International Journal of Semantic Computing

Ontology-based Document Spanning Systems for Information Extraction

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

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

Information Extraction (IE) is the task of automatically organizing in a structured form data extracted from free text documents. In several contexts, it is often desirable that extracted data are then organized according to an ontology, which provides a formal and conceptual representation of the domain of interest. Ontologies allow for a better data interpretation, as well as for their semantic integration with other information, as in Ontology-based Data Access (OBDA), a popular declarative framework for data management where an ontology is connected to a data layer through mappings. However, the data layer considered so far in OBDA has consisted essentially of relational databases, and how to declaratively couple an ontology with unstructured data sources is still unexplored.

By leveraging the recent study on *document spanners* for rule-based IE by Fagin et al., in this paper we propose a new framework that allows to map text documents to ontologies, in the spirit of OBDA. We investigate the problem of answering conjunctive queries in this framework. For ontologies specified in the Description Logics *DL-Lite*$_{\mathcal{R}}$ and *DL-Lite*$_{\mathcal{F}}$, we show that the problem is polynomial in the size of the underlying documents. We also provide algorithms to solve query answering by rewriting the input query on the basis of the ontology and its mapping towards the source documents. Through these techniques we pursue a virtual approach, similar to that typically adopted in OBDA, which allows us to answer a query without having to first populate the entire ontology. Interestingly, for *DL-Lite*$_{\mathcal{R}}$ both the spanners used in the mapping and the one computed by the rewriting algorithm belong to the same class of expressiveness. This holds also for *DL-Lite*$_{\mathcal{F}}$, modulo some limitations on the form of the mapping. These results say that in these cases our framework can be easily implemented by decoupling ontology management and document access, which can be delegated to an external IE system able to compute the extraction rules we use in the mapping.

Keywords: Information Extraction; Ontology Based Data Access; Knowledge Representation.

1. Introduction

A huge portion of information is nowadays spread in free-text documents, like reports, e-mails, web pages, articles, etc. These sources are obviously tailored for the human reading, but it is also often desirable within an organization that relevant data contained therein is extracted and integrated with other corporate data. Information Extraction (IE) is the task

of automatically performing such extraction and organizing gathered data into a structured representation, typically a spreadsheet, database, or even a knowledge base [1, 2].

IE has been intensively studied starting from the late '80s [3], and since then several extraction methods have been proposed, which, in broad terms, can be classified as either statistical or rule-based [4]. In the former case, IE is based on probabilistic models, e.g., classifiers or sequence models, (e.g., [5,6]). Instead, rule-based approaches encode specific extraction tasks into rules, mostly corresponding to finite-state transducers (e.g., [7–9]).

Recently, Fagin et al. [10, 11] have initiated a foundational study on rule-based IE, and proposed a new framework for it constructed on the notion of *(document) spanner*. In a nutshell, a spanner is a program that extracts from a text document $\mathbf{D}$ (i.e., a string) a relation containing tuples of *spans*, which are pairs of indices identifying substrings of $\mathbf{D}$. For example if $\mathbf{D}$ is the string Albert_Einstein_from_Ulma, the span $[8, 16\rangle$ selects the substring Einstein which is the slice of $\mathbf{D}$ going from the eighth to the fifteenth character in $\mathbf{D}$ (by definition, a span $[i, j\rangle$ goes from position i to position j-1, included). Fagin et al. have in depth investigated how to represent spanners and how to combine them through algebraic operators. In particular, they have studied spanners defined by regular expressions with capture variables (called "*regex formulas*") and operators adapted from *relational algebra*.

Whereas the relations returned by the spanners represent a basic way of structuring the extracted information, more rich and semantically meaningful representations are typically desired, to better interpret data and integrate them with the information asset of an organization. This calls for the definition of a conceptual and formal representation of the application domain and for its coupling with the data extracted from documents.

Ontologies have proved themselves over the years to be one of the best means to model knowledge at the conceptual level, and their role in information management is now unanimously recognized. The main advantages in the use of ontologies can be seen in their high-level, easy-to-understand, and unambiguous representation of a domain of discourse, as well as the possibility of reasoning over such representation to obtain all the information it infers (and not only the asserted one). Linking ontologies to data and providing (efficient) reasoning services over them is the main objective of *Ontology-based Data Access (OBDA)* [12,13]. In OBDA the ontology is coupled with external databases through a mapping, which declaratively specifies the semantic relationship between the ontology and the data. A user interacts only with the ontology, e.g., by posing queries, which are automatically processed by sophisticated algorithms that return the answers to the user by reasoning on the ontology and the mapping. In OBDA, however, ontologies have been essentially used so far only on top of relational databases, with very few exceptions (as, e.g., [14]), and how to access unstructured data, like those contained in text documents, using the ontologies as in OBDA is still unexplored.

In this paper we take a first step in this direction and propose a formal framework for coupling ontologies with spanners for IE from documents. Within this framework, we focus on the problem of query answering and provide some complexity results and practical algorithms for the case when the ontologies are specified in some languages of the *DL-Lite* family of Description Logics [15] and are coupled with expressive spanners. More in detail, our contributions can be summarized as follows:

- We introduce the notion of *Ontology-based document spanning (OBDS) system*. In an OBDS system, an ontology is linked to text documents through extraction assertions, which act similarly as mapping assertions in OBDA. Roughly speaking, an extraction assertion associates a document spanner P to a query q over the ontology, with the intended meaning that the tuples of strings corresponding to the spans returned by P evaluated over a text document must be among the answers to q evaluated over the ontology. An extraction assertion can be thus seen as a rule, where P is the body and q is the head.
- We study *query answering over an OBDS system*, i.e., how to answer a user query specified over the ontology by retrieving the answers from the text documents mapped to the ontology. We consider the case in which (i) the ontology is specified in either *DL-Lite*$_\mathcal{R}$ or *DL-Lite*$_\mathcal{F}$ [15], (ii) user's queries are CQs, (iii) spanners in the body of extraction assertions belong to the class $[\![\mathrm{RGX}^{\{\cup,\pi,\bowtie,\zeta^=\}}]\!]$, i.e., are defined as regex formulas extended with the relational algebra operators union, projection, join, and string selection [10], (iv) queries in the head of extraction assertions are CQs. We show that *query answering is in* PTIME *in data complexity* (i.e., the complexity computed only with respect to the size of the underlying documents). We remark that *DL-Lite*$_\mathcal{R}$ and *DL-Lite*$_\mathcal{F}$ are two popular ontology languages, tailored to OBDA and to deal with large datasets[a], CQs are the most expressive queries for which query answering over ontologies has been shown to be decidable, and spanners in $[\![\mathrm{RGX}^{\{\cup,\pi,\bowtie,\zeta^=\}}]\!]$ are among the most expressive document spanners considered in [10]. We note also that extraction assertions we define resemble GLAV mapping assertions used in data integration and in OBDA, i.e., the most expressive form of mappings adopted in these contexts [12, 17–19].
- We investigate *query rewriting in OBDS systems*, i.e., whether it is possible to answer a query by first rewriting it and then evaluating the rewriting over the data layer. Our aim is to understand whether we can reduce query answering to the execution of a document spanner of the same kind of those used in the extraction assertions. We notice that a similar property is considered crucial in OBDA, where one typically looks for the so-called first-order rewritability of query answering, that holds when query answering can be solved by evaluating over the source database a first-order query computed independently from the data. This indeed means that such evaluation can be delegated to the DBMS managing the source data. In our OBDS framework, we positively answer the above question for the case in which ontologies are specified in *DL-Lite*$_\mathcal{R}$. We indeed provide an algorithm that rewrites every CQ issued over an OBDS system (i.e., over its ontology) into a spanner belonging to $[\![\mathrm{RGX}^{\{\cup,\pi,\bowtie,\zeta^=\}}]\!]$. We also show that the same holds when the ontology is expressed in *DL-Lite*$_\mathcal{F}$ and extraction assertions are GAV, i.e., they have in their heads only CQs without existential variables. We believe that these results have an interesting practical fallout, since in these cases it is possible to delegate the evaluation of the rewriting to same engine that is in charge

[a] In particular, *DL-Lite*$_\mathcal{R}$ is the formal counterpart of OWL 2 QL, one of the tractable profiles of OWL 2 [16]

of evaluating the spanners in the body of the extraction assertions. Interestingly, as IE engine we can use an off-the-shelf tool like IBM SystemT [8], whose AQL language allows for expressing spanners belonging to $[\![\mathrm{RGX}^{\{\cup,\pi,\bowtie,\zeta^{=}\}}]\!]$. Also, the modular nature of our rewriting technique seems to streamline the incorporation of OBDS capabilities into current OBDA engines (e.g., [20, 21]).

We conclude this section by observing that the use of ontologies in IE has been already widely considered in the literature (see [22] for a survey on the topic). However, none of the previous works on this matter has proposed a formal declarative framework for the adoption of ontologies in IE, nor did they study the problem of query answering, with the exception of [23], of which the present paper is an extended version. 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 2 we give some preliminaries. In Section 3 we introduce our OBDS framework, and in Section 4 we establish our complexity results on query answering. Then, in Section 5, we provide our query rewriting algorithms for OBDSs systems equipped with *DL-Lite*$_{\mathcal{R}}$ or *DL-Lite*$_{\mathcal{F}}$ ontologies. We finally conclude the paper in Section 6.

2. Preliminaries

In this section we first recall some basic notions on Description Logic (DL) ontologies and on queries over them. Then, we turn our attention to document spanners and describe the formal framework proposed by Fagin et al. in [10, 11].

2.1. *Description Logic ontologies*

Description Logics (DLs) [24] are decidable fragments of first-order logic (FOL) that are largely recognized as one of the best means to specify ontologies, being them formally well-understood and equipped with powerful mechanisms to reason upon the representations they allow to specify. DLs model the domain of interest in terms of *objects*, a.k.a., individuals, *concepts*, that are abstractions for sets of objects, and *roles*, that are binary relations between objects. They 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 [25].

