T}$, i.e., given an ABox $\mathcal{A}$ the certain answers to $q$ over $\langle \mathcal{T}, \mathcal{A} \rangle$ coincide with the evaluation of $q$ over $\mathcal{A}$, seen as a database [18]; (ii) the correctness of the unfolding procedure to rewrite queries over the sources in GAV systems [16], and (iii) the fact that $[\text{RGX}^{\cup,\pi,\kappa,\land}]$ is closed under union, projection, join and string selection [13].

**Example 6.** Consider the setting of Example 5, and the following query $q$ that asks for the persons who teach a course:

$$q = \{ x \mid \exists y. \text{Person}(x) \land \text{teaches}(x, y) \land \text{Course}(y) \}$$

the result of PerfectRef($\mathcal{T}, q$) is the UCQ $Q'$ containing the following CQs:

$$\begin{align*}
q'_1 &= \{ x \mid \exists y. \text{Person}(x) \land \text{teaches}(x, y) \land \text{Course}(y) \} \\
q'_2 &= \{ x \mid \exists y. \text{Professor}(x) \land \text{teaches}(x, y) \land \text{Course}(y) \} \\
q'_3 &= \{ x \mid \text{Person}(x) \land \text{teaches}(x, \_ ) \} \\
q'_4 &= \{ x \mid \text{Professor}(x) \land \text{teaches}(x, \_ ) \}
\end{align*}$$

The query $q'_2$ is generated from $q'_1$ by the rewriting of $\text{Person}(x)$ in $\text{Professor}(x)$ according to the inclusion assertion ($t_2$). The query $q'_3$ is obtained from $q'_1$ after rewriting $\text{Course}(y)$ into $\exists z. \text{teaches}(z, y)$ (accoring to inclusion ($t_3$)) and after a unifying $\text{teaches}(x, y)$ and $\text{teaches}(y, z)$. Similarly for $q'_4$, which is derived from $q'_2$.

After the execution of PerfectRef, Unfolding($Q', \mathcal{R}$) unfolds the queries in $Q'$ using extraction assertions in $\mathcal{R}$. Note that in our example only $q'_1$ can be completely unfolded, whereas other queries in $Q'$ do not contribute to the final rewriting since some of their arguments cannot be unfolded. For $q'_1$, atom $\text{Professor}(x)$ unifies with the atom $\text{Professor}(\text{prof}(x))$ in the extraction assertion $M_1$ and atom $\text{teaches}(x, y)$ unifies with the atom $\text{teaches}(\text{prof}(x_2), \text{course}(y_2))$ in the extraction assertion $M_2$. The unfolding will thus produce the following spanner (modulo the application of the function symbol prof): }\]

$[\rho_{unf}] = [\pi_{z}([\rho_{\text{prof}}(z/x)] \times [\gamma_{\text{teaches}}(z/x_2)])]$

\[\text{eval}([\rho_{unf}]; \mathcal{D}_{\text{ex}}), \text{where } \mathcal{D}_{\text{ex}} \text{ is the document in Figure 1, is the span [11, 19]. Then, cert(q, \mathcal{E}, \mathcal{D}) \text{ are the tuples prof}([\text{eval}([\rho_{unf}]; \mathcal{D}]), \text{ that in our case is prof}(\text{Einstein})]. \]

**B. General extraction assertions**

We now consider the case in which we do not pose any restriction on the extraction assertions, i.e., they are GLAV. We start by noting that an extraction assertion $P(v_1, \ldots, v_n) \rightsquigarrow \Psi(v_1, \ldots, v_n)$ can be always transformed into a pair of assertions of the following form

\[\begin{align*}
P(v_1, \ldots, v_n) &\rightsquigarrow R_{\text{aux}}(v_1, \ldots, v_n) \quad (5) \\
R_{\text{aux}}(v_1, \ldots, v_n) &\rightsquigarrow \Psi(v_1, \ldots, v_n) \quad (6)
\end{align*}\]

where $R_{\text{aux}}$ is an auxiliary predicate symbol that explicitly denotes the relation returned by the execution of $P$ over a document $\mathcal{D}$, i.e., $R_{\text{aux}}$ denotes the set of tuples of strings $(\mathcal{D}_{\mu(v_1)}, \ldots, \mathcal{D}_{\mu(v_n)})$ for each $\mu \in \text{eval}(P, \mathcal{D})$. It is easy to see that the second assertion above is an OBDA GLAV mapping assertion (in fact a LAV assertion [15]), whereas we can treat the first assertion above as a GAV extraction assertion (even though its right-hand side contains an $n$-ary predicate). Our rewriting algorithm thus proceeds as follows:

(i) we first rewrite the user query according to the ontology (ontology-based rewriting); this steps can be done through PerfectRef and returns a UCQs over the ontology;

(ii) Each CQ in the rewriting returned by PerfectRef is rewritten according to the LAV mapping assertions using predicate $R_{\text{aux}}$; to this aim we can use any of the well-known algorithm to rewrite a CQ with respect to a set of LAV mappings in the relational framework [24]; Specifically, we use the MiniCon algorithm proposed in [25], which, in our framework, returns a UCQ over the predicate $R_{\text{aux}}$;

(iii) at this point it remains to simply unfold the query returned by the MiniCon algorithm; to this aim we can use the Unfolding procedure we have seen before.