Formally an DL ontology $\mathcal{O}$ is defined as a pair $\langle \mathcal{T}, \mathcal{A} \rangle$ where:

- $\mathcal{T}$, called TBox, is the *terminological component*, which contains assertions (i.e., closed formulas of the logic, a.k.a. sentences) representing intensional knowledge, and
- $\mathcal{A}$, called ABox, is the *assertional component*, which contains assertions representing extensional knowledge.

From now on we assume to have a fixed infinite countable alphabet Γ of names for concepts, roles and individuals. The formal semantic of a DL language is given in terms of FOL interpretations. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ over Γ consists of a non-empty

set $\Delta^{\mathcal{I}}$ (the interpretation domain) and an interpretation function $\cdot^{\mathcal{I}}$ that assigns to each concept C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and to each role R a binary relation $R^{\mathcal{I}}$ over $\Delta^{\mathcal{I}}$, and to each individual a an object $o \in \Delta^{\mathcal{I}}$.

An interpretation $\mathcal{I}$ is a model of an ontology $\mathcal{O}$ if $\mathcal{I}$ satisfies all the assertions in $\mathcal{T}$ and $\mathcal{A}$. The satisfaction of an assertion is defined in the standard way for DLs [24]. We denote with $Mod(\mathcal{O})$ the set of all models of an ontology $\mathcal{O}$. We say that $\mathcal{O}$ is satisfiable if $Mod(\mathcal{O}) \neq \emptyset$, unsatisfiable, otherwise. We also say that $\mathcal{O}$ entails a FOL sentence ψ, denoted $\mathcal{O} \models \psi$, if $\psi^{\mathcal{I}}$ evaluates to true in every $\mathcal{I} \in Mod(\mathcal{O})$, where $\psi^{\mathcal{I}}$ denote the standard interpretation of a FOL sentence [26].

In this work we will focus on ontologies expressed in *DL-Lite*, a family of DLs particularly suited for specifying ontologies on top of large data repositories [15, 27], and that is at the basis of `OWL 2 QL`, one of the tractable profiles of `OWL 2` [16]. More specifically, we will consider the two basic members of this family, i.e., *DL-Lite*$_{\mathcal{R}}$ and *DL-Lite*$_{\mathcal{F}}$.

In *DL-Lite*$_{\mathcal{R}}$, the TBox is a finite set of assertions having the following forms:

$$\begin{array}{ll} B_1 \sqsubseteq B_2 & Q_1 \sqsubseteq Q_2 \quad (concept/role\ inclusion) \\ B_1 \sqsubseteq \neg B_2 & Q_1 \sqsubseteq \neg Q_2 \quad (concept/role\ disjointness) \end{array}$$

where: each Q_i, with $i \in \{1, 2\}$, is basic role, i.e., a role name $R \in \Gamma$ (a.k.a. atomic role) or its inverse R^-; each B_i, with $i \in \{1, 2\}$, is a concept name $A \in \Gamma$ (a.k.a. atomic concept), or a concept of the form $\exists R$, or $\exists R^-$, i.e., unqualified existential restrictions, which denote the set of objects occurring as first argument (a.k.a., domain) or second argument (a.k.a. range) of R, respectively. For all details on the semantics of all above concept and role expressions we refer the reader to [15]. We however recall that an interpretation $\mathcal{I}$ satisfies an inclusion $B_1 \sqsubseteq B_2$ if $B_1^{\mathcal{I}} \subseteq B_2^{\mathcal{I}}$, and satisfies $B_1 \sqsubseteq \neg B_2$ if $B_1^{\mathcal{I}} \cap B_2^{\mathcal{I}} = \emptyset$, and analogously for assertions on roles.

In *DL-Lite*$_{\mathcal{F}}$, inclusions and disjointnesses between roles are not allowed but it is possible to specify functionalities, which are assertions of the form:

$$(\mathsf{funct}\ Q) \quad (role\ functionalities)$$

where Q is a basic role. An interpretation $\mathcal{I}$ satisfies $(\mathsf{funct}\ Q)$ if there are no $o_1, o_2, o_3 \in \Delta^{\mathcal{I}}$ such that both (o_1, o_2) and (o_1, o_3) belong to $Q^{\mathcal{I}}$.

In both the above logics, the ABox is a finite set of membership assertions of the form $C(a)$ or $R(a, b)$, where C and R are an atomic concept and an atomic role, respectively, and a and b are individual names (a.k.a., constants). We finally note that *DL-Lite* logics adopt the Unique Name Assumption, that is, in every interpretation different constants are interpreted with different objects.

Example 1. Consider the atomic concepts $Professor$ and $Course$, the atomic roles $teaches$ and $expert\_in$, and the following *DL-Lite*$_{\mathcal{R}}$ TBox:

$$\begin{array}{ll} \theta_1: \ Course \sqsubseteq \neg Person & \theta_2: \ Professor \sqsubseteq Person \\ \theta_3: \ teaches \sqsubseteq expert\_in & \theta_4: \ \exists teaches^- \sqsubseteq Course \end{array}$$

Such a TBox states that a course is not a person (θ_1), every professor is a person (θ_2), whoever teaches a course is an expert about it (θ_3), and that everything that is taught (i.e., occurs in the range of $teaches$) is a course (θ_4).

Instead, we obtain a TBox in *DL-Lite*$_{\mathcal{F}}$ if we substitute θ_3 with the assertion

$$(\mathsf{funct}\ teaches^-)$$

specifying that each course can be taught by at most one professor.

The following assertions are an example of ABox (for both *DL-Lite*$_{\mathcal{R}}$ and *DL-Lite*$_{\mathcal{F}}$).

$$\begin{array}{ll} \alpha_1: & Professor(Einstein) \\ \alpha_2: & teaches(Einstein, Physics) \end{array}$$

Such an ABox states that (the individual denoted by) $Einstein$ is a Professor (α_1) and that $Einstein$ teaches Physics (α_2). □

Query answering. One of the most important reasoning service in the presence of ontologies coupled with data, and which is the service we are mainly interested in for this paper, is *query answering*. We start with a general notion of queries in FOL, and then we move to the definition of queries over DL ontologies. A *query* is a function-free FOL open formula, which we denote as:

$$\{\vec{x} \mid \exists \vec{y}.\phi(\vec{x}, \vec{y})\} \tag{1}$$

where $\exists \vec{y}.\phi(\vec{x}, \vec{y})$ called the body of the query is a FOL formula with free variables $\vec{x}$, also called the target list of the query, 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., queries in which $\exists \vec{y}.\phi(\vec{x}, \vec{y})$ is a conjunction of the form $\exists \vec{y}.p_1(\vec{x}_1, \vec{y}_1) \wedge \ldots \wedge p_n(\vec{x}_n, \vec{y}_n)$, where each $p_i(\vec{x}_i, \vec{y}_i)$ 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 atom predicate p_i is either an atomic concept or an atomic role from the ontology signature. A *union of conjunctive query* (UCQ) is a FOL query of the form:

$$\{\vec{x} \mid \exists \vec{y}_1.\phi_1(\vec{x}, \vec{y}_1) \vee \cdots \vee \exists \vec{y}_n.\phi_n(\vec{x}, \vec{y}_n)\} \tag{2}$$

such that each $\{\vec{x} \mid \exists \vec{y}_i.\phi_i(\vec{x}, \vec{y}_i)\}$ is a CQ. To simplify notation, throughout the paper we can write a FOL query $\{\vec{x} \mid \exists \vec{y}.\phi(\vec{x}, \vec{y})\}$ as the formula $\exists \vec{y}.\phi(\vec{x}, \vec{y})$, and a UCQ as a set 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 arity n of the form (1) over an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, an n-tuple $\vec{c}$ of constants is a *certain answer* to q in $\mathcal{O}$ if $\mathcal{O} \models \exists \vec{y}.\phi(\vec{c}, \vec{y})$, i.e., the sentence obtained by substituting in q each variable in $\vec{x}$ with the corresponding constant in $\vec{c}$ is entailed by $\mathcal{O}$. In the following, we can write $q(\vec{x})$ to denote a query of the form (1) with free variables $\vec{x}$, and $q(\vec{c})$ to denote $\exists \vec{y}.\phi(\vec{c}, \vec{y})$. The set of certain answers to q in $\mathcal{O}$ is denoted by $cert(q, \mathcal{O})$.

We recall that establishing whether an ontology is *satisfiable*, i.e., whether it admits at least one model, can be reduced to query answering over a satisfiable ontology, for both

DL-Lite$_\mathcal{R}$ and *DL-Lite*$_\mathcal{F}$, as shown in [15, 28]. Also, query answering over an unsatisfiable ontology is meaningless, since computing the certain answers to a query amounts to get all tuples of constants having the same arity of the query. For these reasons, in this work we will consider only query answering over satisfiable ontologies.

Computing the certain answers to a query q with respect to a satisfiable *DL-Lite*$_\mathcal{R}$ or *DL-Lite*$_\mathcal{F}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ can be solved by first computing the so called perfect reformulation q_r of q with respect to the TBox $\mathcal{T}$, and then evaluating q_r over the ABox $\mathcal{A}$ (that is, evaluating it over an interpretation that is isomorphic to $\mathcal{A}$, which intuitively corresponds to consider $\mathcal{A}$ a relational database instance). Formally, given a query q and a TBox $\mathcal{T}$, a *perfect reformulation* of q with respect to $\mathcal{T}$ is a query q_r such that, for every ABox $\mathcal{A}$, $cert(q, \langle \mathcal{T}\, \mathcal{A} \rangle) = cert(q_r, \mathcal{A})$ (notice that $cert(q_r, \mathcal{A})$ indeed corresponds to evaluate q_r over $\mathcal{A}$ seen as a database).

Calvanese et al. proposed in [15] a prototypical algorithm, called PerfectRef, for computing the perfect reformulation of a UCQ Q with respect to a *DL-Lite*$_\mathcal{R}$ or *DL-Lite*$_\mathcal{F}$ TBox. At the basis of the algorithm there is a property saying that to compute the certain answers via rewriting over satisfiable *DL-Lite* ontologies, only concept/role inclusions (also called positive inclusions) need to be used in the reformulation process. According to PerfectRef, such inclusions are used as rewriting rules, from right to left, to repeatedly rewrite atoms in the queries in Q (seen as a set of CQs). 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 UCQ. For example, given a TBox assertion $B \sqsubseteq A$, 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 [15]. Moreover, when atoms in the query unify, PerfectRef performs such unification, which may then trigger some further atom rewriting[b].

For more details on PerfectRef we refer the reader to [15]. Below, we simply provide an example to intuitively show how it works.

Example 2. Consider the following query q expressed over the ontology $\mathcal{O}$ of Example 1:

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

asking for persons who teaches a course.

The certain answers to q in $\mathcal{O}$ are given by the evaluation of the UCQ Q produced by the algorithm PerfectRef over $\mathcal{A}$. Q is a set consisting of the following CQs:

$$\begin{aligned} q:\ & \{x \mid \exists y.Person(x) \wedge teaches(x, y) \wedge Course(y)\} \\ q_1: & \{x \mid \exists y.Professor(x) \wedge teaches(x, y) \wedge Course(y)\} \\ q_2: & \{x \mid \exists y.Person(x) \wedge teaches(x, y)\} \\ q_3: & \{x \mid \exists y.Professor(x) \wedge teaches(x, y)\}. \end{aligned}$$

[b] We note that, as a consequence of unification operations, the target list of a query in the set of CQs returned by PerfectRef may also contain constants, and that the taget lists of the CQs in the returned set may also not be equal to one another.

P	r	o	f	e	s	s	o	r	_	E	i	n	s	t	e	i	n	_	t	a	u	g	h	t	_	p	h	y	s	i	c	s	.
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
T	h	e	_	P	r	o	f	e	s	s	o	r	_	w	o	n	_	a	_	n	o	b	e	l	.								
35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60								

Figure 1. Document $\mathbf{D}^{\mathbf{ex}}$

The query q_1 is obtained from q by rewriting $Person(x)$ into $Professor(x)$, according to inclusion θ_2. q_2 is obtained from q after rewriting $Course(y)$ into $\exists z.teaches(z, y)$ (according to inclusion θ_4) and after unifying $teaches(x, y)$ and $teaches(z, y)$. Similarly for q_3, which is derived from q_1. Since $\mathcal{O}$ is satisfiable, we can obtain the certain answers to q in $\mathcal{O}$ by evaluating Q over the ABox $\mathcal{A}$, which returns the set $\{Einstein\}$. □

Interestingly, the perfect rewriting returned by PerfectRef is a UCQ, thus showing that query answering in *DL-Lite* logics is tractable in data complexity (the complexity computed with respect to the size of the ABox only), more precisely, in AC^0. At the same time, the evaluation of such UCQ can be delegated to a relational DBMS in charge of managing the data in the ABox, thus making *DL-Lite* logics particularly suited for ontology-based data management.

2.2. *Document spanners*

We now recall the definitions of spans and spanners, discuss a way of representing spanners, and present an algebra, through which the spanners of interest in this work are defined. Our presentation is necessarily concise. For more details we refer the reader to [10].

Strings and spans. We fix a finite alphabet Σ of symbols, which we assume totally ordered. In the following examples Σ is composed by the lower and capital letters of English alphabet, the full stop (“.”), and the underscore (“_”), which stands for the space character. We denote by Σ^* the set of all finite strings, called also *documents* over Σ. Thus, a document $\mathbf{D} \in \Sigma^*$ is such that $\mathbf{D} = \sigma_1 \dots \sigma_n$, with $n \geq 0$ and $\sigma_i \in \Sigma$ for $i \in \{1, \dots, n\}$.

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

We denote by $\mathsf{Spans}(\mathbf{D})$ the set of all possible spans of $\mathbf{D}$.

Example 3. Consider the document $\mathbf{D}^{\mathbf{ex}}$ given in Figure 1, and the span $[11, 19\rangle$. It identifies the substring Einstein, i.e., $\mathbf{D}^{\mathbf{ex}}_{[11,19\rangle} = \mathsf{Einstein}$. □

We assume to have a fixed and infinite set SVars of *variables*, disjoint from Σ^*. Given a finite set $V \subseteq \mathsf{SVars}$ and a document $\mathbf{D} \in \Sigma^*$, a $(V, \mathbf{D})$-*tuple* is a mapping $\mu : V \to \mathsf{Spans}(\mathbf{D})$ that assigns a span of $\mathbf{D}$ to each variable in V. When V is clear from the context, we simply call the above tuple a $(\mathbf{D})$-*tuple*. A $(V, \mathbf{D})$-*relation* is a set of $(V, \mathbf{D})$-*tuples*.

A *document spanner* (or simply *spanner*) is a function P over V that maps a document $\mathbf{D}$ to a $(V, \mathbf{D})$-*relation*. We use $\mathsf{SVars}(P)$ to denote the set of variables of a spanner P.

$\mathsf{eval}(\llbracket\gamma_{tok}\rrbracket, \mathbf{D^{ex}})$	
	x
μ_1	$[1, 10\rangle$
μ_2	$[11, 19\rangle$
μ_3	$[20, 26\rangle$
μ_4	$[27, 34\rangle$
μ_5	$[35, 38\rangle$
μ_6	$[39, 48\rangle$
μ_7	$[49, 52\rangle$
μ_8	$[53, 54\rangle$
μ_9	$[55, 60\rangle$

Figure 2. Spanner $\llbracket\gamma_{tok}\rrbracket$ applied to the document in Figure 1

The cardinality of $\mathsf{SVars}(P)$ is the *arity* of 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 $\mathbf{D}$, we write $\mathsf{eval}(P, \mathbf{D})$ to denote the $(V, \mathbf{D})$-*relation* returned by P with $\mathbf{D}$ as input.

Example 4. In Figure 2 we provide an example of $(V, \mathbf{D})$-*relation*, for the spanner $\llbracket\gamma_{tok}\rrbracket$, such that $\mathsf{SVars}(\llbracket\gamma_{tok}\rrbracket) = \{x\}$. $(V, \mathbf{D})$-tuples in this figure correspond to the words of $\mathbf{D^{ex}}$ from Figure 1 (we discuss below how to represent such spanner in formulas). □

Spanner representation. Among the possible ways of representing spanners [10], in this paper we use so-called regex formulas. A *variable regex* is an extension of a regular expression with *capture variables*. Its grammar is defined as follows:

$$\gamma := \emptyset \mid \epsilon \mid \sigma \mid (\gamma \vee \gamma) \mid (\gamma \cdot \gamma) \mid \gamma^* \mid x\{\gamma\} \tag{3}$$

The symbol $\emptyset$ defines the empty set, ϵ is the empty string, and $\sigma \in \Sigma$. The $\vee$, $\cdot$, and * symbols denote disjunction, concatenation, and the Kleene-star operators, respectively. $x\{\gamma\}$ instead indicates that the match obtained through the variable regex γ is mapped (in the form of a span) to the variable $x \in \mathsf{SVars}$. Parenthesis may be used in a variable regex in the usual way to specify precedence between operators.

We denote by $\mathsf{SVars}(\gamma)$ the set of variables that occur in γ. We use γ^+ as abbreviations $\gamma \cdot \gamma^*$, and $[\sigma_i\text{-}\sigma_j]$ as a shortcut for the disjunction of all characters $\sigma \in \Sigma$ such that $\sigma_i \leq \sigma \leq \sigma_j$.

In this paper we consider only variable regex expressions that are *functional*, i.e., such that in a matching over a document each variable is associated with one span. A functional variable regex is also called *regex formula*. The class of regex formulas is denoted by RGX.

Example 5. l Consider the following (simplified) regex formulas system:

- $\gamma_{tok} = (\epsilon \vee (\Sigma^* \cdot (. \vee \_))) \cdot x_1\{[a\text{-}zA\text{-}Z]^+\} \cdot ((. \vee \_) \cdot \Sigma^*)$,
  i.e., a regex formula assigning to x_1 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);

- $\gamma_{cap} = (\epsilon \vee (\Sigma^* \cdot (. \vee \_)) \cdot x_1\{[A\text{-}Z] \cdot \Sigma^*\} \cdot ((. \vee \_) \cdot \Sigma^*)$,

  $$\gamma_{cap} = (\epsilon \vee (\Sigma^* \cdot (. \vee \_))) \cdot x_1\{[A\text{-}Z] \cdot \Sigma^*\} \cdot ((. \vee \_) \cdot \Sigma^*)$$

  i.e., a regex formula assigning to x_1 the words in a document that begin with a capital letter;
- $\gamma_{aft\_prof} = (\Sigma^* \cdot \_) \cdot (\mathsf{Professor} \cdot \_) \cdot x_1\{\Sigma^+\} \cdot (\_ \cdot \Sigma^*)$,
  i.e., a regex formula assigning to x_1 the words in a document that follow the word Professor (plus a space). □

A regex formula γ naturally represents a spanner, and by $[\![\gamma]\!]$ we denote the spanner that is represented by γ. Then, with $[\![\mathrm{RGX}]\!]$ we denote the class of all spanners represented by regex formulas.