The overall rewriting algorithm is given below. In the algorithm we denote with $R'$ extraction assertions of the form (5) and with $L$ LAV mapping assertions of the form (6).

**Algorithm OBDS_Rewriting(\mathcal{E}, q)**

**Input:** OBDS $\mathcal{E} = \langle \mathcal{T}, \mathcal{R} \rangle$, such that $\mathcal{T}$ is a DL-Lite$_R$ TBox

**Output:** $P \in [\text{RGX}^{\cup,\pi,\kappa,\land}]$

\[\begin{align*}
&\text{begin} \\
&\quad Q_T = \text{PerfectRef}(\mathcal{T}, q) \\
&\quad Q_L = \emptyset \\
&\quad \text{foreach } q \in Q_T \\
&\quad \quad Q_L = Q_L \cup \{ \text{MiniCon}(L, Q_T) \} \\
&\quad P = \text{Unfolding}(Q_L, R') \\
&\quad \text{return } P \\
&\text{end}
\]

The following theorem can be proved in a way similar to Theorem 1, considered also that MiniCon returns a perfect rewriting for a CQ over a relational schema with respect to a LAV mapping [25].

**Theorem 2.** Let $\mathcal{E} = \langle \mathcal{T}, \mathcal{R} \rangle$ be an OBDS system such that $\mathcal{T}$ is a DL-Lite$_R$, let $\mathcal{D}$ be a document, let $q$ be a CQ of arity $n$ over $\mathcal{E}$, and let $P(v_1, \ldots, v_n)$ be the spanner returned by OBDS_Rewriting(\mathcal{E}, q). Then, a tuple of constants $(t_1, \ldots, t_n) \in \text{cert}(q, \mathcal{E}, \mathcal{D})$ if and only if there exists $\mu \in \text{eval}(P, \mathcal{D})$ such that $t_i = \mathcal{D}_{\mu(v_i)}$ for each $i \in 1, \ldots, n$.

Furthermore $P \in [\text{RGX}^{\cup,\pi,\kappa,\land}]$.

Since executing a spanner in $[\text{RGX}^{\cup,\pi,\kappa,\land}]$ is polynomial in the size of the underlying document [13] (i.e., the length of the string corresponding to the document), the following result follows straightforwardly from Theorem 2.

**Corollary 1.** Let $\mathcal{E} = \langle \mathcal{T}, \mathcal{R} \rangle$ be an OBDS system such that $\mathcal{T}$ is a DL-Lite$_R$ TBox, let $\mathcal{D}$ be a document and let $q$ be a CQ over $\mathcal{E}$. Then computing $\text{cert}(q, \mathcal{E}, \mathcal{D})$ is polynomial in the size of $\mathcal{D}$. 


V. CONCLUSIONS

The research in the OBDS framework can be continued in many directions. From the theoretical perspective, it would be interesting to investigate query answering in OBDS systems with more expressive languages for the ontology. First candidates are other tractable logics of the DL-Lite family, like DL-LiteA, which extends DL-LiteR with functionalities on roles (in a controlled way). It is known that, differently from DL-LiteR, query answering for DL-LiteA TBoxes and GLAV mappings in OBDA is not FOL-rewritable. This is caused by the interaction of existential variables in right-hand side of mapping assertions and functionalities in the TBox. How this interaction affects query answering in OBDS systems, and whether it is possible to solve query answering by rewriting in spanners of the $R_{\text{RGX}}^{(\cup, \pi, \sigma, C)}$ class is left for future study. We also plan to investigate query answering in OBDS systems in which the TBox is specified in other DLs for which standard query answering over ontologies is polynomial in data complexity, e.g., $\mathcal{EL}$ [26], or horn DLs [27], [28].

More in general, we believe that our framework paves the way for a comprehensive study on the use of ontologies in IE, and it can help understanding how reasoning services over the ontology may improve IE. For example, we believe that in our framework it is possible to exploit reasoning to identify anomalies in the specification of extraction rules (e.g., inconsistencies), in the spirit of the work on mapping analysis in OBDA [29]. Finally, an obvious future line of research is to develop software tools for OBDS, in order to verify the realizability of our approach in the practice.

REFERENCES