An algebra over spanners. We now present an algebra over spanners. This algebra extends the class of spanners that are represented by regex formulas, i.e., $[\![\mathrm{RGX}]\!]$, with the following operators: union ($\cup$), projection (π), (natural) join ($\bowtie$), and string-equality selection ($\zeta^=$). The set of spanners represented by formulas in the class RGX closed under $\cup, \pi, \bowtie$ and $\zeta^=$ is denoted by $[\![\mathrm{RGX}^{\{\cup,\pi,\bowtie,\zeta^=\}}]\!]$. Formally, let P, P_1 and P_2 be spanners and let $\mathbf{D}$ be a document, the above operators are defined as follows [10]:

- **Union.** The union $P_1 \cup P_2$ is defined when P_1 and P_2 are *union compatible*, that is, $\mathsf{SVars}(P_1) = \mathsf{SVars}(P_2)$. In that case, $\mathsf{SVars}(P_1 \cup P_2) = \mathsf{SVars}(P_1)$ and $\mathsf{eval}(P_1 \cup P_2, \mathbf{D}) = \mathsf{eval}(P_1, \mathbf{D}) \cup \mathsf{eval}(P_2, \mathbf{D})$.
- **Projection.** If $\mathbf{v} \subseteq \mathsf{SVars}$, then $\pi_{\mathbf{v}}(P)$ is the spanner such that $\mathsf{SVars}(\pi_{\mathbf{v}}(P)) = \mathbf{v}$ and $\mathsf{eval}(\pi_{\mathbf{v}}(P), \mathbf{D})$ is obtained from $\mathsf{eval}(P, \mathbf{D})$ by restricting the domain of each $(\mathbf{D})$-*tuple* to $\mathbf{v}$.
- **(Natural) Join.** The join between spanners is defined as $P_1 \bowtie P_2$. It holds that $\mathsf{SVars}(P_1 \bowtie P_2) = \mathsf{SVars}(P_1) \cup \mathsf{SVars}(P_2)$, and $\mathsf{eval}(P_1 \bowtie P_2, \mathbf{D})$ consists of all $(\mathbf{D})$-*tuple*s μ that agree with some $\mu_1 \in \mathsf{eval}(P_1, \mathbf{D})$ and $\mu_2 \in \mathsf{eval}(P_2, \mathbf{D})$.
- **String selection.** Let x and y be two variables in $\mathsf{SVars}(P)$, the string-equality selection operator is defined as $\zeta^=_{x,y}P$. We have that $\mathsf{SVars}(\zeta^=_{x,y}P) = \mathsf{SVars}(P)$, and $\mathsf{eval}(\zeta^=_{x,y}P, \mathbf{D})$ consists of all $(\mathbf{D})$-*tuple*s μ in $\mathsf{eval}(P, \mathbf{D})$ such that $\mathbf{D}_{\mu(x)} = \mathbf{D}_{\mu(y)}$.

Example 6. Using the regex formula defined in Example 5 we can define, using the spanner algebra, the following more expressive and complex $[\![\mathrm{RGX}^{\{\cup,\pi,\bowtie,\zeta^=\}}]\!]$-spanner.

- $[\![\rho_{prof}]\!] = [\![\gamma_{cap}]\!] \bowtie [\![\gamma_{aft\_prof}]\!]$, 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 $[\![\rho_{prof}]\!]$ to the document $\mathbf{D}^{\mathbf{ex}}$ in Figure 1 is shown in Figure 3. The extracted span is $[11, 19\rangle$ corresponding to the substring Einstein. □

$\mathsf{eval}([\![\rho_{prof}]\!], \mathbf{D^{ex}})$	
	x_1
μ_1	$[11, 9\rangle$

Figure 3. Result of spanner $[\![\rho_{prof}]\!]$ applied to the document in Figure 1

In our framework, which we introduce in the next section, we will consider only spanners belonging to $[\![\mathrm{RGX}^{\{\cup,\pi,\bowtie,\zeta^=\}}]\!]$. It is worthwhile to remind the reader that spanners in such class are called *core spanners*, for being them able to capture the core of AQL, the declarative language used in SystemT, the IBM rule-based IE tool [8]. Therefore, an actual implementation of our framework can be straightforwardly based on the usage of a state-of-the-art tool like SystemT for declarative IE from text documents.

3. Linking text documents to ontologies

In this section we present our framework for coupling documents to ontologies, which we call *Ontology-based document spanning* (OBDS) framework. In the last part of the section, we also describe the problem of query processing in OBDS, which we will then study in depth in the last part of the paper.

Before delving into the details of the framework, we discuss how to deal with the following main problem: when mapping text documents to ontologies, it is likely that the text does not directly contain the identifiers that are used at the ontology level to denote the objects that are instances of the predicates of the ontology. Rather, the strings that are extracted from the document should more correctly interpreted as values.Our basic idea to deal with this problem is to devise a linking mechanism that is inspired by the mapping used in OBDA, and adopt the same technique adopted in OBDA to construct objects from values: consider object identifiers formed by (logic) terms built out from the string values extracted from the documents [27]. To formally describe this mechanism we recall the notions of object term and variable term. An object term has the form $\mathbf{f}(\vec{d})$ where $\vec{d}$ is an m-tuple of either constants or variables and $\mathbf{f}$ is a function symbols of arity m. If $\vec{t}$ is a tuple of variables without constants, $\mathbf{f}(\vec{d})$ is called variable object term. If instead $\vec{d}$ is a tuple of constants, $\mathbf{f}(\vec{d})$ is called ground object term.

We now turn to the framework definition. The three ingredients for an OBDS system are the ontology, a set of extraction assertions linking text data to the ontology, and a source text document.

Definition 1. 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

$$P(\vec{x}) \rightsquigarrow \Psi(\vec{x}) \tag{4}$$

  where
  - $P(\vec{x})$ (the left-hand side of the assertion) is a $[\![\mathrm{RGX}^{\{\cup,\pi,\bowtie,\zeta^=\}}]\!]$-spanner.

– $\Psi(\vec{x})$ (the right-hand side of the assertion) is a CQ over $\mathcal{T}$ with free variables $\vec{x}$, possibly using variable terms $\mathbf{f}(\vec{w})$, such that $\vec{w} \subseteq \vec{x}$, as arguments of its atoms. Note that $\Psi(\vec{x})$ may contain also existentially quantified variables.

In the following, when the TBox of an OBDS system $\mathcal{E}$ is specified in a DL language $\mathcal{L}$ we say that $\mathcal{E}$ is an $\mathcal{L}$ OBDS system.

Example 7. Let $\mathcal{E} = \langle \mathcal{T}, \mathcal{R} \rangle$ be an OBDS system where $\mathcal{T}$ is as in Example 1, and let $[\![\rho_{prof}]\!]$ and $[\![\gamma_{teaches}]\!]$ be two spanners, where $[\![\rho_{prof}]\!]$ is as defined in Example 6, whereas the regex formula representing $[\![\gamma_{teaches}]\!]$ is:

$$\gamma_{teaches} = (\epsilon \vee (\Sigma^* \cdot \_)) \cdot x_2\{\Sigma^+\} \cdot (\_ \cdot \mathsf{taught} \cdot \_) \cdot y_2\{\Sigma^+\} \cdot ((. \vee \_) \cdot \Sigma^*)$$

i.e., a regex assigning to x_2 the words before the word taught, and to y_2 the words after taught.

The set of extraction assertions $\mathcal{R}$ is as follows:

$$\begin{array}{ll} m_1 : [\![\rho_{prof}]\!](x_1) & \rightsquigarrow Professor(\mathbf{prof}(x_1)) \\ m_2 : [\![\gamma_{teaches}]\!](x_2, y_2) & \rightsquigarrow teaches(\mathbf{prof}(x_2), \mathbf{course}(y_2)) \end{array}$$

Notice that both **prof** and **course** are function symbols of arity 1 used to construct individuals from the string returned by the spanners. □

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

- $\mathcal{I}$ is a model for $\mathcal{T}$, and
- $\Psi(\mathbf{D}_{\mu(x_1)}, \ldots, \mathbf{D}_{\mu(x_n)})$ evaluates to true in $\mathcal{I}$ for each $\mu \in \mathsf{eval}(P, \mathbf{D})$.

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

In a similar way to what happens for mappings in the context of data integration [17] and OBDA, we can have two types of extraction assertions, i.e., GAV and the GLAV. GLAV assertions are exactly assertions of the kind we discussed so far. Instead, in a GAV *extraction assertion* there are no existentially quantified variables in its right-hand side. In this case, $\Psi(\vec{x})$ in assertions of type (4) is in the form $p_1(\vec{x}_1) \wedge \ldots \wedge p_k(\vec{x}_k)$, with $\cup_{i=1}^{k} \vec{x}_i = \vec{x}$. It is easy to see that the previous extraction assertion is equivalent to the set of assertions:

$$\begin{array}{c} P(\vec{x}_1) \rightsquigarrow p_1(\vec{x}_1) \\ \ldots \\ P(\vec{x}_k) \rightsquigarrow p_k(\vec{x}_k), \end{array}$$

that is, the right-hand side of each assertion is a single-atom query without existential variables (but still possibly containing variable terms). Therefore, from now on, we always assume that GAV *extraction assertions* have the form above (unless otherwise specified).

We conclude this section by talking about query answering in OBDS systems. Query answering in one such system refers to the task of computing the answer set to a query posed on the ontology. As stated earlier, we adopt the notion of certain answers for the

semantics of query answering. In a OBDS system, computing the *certain answers* to a query q with respect to a document $\mathbf{D}$, denoted by $cert(q, \mathcal{E}, \mathbf{D})$, amounts to finding the answers to q that hold in all models for $\mathcal{E}$ with respect to $\mathbf{D}$.

Definition 2. Let $\mathcal{E} = \langle \mathcal{T}, \mathcal{R} \rangle$ be an OBDS system, let q be a query, and let $\mathbf{D}$ be a document. A tuple of constants and ground object terms $\vec{t}$ is a *certain answer* to q in $\mathcal{E}$ with respect to $\mathbf{D}$ if for every model $\mathcal{I} \in Mod(\mathcal{E}, \mathbf{D})$ it holds that $(q(\vec{t}))^{\mathcal{I}}$ evaluates to true[c].

For example, the set of the certain answers to the query $\{x \mid Person(x)\}$ in the OBDS system $\mathcal{E}$ of Example 7 with respect to the document $\mathbf{D^{ex}}$ in Figure 1 is $\{\mathbf{prof}(\mathsf{Einstein})\}$.

4. Complexity of query answering in OBDS systems

To establish computational complexity of query answering in our framework we show how to reduce this problem to query answering in an OBDA system. We thus first recall some basic notions 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 a relational source schema, 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 (4)), 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 it is defined with respect to a database instance for $\mathcal{S}$. More precisely, given one such database DB (called source database), a model for $\mathcal{J}$ is any interpretation $\mathcal{I}$ 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, $(\Psi(c_1, \ldots, c_n))^{\mathcal{I}}$ evaluates to true. The notion of entailment in an OBDA system is analogous to the same notion for ontologies and OBDA systems. Also the notion of certain answers to a query q in an OBDA system $\mathcal{J}$ is as the one for OBDS systems (cf. Definition 2), but in this case it is given with respect to a source database $\mathbf{DB}$, denoted $cert(q, \mathcal{J}, \mathbf{DB})$.

Given these similarities between the two frameworks, we can easily reduce query answering in an OBDS system to query answering in an OBDA system. Intuitively, given an OBDS system $\mathcal{E} = \langle \mathcal{T}, \mathcal{R} \rangle$ and a document $\mathbf{D}$, we can construct an OBDA system $J = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$ and a source database DB for J such that:

- $\mathcal{T}$ is the same TBox of $\mathcal{E}$;
- $\mathcal{S}$ is a source schema which contains a relation schema T_P for each spanner P occurring in $\mathcal{R}$, such that the arity of T_P coincides with the number of variables in $\mathsf{SVars}(P)$ (in other terms, $\mathcal{S}$ is the schema "produced" by the spanners in $\mathcal{R}$);

[c] $(q(\vec{t}))^{\mathcal{I}}$ is the interpretation in $\mathcal{I}$ of the sentence $q(\vec{t})$, which possibly contains ground object terms. Each such term $\mathbf{f}(\vec{c})$ is interpreted exactly as a constant, i.e., $(\mathbf{f}(\vec{c}))^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ and no two different terms are interpreted with the same object in $\Delta^{\mathcal{I}}$ (i.e., we adopt the unique name assumption on terms, too).

Algorithm obds2obda
Input: A set of extraction assertions $\mathcal{R}$, a document $\mathbf{D}$
Output: A mapping $\mathcal{M}$, a relational schema $\mathcal{S}$ and a relational database DB
begin
$\mathcal{S} \leftarrow \emptyset$;
$\mathsf{DB} \leftarrow \emptyset$;
$\mathcal{M} \leftarrow \emptyset$;
for each $r \in \mathcal{R}$, where $r = P(\vec{x}) \rightsquigarrow \Psi(\vec{x})$, **do**
$\mathcal{S} \leftarrow \mathcal{S} \cup \{T_P \mid$ such that T_P is a fresh relation schema of the same arity of $P\}$;
$\mathsf{DB} \leftarrow \mathsf{DB} \cup \{T_P(\mathbf{D}_{\mu(x_1)}, \ldots, \mathbf{D}_{\mu(x_n)}) \mid \mu \in eval(P, \mathbf{D})\}$;
$\mathcal{M} \leftarrow \mathcal{M} \cup \{T_P(\vec{x}) \rightsquigarrow \Psi(\vec{x})\}$;
return $\mathcal{M}$, $\mathcal{S}$, and DB
end

Figure 4. The obds2obda$(\mathcal{R}, \mathbf{D})$ algorithm

- $\mathcal{M}$ is a mapping containing an assertion m for each extraction assertion e in $\mathcal{R}$, such that m and e have the same right-hand side, and, let P be spanner of e such that $|\mathsf{SVars}(P)|$ is n, the left-hand side of m is the query $T_P(x_1, \ldots, x_n)$ (in other terms, $\mathcal{M}$ contains the same assertions of $\mathcal{R}$, modulo a substitution of the spanners with the corresponding relation symbol in $\mathcal{S}$);
- DB is a source database instance for $\mathcal{S}$ obtained by evaluating each spanner in $\mathcal{R}$ over the document $\mathbf{D}$, which returns tuples of spans, and by extracting the substrings of $\mathbf{D}$ identified by such spans.

In Figure 4, we give an algorithm, called obds2obda, that taken as input a set of extraction assertions $\mathcal{R}$ returns $\mathcal{M}$, $\mathcal{S}$, and DB as described above.

The following lemma shows the semantic relation between an OBDS system and the corresponding OBDA system constructed with the algorithm obds2obda.

Lemma 1. *Let $\mathcal{E} = \langle \mathcal{T}, \mathcal{R} \rangle$ be Given an OBDS system and $\mathbf{D}$ be a document. Let $\mathcal{M}$, $\mathcal{S}$ and DB be respectively the mapping, the relational schema, and the database returned by obds2obda$(\mathcal{R}, \mathbf{D})$, and let $\mathcal{J} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$ be an OBDA system. Then, $Mod(\mathcal{E}, \mathbf{D}) = Mod(\mathcal{J}, \mathsf{DB})$.*

Proof. Let us assume that there exists $\mathcal{I} \in Mod(\mathcal{E}, \mathbf{D})$ such that $\mathcal{I} \notin Mod(\mathcal{J}, \mathsf{DB})$. Since $\mathcal{I}$ is a model for $\mathcal{E}$ with respect to $\mathbf{D}$, then $\mathcal{I}$ satisfies $\mathcal{T}$. Thus, if $\mathcal{I}$ is not a model of $\mathcal{J}$ with respect to DB, $\mathcal{I}$ does not satisfy $\mathcal{M}$. This means that there must be an assertion $\Phi(\vec{x}) \rightsquigarrow \Psi(\vec{x})$ belonging to $\mathcal{M}$ such that there exists a tuple of constants $\vec{c}$ in the evaluation of $\Phi(\vec{x})$ over DB for which $(\Psi(\vec{c}))^{\mathcal{I}}$ evaluates to false. However, by construction of DB, $\Phi(\vec{x}) = T_P(\vec{x})$, and every tuple $\vec{c}$ is such that $\vec{c} = (\mathbf{D}_{\mu(x_1)}, \ldots, \mathbf{D}_{\mu(x_n)})$ for some $\mu \in \mathsf{eval}(P, \mathbf{D})$, and since $\mathcal{I}$ satisfies $\mathcal{R}$ (by hypothesis), it holds that $(\Psi(\vec{c}))^{\mathcal{I}}$ evaluates to true in $\mathcal{I}$. This leads to a contradiction and thus shows that $Mod(\mathcal{E}, \mathbf{D}) \subseteq Mod(\mathcal{J}, \mathsf{DB})$. The

fact that $Mod(\mathcal{J}, \mathsf{DB}) \subseteq Mod(\mathcal{E}, \mathbf{D})$ can be proved in an analogous way, thus finally showing the thesis. □

The theorem below follows from Lemma 1 and the fact that computing the certain answers to a CQ over an OBDA system whose TBox is specified in either *DL-Lite*$_\mathcal{R}$ or *DL-Lite*$_\mathcal{F}$ is in AC_0 in data complexity [15].

Theorem 1. *Let $\mathcal{E} = \langle \mathcal{T}, \mathcal{R} \rangle$ be either a DL-Lite$_\mathcal{R}$ or DL-Lite$_\mathcal{F}$ OBDS system, $\mathcal{R}$ be a set of GLAV extraction assertions, $\mathbf{D}$ be a document, and q be a CQ over $\mathcal{E}$. Then computing $cert(q, \mathcal{E}, \mathbf{D})$ can be solved in time polynomial in the size of $\mathbf{D}$.*

Proof. From Lemma 1 it follows that $cert(q, \mathcal{E}, \mathbf{D}) = cert(q, \mathcal{J}, \mathsf{DB})$, where $\mathcal{J} = \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$, and $\mathcal{M}$, $\mathcal{S}$ and DB are returned by obds2obda$(\mathcal{R}, \mathbf{D})$. Thus the data complexity of computing $cert(q, \mathcal{E}, \mathbf{D})$ is equal to the execution cost of obds2obda, with respect to the input document $\mathbf{D}$, and the cost of computing $cert(q, \mathcal{J}, \mathsf{DB})$. It is also easy to verify that obds2obda runs in polynomial time in the size of $\mathbf{D}$. Indeed, the only steps of obds2obda that depend on $\mathbf{D}$ concern with the construction of $\mathbf{DB}$, which is obtained by the evaluation of all the spanners in $\mathcal{R}$ over $\mathbf{D}$, and the subsequent extraction of the substrings of $\mathbf{D}$ identified by the spans returned by such evaluations, which clearly are tasks polynomial in $\mathbf{D}$ (see also [10]). As for the cost of computing $cert(q, \mathcal{J}, \mathsf{DB})$, we recall that conjunctive query answering in OBDA systems having either *DL-Lite*$_\mathcal{R}$ or *DL-Lite*$_\mathcal{F}$ TBoxes and GAV mappings is in AC^0 in data complexity [27]. This result extends also to GLAV mappings, for *DL-Lite*$_\mathcal{R}$ TBoxes, as shown in [29]. When mappings are GLAV and TBoxes are in *DL-Lite*$_\mathcal{F}$ the complexity rises to PTIME, which follows from the results in [30] and [15]. Thus, in all cases the problem can be solved in time polynomial in the size of $\mathbf{D}$. □

We notice that the above technique that reduces query answering over OBDS systems to query answering over OBDA systems is obviously general and can be used also when the TBox is specified in other DL languages. In all cases, however, we need to pay the cost of constructing a source database for the OBDA system $\mathcal{J}$ by evaluating the spanners in $\mathcal{R}$ over the document $\mathbf{D}$ (which is polynomial).

From practical perspective, however, the approach of "materializing" the result of spanner evaluation may have some drawbacks. Indeed, the source document is independent from the ontology, and thus it may happen that, during the lifetime of an OBDS system, its content is modified (in other terms, the system can be coupled with a new document, still using the same extraction assertions). This would clearly require to set up a mechanism for keeping the database created via spanner execution up-to-date with respect to the document "evolution". Furthermore, this is not in the spirit of virtual data integration, which is typically performed through OBDA systems. To overcome such problems, in the next section we propose a different approach to query answering, which we base on query rewriting.

5. Query Answering via Query Rewriting in *DL-Lite*

In this section we study query rewriting over OBDS systems, i.e. how to answer a CQ q posed over one such system $\mathcal{E}$ by transforming q into a spanner whose evaluation over an underlying document $\mathbf{D}$ returns the certain answers to q in $\mathcal{E}$ with respect to $\mathbf{D}$.

We start by considering OBDS systems equipped with GAV extraction assertions, and show that, in this case, CQ answering in both *DL-Lite*$_\mathcal{R}$ and *DL-Lite*$_\mathcal{F}$ OBDS systems is reducible to the evaluation of a $[\![\mathrm{RGX}^{\{\cup,\pi,\bowtie,\zeta^=\}}]\!]$-spanner over a document $\mathbf{D}$, i.e. a spanner of the same expressiveness of those allowed in the extraction assertions.

Then we tackle the general case of GLAV extraction assertions. For this setting, we show that the above result still holds for *DL-Lite*$_\mathcal{R}$ OBDS systems, and actually, we can use the same technique of the GAV case, modulo an easy transformation of the extraction assertions. We also show that, instead, this technique does not work for *DL-Lite*$_\mathcal{F}$ OBDS systems. For this case, we envisage that input queries should be rewritten in an algebra over regex formulas allowing for recursion, i.e. that allows for expressing spanners that go beyond $[\![\mathrm{RGX}^{\{\cup,\pi,\bowtie,\zeta^=\}}]\!]$ (e.g. a spanner in RGXlog [31]).

5.1. *GAV Extraction Assertions*

Given a GAV *DL-Lite*$_\mathcal{R}$ or *DL-Lite*$_\mathcal{F}$ OBDS system $\mathcal{E} = \langle \mathcal{T}, \mathcal{R} \rangle$ and a query q over $\mathcal{E}$, we rewrite q in three steps, which we call *rewriting based on the ontology*, *rewriting based on the extraction assertions* and *reformulation into document spanners*. 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 $\mathcal{R}$. The result produced at this step is a set U of queries, having an "intermediate syntax" between CQs and spanners in $[\![\mathrm{RGX}^{\{\cup,\pi,\bowtie,\zeta^=\}}]\!]$. The final step transforms the queries in U into document spanners in the class $[\![\mathrm{RGX}^{\{\cup,\pi,\bowtie,\zeta^=\}}]\!]$.

Rewriting based on the ontology. For the first step we adopt the algorithm PerfectRef presented in [15], which we have briefly described in Sec. 2.

Rewriting based on the extraction assertions. The second step is through an *unfolding* method, similar to the one described in [27]. Roughly, the unfolding procedure substitutes, in all possible ways, each atom α in each query returned by PerfectRef with the spanners occurring in the left-hand side of extraction assertions referring to the predicate occurring in α. To this aim, we use the procedure Unfolding, which takes as input a UCQ Q and a set of extraction assertions $\mathcal{R}$. This procedure, for each CQ $q \in Q$, 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 assertion $P(\vec{v}_i) \rightsquigarrow p_i(\vec{v}_i)$, computes the most general unifier σ between $p_i(\vec{t}_i)$ and $p_i(\vec{v}_i)$, and, if such a σ exists, substitutes $p_i(\vec{t}_i)$ with $P(\vec{v}_i)$ and applies σ to the obtained formula. Note that only queries having all atoms that unify with at least one extraction assertion are completely unfolded and returned by Unfolding. After this step, the returned set U contains queries of the form $\{\vec{t} \mid \exists \vec{y}_1, \ldots, \vec{y}_n . P_1(\vec{t}_1, \vec{y}_1) \wedge \cdots \wedge P_n(\vec{t}_n, \vec{y}_n)\}$, where the

target list $\vec{t}$ may contain variables, constants, and object terms[d], each P_i is a spanner, each $\vec{y_i}$ is a (possibly empty) sequence of variables, and each $\vec{t_i}$ is a (possibly empty) sequence of variables occurring also in $\vec{t}$. An example of unfolding is given in Example 8.

Reformulation into document spanners. The last step is carried out by the Transform algorithm. Roughly speaking, such an algorithm transforms the body of each query $f \in U$ into a document spanner in $[\![\mathrm{RGX}^{\{\cup,\pi,\bowtie,\zeta^=\}}]\!]$, and returns this spanner together with the target list of f, suitably modified on the basis of certain variable substitutions needed for the transformation. In particular, Transform converts each join between values (expressed by multiple occurrences of a variable in the body of f) and each selection (specified through the occurrence of a constant in the body of f) into a Cartesian product between spanners (i.e. a $\bowtie$ between spanners with no common variables), to which a string selection (i.e. $\zeta^=$) is applied. More precisely, Transform operates in three steps. First of all, it substitutes each constant c occurring in f with a fresh (existentially quantified) variable, say w, and adds to the conjunction in f the atom $P_c(w)$, where $P_c = [\![\Sigma^* \cdot w\{c\} \cdot \Sigma^*]\!]$, i.e. P_c is the spanner represented by a regex formula that assigns to the variable w only the spans matching with the constant c. For example, given the query $\hat{f} = \{y \mid \exists x.P_1(x,y) \wedge P_2(y,c)\}$, Transform, in its first step, reformulates $\hat{f}$ into $\hat{f}' = \{y \mid \exists x,w.P_1(x,y) \wedge P_2(y,w) \wedge P_c(w)\}$. In the second step, for each variable z that appears more than once in the query body, Transform substitutes each occurrence of z with a fresh variable, and adds to the query body a conjunction of equalities specifying that all such fresh variables are equal to one another. If z occurs in the target list (as free variable or as argument of object terms), it is substituted with any of the newly introduced variables. In our ongoing example, $\hat{f}'$ is reformulated into $\hat{f}'' = \{y_1 \mid \exists x,y_2,w_1,w_2.P_1(x,y_1) \wedge P_2(y_2,w_1) \wedge P_c(w_2) \wedge y_1 = y_2 \wedge w_1 = w_2\}$. In its third step, Transform iteratively applies the following rule, as long as it is applicable: let f'' be the query computed after the second step of Transform, let β be a conjunction of atoms of the from $\alpha_1 \wedge \alpha_2 \wedge x = y$ occurring in f'', such that x occurs in α_1 and y occurs in α_2, substituting β in f'' with $\zeta^=_{x,y}(\alpha_1 \bowtie \alpha_2)$. Finally, Transform adds a projection (i.e. π) to the query body in order to project out only the variables occurring in the target list of the query, eliminates the existential quantification to obtain a syntactically correct span representation, and returns both the target list and the computed spanner. In our example, the body of $\hat{f}''$ is thus finally transformed into $\pi_{y_1}(\zeta^=_{y_1,y_2}(P_1(x,y_1) \bowtie (\zeta^=_{w_1,w_2}(P_2(y_2,w_1) \bowtie P_c(w_2)))))$, whereas the target list returned by Transform is simply constituted by the variable y[e].

The rewriting algorithm for the GAV case, which put together the three functions we have just described is given below. A complete example of the entire rewriting process is given in Example 8 (see Eq. 6).

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

[d] With a little abuse of notation we continue to call $\vec{t}$ target list, even though it does not contain only variables (as defined in Sec. 2).

[e] In this simple example, Transform could even not explicitly return the target list, but in general the target list conveys information crucial to construct object terms that may occur in the certain answers (see Example 8).

Input: OBDS $\mathcal{E} = \langle \mathcal{T}, \mathcal{R} \rangle$, such that $\mathcal{T}$ is either a *DL-Lite*$_\mathcal{R}$ or a *DL-Lite*$_\mathcal{F}$ TBox and $\mathcal{R}$ is a set of GAV extraction assertions,
CQ q
Output: Sequence of terms T (i.e. a target list),
Document spanner $P \in [\![\mathrm{RGX}^{\{\cup,\pi,\bowtie,\zeta^=\}}]\!]$
begin
$Q = \mathsf{PerfectRef}(\mathcal{T}, q)$
$U = \mathsf{Unfolding}(Q, \mathcal{R})$
$(T, P) = \mathsf{Transform}(U)$
return (T, P)
end

Example 8. Consider the setting of Example 7, and the following query q that asks for the persons who teach a course:

$$q = \{x \mid \exists y.\ Person(x) \wedge teaches(x, y) \wedge Course(y)\} \tag{5}$$

as shown in Example 2, the result of $\mathsf{PerfectRef}(\mathcal{T}, q)$ is the set Q containing the following CQs:

$$\begin{aligned} q:\ & \{x \mid \exists y. Person(x) \wedge teaches(x, y) \wedge Course(y)\} \\ q_1: & \{x \mid \exists y. Professor(x) \wedge teaches(x, y) \wedge Course(y)\} \\ q_2: & \{x \mid \exists y. Person(x) \wedge teaches(x, y)\} \\ q_3: & \{x \mid \exists y. Professor(x) \wedge teaches(x, y)\}. \end{aligned}$$

After the execution of PerfectRef, $\mathsf{Unfolding}(Q, \mathcal{R})$ unfolds the queries in Q by using the extraction assertions in $\mathcal{R}$. In our example only q_3 can be completely unfolded. For q_3, the atom $Professor(x)$ unifies with the atom $Professor(\mathbf{prof}(x_1))$ in the extraction assertion m_1 through the unifier $\sigma' = \{x \rightarrow \mathbf{prof}(x_1)\}$, and then the atom $\sigma'(teaches(x, y)) = teaches(\mathbf{prof}(x_1), y)$ unifies with the atom $teaches(\mathbf{prof}(x_2), \mathbf{course}(y_2))$ in the extraction assertion m_2 with the unifier $\sigma'' = \{x_1 \rightarrow x_2, y \rightarrow \mathbf{course}(y_2)\}$. The unfolding will thus produce the following query[f]:

$$\{\mathbf{prof}(x_2) \mid \exists y_2.([\![\rho_{prof}]\!](x_2) \wedge [\![\gamma_{teaches}]\!](x_2, y_2))\} \tag{6}$$

In the above query (6), we are slightly abusing the notation, since, after the unfolding, the variables denote spans, and not directly the strings we are looking for. Thus, when we write $\mathbf{prof}(x_2)$ we in fact mean $\mathbf{prof}(\mathbf{D}_{x_2})$, where $\mathbf{D}$ denotes the underlying text document. In other words, $\mathbf{prof}(x_2)$ indicates that the answer to the query consists of ground object terms with function symbol $\mathbf{prof}$ and as argument the strings identified by the spans returned through x_2, when the spanner represented by the regex formula in (the body of) the query (6) is evaluated.

Afterwards, the algorithm $\mathsf{Transform}(U)$ rewrites the above query in the spanner syntax in order to obtain a document spanner ready to be evaluated over the underlying document.

[f] Note that the application of the unifiers actually renames the variables used in the spanners.

$\mathsf{Transform}(U)$ first produces the following query, where no variable occurs more than once (see the description of the second step of Transform):

$$\{\mathbf{prof}(z_1) \mid \exists y_2, z_2.([\![\rho_{prof}]\!](z_1) \wedge [\![\gamma_{teaches}]\!](z_2, y_2) \wedge z_1 = z_2)\} \tag{7}$$

Then, it produces a representation of (the body of) the above query in the $[\![\mathrm{RGX}^{\{\cup,\pi,\bowtie,\zeta^=\}}]\!]$ syntax. More precisely, it computes the spanner, which we denote $[\![\rho_{transf}]\!]$, defined as follows:

$$[\![\rho_{transf}]\!] = \pi_{z_1}(\zeta^=_{z_1,z_2}([\![\rho_{prof}]\!] \bowtie [\![\gamma_{teaches}]\!])) \tag{8}$$

The above spanner is returned together with the target list $T = \mathbf{prof}(z_1)$. Note that, in Eq. (8), $[\![\rho_{prof}]\!]$ and $[\![\gamma_{teaches}]\!]$ are the spanners as defined in Examples 5 and 6, but in which the variables have been renamed by the functions Unfolding and Transform. More in detail, the original variable x_1 in $[\![\rho_{prof}]\!]$ is now z_1, and the original variable x_2 of $[\![\gamma_{teaches}]\!]$ is now z_2. □

We show in the following that the algorithm OBDS_Rewriting can be used to obtain the certain answers to a CQ q. It is indeed sufficient to evaluate over the underlying document the spanner returned by the algorithm, extract the strings corresponding to the spans produced by such an evaluation, and use them to bind the variables in the target list returned by OBDS_Rewriting. To formalize this last aspect, we need to introduce the function res. Given a document $\mathbf{D}$, a spanner P such that $\mathsf{SVars}(P) = V = v_1, \ldots, v_m$, a target list $T = t_1, \ldots, t_n$, such that set of variables occurring in T coincides with V, and given a $(V, \mathbf{D})$-*tuple* $\mu \in \mathsf{eval}(P, \mathbf{D})$, we define $\mathsf{res}(T, \mu)$ as the function that returns a tuple of constants and ground object terms $c_1, \ldots, c_n$ such that each c_i is obtained as follows:

- if t_i is a constant, $c_i = t_i$;
- if $t_i = v_j$, where $1 \leq j \leq m$, $c_i = \mathbf{D}_{\mu(v_j)}$;
- if $t_i = \mathbf{f_i}(v_{j_1}, \ldots, v_{j_k})$, where $1 \leq j_i \leq m$ for $i \in \{1, \ldots, k\}$, $c_i = \mathbf{f_i}(\mathbf{D}_{\mu(v_{j_1})}, \ldots, \mathbf{D}_{\mu(v_{j_k})})$.

We are now ready to provide the main result of this section.

Theorem 2. *Let $\mathcal{E} = \langle \mathcal{T}, \mathcal{R} \rangle$ be either a DL-Lite$_\mathcal{R}$ or DL-Lite$_\mathcal{F}$ OBDS system, such that $\mathcal{R}$ is a set of GAV extraction assertions, let $\mathbf{D}$ be a document, let q be a CQ over $\mathcal{E}$, and let T and P be the target list and spanner returned by* $\mathsf{OBDS\_Rewriting}(\mathcal{E}, q)$*, respectively. Then, $cert(q, \mathcal{E}, \mathbf{D}) = \bigcup_{\mu \in \mathsf{eval}(P,\mathbf{D})} \mathsf{res}(T, \mu)$. Furthermore, $P \in [\![\mathrm{RGX}^{\{\cup,\pi,\bowtie,\zeta^=\}}]\!]$.*

Proof. The result follows from the following facts: (i) $\mathsf{PerfectRef}(q, \mathcal{T})$ returns the perfect rewriting of a CQ q with respect to a *DL-Lite$_\mathcal{R}$* or *DL-Lite$_\mathcal{F}$* TBox $\mathcal{T}$, 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 [15]; (ii) the soundness of the procedure Unfolding to rewrite queries in GAV OBDA systems, as shown in [27], and (iii) the correctness of the algorithm Transform, which performs a purely syntactic/symbolic conversion. As for this last point, Transform simply converts CQs whose atoms use (symbols denoting) document spanners as predicates, into spanners represented in $[\![\mathrm{RGX}^{\{\cup,\pi,\bowtie,\zeta^=\}}]\!]$. It is not difficult to see that, for each

CQ q in U, the first and second step of $\mathsf{Transform}(U)$ produce a CQ query that is equivalent to q. Then the algorithm simply turns joins in the CQ (which are expressed through equalities between variables) into Cartesian products between spanners (i.e. natural joins between spanners with no common variables), which are in fact expressed over spans. The semantics of the joins between values is then obtained through the string-selection operator applied to the result of the natural joins between spans. As a final step, the algorithm simply re-expresses the projection specified in the query through the target list by using the projection operator π. It is then easy to see that T and P respect the pre-conditions of the function res, i.e. that the set of variables occurring in T coincides with $\mathsf{SVars}(P)$. Then, by construction we get that P belongs to the class $[\![\mathrm{RGX}^{\{\cup,\pi,\bowtie,\zeta^{=}\}}]\!]$. □

Example 9. In continuation of Example 8, we execute $\mathsf{eval}([\![\rho_{transf}]\!], \mathbf{D^{ex}})$, where $\mathbf{D^{ex}}$ is the document in Fig. 1, and we obtain the span $[11, 19\rangle$. Then, $cert(q, \mathcal{E}, \mathbf{D^{ex}}) = \{\mathbf{prof}(\mathsf{Einstein})\}$. □

5.2. *GLAV 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 first consider *DL-Lite*$_\mathcal{R}$ OBDS systems, and show that one such system $\mathcal{E}$ with GLAV extraction assertions can be transformed into a system $\mathcal{E}'$ having GAV extraction assertions only and an analogous behaviour for query answering. That is, the set of certain answers to a CQ q in $\mathcal{E}$ with respect to a document $\mathbf{D}$ coincides with the set of certain answers to q in $\mathcal{E}'$ with respect to $\mathbf{D}$. To this aim we exploit a transformation technique from GLAV to GAV OBDA systems presented in [29]. Since this technique only requires to modify the right-hand side of mapping assertions, which have the same form in both OBDS and OBDA systems, we can in fact apply this transformation exactly as it is given in [29]. For the sake of completeness, we describe below the transformation from [29] (slightly adapted to the OBDS setting).

First thing, we recall that a GLAV extraction assertion r has the form $P(\vec{x}) \rightsquigarrow \Psi(\vec{x})$ where $\Psi(\vec{x})$ is a CQ, i.e. an expression of the form $\exists\vec{y}.\phi(\vec{x}, \vec{y})$, and $P(\vec{x})$ is a document spanner. Given an OBDS system $\mathcal{E} = \langle\mathcal{T}, \mathcal{R}\rangle$, we can thus turn it into a system having only GAV assertions by transforming each assertion $r \in \mathcal{R}$ as follows:

> Substituting each y_i in the right-hand side of r with the term $f_i(\vec{x})$, such that f_i is a fresh function symbol, i.e. it is different from all function symbols used in the assertions in $\mathcal{R}$, it is different from all other fresh function symbols used to transform other extraction assertions, and it is such that $f_i \neq f_j$, for each $i, j \in 1, \ldots, n$, where n is the number of variables in $\vec{y}$;

We denote with $\tau(r)$ the GAV extraction assertion obtained from a GLAV assertion r through the above procedure. Given a set of GLAV extraction assertions $\mathcal{R}$, we define $\tau(\mathcal{R}) = \{\tau(r) \mid r \in \mathcal{R}\}$. The following theorem rephrases in the OBDS setting the analogous theorem given in [29] for OBDA systems.

Theorem 3. *Let $\mathcal{E} = \langle \mathcal{T}, \mathcal{R} \rangle$ be a DL-Lite$_\mathcal{R}$ OBDS system, let q be a CQ, let $\mathbf{D}$ be a document, and let $\mathcal{E}_\tau = \langle \mathcal{T}, \tau(\mathcal{R}) \rangle$. Then $cert(q, \mathcal{E}, \mathbf{D}) = cert(q, \mathcal{E}_\tau, \mathbf{D})$.*

With the above result in place we are thus able to compute the certain answers to a conjunctive query q in a general (i.e. GLAV) OBDS system $\mathcal{E} = \langle \mathcal{T}, \mathcal{R} \rangle$ when the TBox is specified in *DL-Lite*$_\mathcal{R}$. It is indeed sufficient to apply the transformation τ to the set of extraction assertions $\mathcal{R}$, thus obtaining $\mathcal{E}_\tau = \langle \mathcal{T}, \tau(\mathcal{R}) \rangle$, and then proceed with the query rewriting method described in Sec. 5.1, i.e. execute $\mathsf{OBDS\_Rewriting}(\mathcal{E}_\tau, q)$, modulo a trivial split of each extraction assertion in such a way that the resulting set of extraction assertions contains only assertions with a single atom query in their right-hand side (as described in Sec. 3). Certain answers are thus obtained through the evaluation over the underlying document of the spanner returned by $\mathsf{OBDS\_Rewriting}(\mathcal{E}_\tau, q)$ and the use of the coupled target list that this algorithm also returns (see Theorem 2), provided that tuples containing object terms constructed with the fresh function symbols introduced by the transformation τ are excluded from the answer (these tuples are indeed not certain answers, because the object terms produced by τ are denoting only the existence of individuals, which may be different in the various models).

Let us now consider *DL-Lite*$_\mathcal{F}$ OBDS systems. According to [29], Theorem 3 does no longer hold when the TBox is specified in that logic. This is related to the fact that functionalities present in *DL-Lite*$_\mathcal{F}$ OBDS systems (or OBDA systems) induce equalities on existential variables which can never be satisfied by object terms introduced by the transformation τ due to the Unique Name Assumption adopted on *DL-Lite*$_\mathcal{F}$ ontologies.

For this case, as already said at the beginning of this section, we think that input queries should be rewritten in spanners specified in an algebra over regex formulas allowing for recursion, in the same spirit of rewriting algorithms for answering conjunctive queries in data integration systems in the presence of (G)LAV mappings, such as the one proposed in [30].

6. Conclusions

The research in the OBDS framework can be continued in many directions. From the theoretical perspective, it would be obviously interesting to close the case of *DL-Lite*$_\mathcal{F}$ OBDS systems with GLAV extraction assertions, by providing a tailored rewriting technique for this setting. Also, query answering might be investigated for OBDS systems with more expressive languages for the ontology. In particular we plan to study the cases where the TBox is expressed in other DLs for which standard query answering over ontologies is polynomial in data complexity, e.g, $\mathcal{EL}$ [32], or horn DLs [33, 34].

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 [35]. 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.

Bibliography

[1] J. Cowie and Y. Wilks, Information extraction, in *Handbook of Natural Language Processing*, eds. R. Dale, H. Moisl and H. Somers (Marcel Dekker Inc., 2000), pp. 241–259.

[2] D. Jurafsky and J. H. Martin, *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd Edition* (Prentice Hall, Pearson Education International, 2009).

[3] R. Grishman and B. Sundheim, Message understanding conference- 6: A brief history, in *Proc. of the 16th Int. Conf. on Computational Linguistics (COLING)* 1996, pp. 466–471.

[4] S. Sarawagi, Information extraction, *Foundations and Trends in Databases* **1**(3) 261–377 (2008).

[5] R. Hoffmann, C. Zhang, X. Ling, L. S. Zettlemoyer and D. S. Weld, Knowledge-based weak supervision for information extraction of overlapping relations2011, pp. 541–550.

[6] D. Freitag, Machine learning for information extraction in informal domains, *Machine Learning* **39**(2/3) 169–202 (2000).

[7] H. Cunningham, GATE, a general architecture for text engineering, *Computers and the Humanities* **36**(2) 223—-254 (2002).

[8] L. Chiticariu, R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss and S. Vaithyanathan, SystemT: an algebraic approach to declarative information extraction, in *Proc. of the 49th Annual Meeting of the Association for Computational Linguistics (ACL)* 2010, pp. 128–137.

[9] W. Shen, A. Doan, J. F. Naughton and R. Ramakrishnan, Declarative information extraction using datalog with embedded extraction predicates, in *Proc. of the 33rd Int. Conf. on Very Large Data Bases (VLDB)* 2007, pp. 1033–1044.

[10] R. Fagin, B. Kimelfeld, F. Reiss and S. Vansummeren, Document spanners: A formal approach to information extraction, *J. of the ACM* **62**(2) p. 12 (2015).

[11] R. Fagin, B. Kimelfeld, F. Reiss and S. Vansummeren, Declarative cleaning of inconsistencies in information extraction, *ACM Trans. on Database Systems* **41**(1) 6:1–6:44 (2016).

[12] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini and R. Rosati, Ontology-based data access and integration, in *Encyclopedia of Database Systems, Second Edition*, eds. L. Liu and M. T. Özsu 2018.

[13] G. Xiao, D. Calvanese, R. Kontchakov, D. Lembo, A. Poggi, R. Rosati and M. Zakharyaschev, Ontology-based data access: A survey, in *Proc. of the 27th Int. Joint Conf. on Artificial Intelligence (IJCAI)* 2018, pp. 5511–5519.

[14] E. Botoeva, D. Calvanese, B. Cogrel, J. Corman and G. Xiao, Ontology-based data access – Beyond relational sources, *Intelligenza Artificiale* **13**(1) 21–36 (2019).

[15] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini and R. Rosati, Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family, *J. of Automated Reasoning* **39**(3) 385–429 (2007).

[16] B. Motik, A. Fokoue, I. Horrocks, Z. Wu, C. Lutz and B. Cuenca Grau, OWL Web Ontology Language profiles, W3C Recommendation, World Wide Web Consortium (October 2009), Available at `http://www.w3.org/TR/owl-profiles/`.

[17] M. Lenzerini, Data integration: A theoretical perspective., in *Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS)* 2002, pp. 233–246.

[18] A. Doan, A. Y. Halevy and Z. G. Ives, *Principles of Data Integration* (Morgan Kaufmann, 2012).

[19] R. Fagin, P. G. Kolaitis, R. J. Miller and L. Popa, Data exchange: Semantics and query answering, *Theoretical Computer Science* **336**(1) 89–124 (2005).

[20] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi and D. F. Savo, The Mastro system for ontology-based data access, *Semantic Web J.* **2**(1) 43–53 (2011).

[21] G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, R. Rosati, M. Ruzzi and D. F. Savo, MAS-

TRO: A reasoner for effective ontology-based data access, in *Proc. of the 1st Int. Workshop on OWL Reasoner Evaluation (ORE)* 2012.

[22] D. C. Wimalasuriya and D. Dou, Ontology-based information extraction: An introduction and a survey of current approaches, *Information Sciences* **36**(3) 306–323 (2010).

[23] D. Lembo and F. M. Scafoglieri, A formal framework for coupling document spanners with ontologies, in *Proc. of the 2nd IEEE Int. Conf. on Artificial Intelligence and Knowledge Engineering (AIKE)* 2019, pp. 155–162.

[24] F. Baader, D. Calvanese, D. McGuinness, D. Nardi and P. F. Patel-Schneider (eds.), *The Description Logic Handbook: Theory, Implementation and Applications*, 2nd edn. (Cambridge University Press, 2007).

[25] B. Cuenca Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider and U. Sattler, OWL 2: The next step for OWL, *J. of Web Semantics* **6**(4) 309–322 (2008).

[26] S. Abiteboul, R. Hull and V. Vianu, *Foundations of Databases* (Addison Wesley Publ. Co., 1995).

[27] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini and R. Rosati, Linking data to ontologies, *J. on Data Semantics* **X** 133–173 (2008).

[28] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini and R. Rosati, Data complexity of query answering in description logics, *Artificial Intelligence* **195** 335–360 (2013).

[29] G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi and R. Rosati, Using ontologies for semantic data integration, in *A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years.*, 2018, pp. 187–202.

[30] O. M. Duschka, M. R. Genesereth and A. Y. Levy, Recursive query plans for data integration, *J. of Logic Programming* **43**(1) 49–73 (2000).

[31] L. Peterfreund, B. ten Cate, R. Fagin and B. Kimelfeld, Recursive programs for document spanners, in *Proc. of the 22nd Int. Conf. on Database Theory (ICDT)* 2019, pp. 13:1–13:18.

[32] F. Baader, S. Brandt and C. Lutz, Pushing the $\mathcal{EL}$ envelope, in *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI)* 2005, pp. 364–369.

[33] U. Hustadt, B. Motik and U. Sattler, Data complexity of reasoning in very expressive description logics, in *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI)* 2005, pp. 466–471.

[34] M. Ortiz, S. Rudolph and M. Simkus, Query answering in the Horn fragments of the description logics $\mathcal{SHOIQ}$ and $\mathcal{SROIQ}$, in *Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI)* 2011, pp. 1039–1044.

[35] D. Lembo, J. Mora, R. Rosati, D. F. Savo and E. Thorstensen, Mapping analysis in ontology-based data access: Algorithms and complexity, in *Proc. of the 14th Int. Semantic Web Conf. (ISWC)* 2015, pp. 217–234.