Synthesizing Extensional Constraints in Ontology-based Data Access

Marco Console, Maurizio Lenzerini, Riccardo Mancini,
Riccardo Rosati, and Marco Ruzzi

Dipartimento di Ing. Informatica, Automatica e Gestionale "Antonio Ruberti"
SAPIENZA Università di Roma
Via Ariosto 25, I-00186 Roma, Italy
⟨lastname⟩@dis.uniroma1.it

Abstract. Several recent techniques and tools for Ontology-based Data Access (OBDA) make use of the so-called *extensional constraints* (a.k.a. *ABox dependencies*). So far, extensional constraints have been mainly considered in a setting where data are represented in an ABox, instead of external data sources connected to the ontology through declarative mappings. Moreover, the issue of how to generate extensional constraints in practice has not been addressed yet. In this paper we first provide a formal account of the notion of extensional constraints in a full-fledged OBDA setting, where an ontology is connected to the data sources of the information system by means of mappings, and then present an approach to the automatic generation of extensional constraints in such a setting. The technique we propose is based on the use of a first-order theorem prover that checks validity of relevant formulas built over the mapping views. The experiments we have carried out in real-world OBDA projects show the effectiveness of our approach in discovering large collections of extensional constraints entailed by the OBDA specification.

1 Introduction

Ontology-based data management [11] is a novel paradigm for managing, governing and querying the data sources of an organization by means of an ontology. One of the key aspects of ontology-based data management is ontology-based data access (OBDA), where the ontology provides a formal representation of the domain of interest, and is used as a tool provided to the user to express queries. A suitable system is able to answering such queries by reasoning on the ontology and by extracting the appropriate data from the data sources.

Query answering in OBDA is currently a hot research topic (see e.g., [5, 12, 4, 9, 3, 8]). One of the outcomes of this research is a detailed study of the complexity of query answering. In particular, the study of OBDA has focused on understanding which ontology languages allow query answering to be performed with reasonable computational complexity with respect to the size of the data. It is now well-known that efficiency of query answering cannot be guaranteed if the ontology is expressed in a logic whose expressive power exceeds the one of lightweight language [6]. With such languages, in particular with the languages of the *DL-Lite* family, conjunctive queries are *first-order rewritable*, i.e., answering (unions of) conjunctive queries, (U)CQs, expressed over the

ontology can be reduced to the evaluation of a suitable first-order query (called the perfect rewriting of the original query) expressed over the data sources.

Recent works on this subject go beyond the mere rewritability property, and study optimization techniques aiming at minimizing the size of the rewritten query. Many proposals of this type assume that the query answering algorithm make use of additional knowledge with respect to the usual input, which is constituted by the TBox representing the ontology and the data at the sources. In [13, 15, 14] such additional knowledge is expressed in terms of the so-called extensional constraints, where an extensional constraint is an axiom that has the same form of a TBox assertion, but is interpreted as an integrity constraint for the ABox (i.e., the set of asserted instances). An example of an extensional constraint is an inclusion assertion $A_1 \sqsubseteq_{\mathsf{ext}} A_2$ specifying that the instances that have been asserted for one concept A_1 in the ABox form a subset of the instances that have been asserted for the concept A_2. By using such assertions, the query answering algorithm is able to prune the space of query rewriting, for example by removing those queries that are contained into other queries under the extensional constraints [13, 15]. For a very simple example, if we know that the extension of the concept ForeignStudent is a subset of the extension of the concept Student, and if the query asks for all instances of Student, then the query answering algorithm can simply avoid accessing the instances of ForeignStudent.

So far, extensional constraints have been mainly considered in the simplified setting where data in the OBDA system are represented in an ABox, i.e., they are stored in the form of a set of facts (possibly managed by a Database Management System - DBMS) in the alphabet of the ontology. Conversely, in a full-fledged OBDA system, the ABox does not really exist. Rather, the TBox is mapped (through mapping assertions) to existing data sources that are tipically managed by external systems (often, DBMSs). In our experience, full-fledged OBDA is a very effective paradigm in practice, because it allows mapping the ontology to the existing data sources of the organization (which, therefore, need not to be modified). Since the data sources and the ontology are not necessarily mutually coherent, the mapping mechanisms should be able to connect any query over the data sources to the elements of the ontology. In this sense, OBDA can be seen as a form of information integration, where data are not duplicated or moved to an ABox, but remain in the data sources of the organization, continuing to serve their purposes in the applications. Once we accept that data remain at the sources, we also accept that they are subject to updates, according to the applications using them. It follows that deriving extensional constraints by inspecting the current state of data might be too expensive, or even unfeasible at all. For this reason, we follow the approach alread implicit in [13], based on static analysis of the OBDA specification. In other words, we aim at deriving the extensional constraints from the OBDA specification, independently of the specific state (i.e., on the specific data) of the current database.

We also observe that the issue of how to generate extensional constraints has not been addressed extensively. For example, while [13] discusses the idea of inducing extensional constraints from mapping assertions, it does not present any systematic techniques for doing so.

Based on the above considerations, in this paper we provide the following contributions.

– We present (Section 3) a formal account of the notion of extensional constraints in a full-fledged OBDA setting, where an ontology is connected to the data sources of the information system by means of mappings. In particular, we define the notion of *valid* extensional constraint in an OBDA specification.
– We observe that, due to the high expressive power of the mapping language, deriving all the extensional constraints that are valid in an OBDA specification is undecidable. Therefore, we propose (Section 4) a "best effort" approach to the automatic generation of extensional constraints in such setting. The technique we present is based on the use of a first-order theorem prover that checks validity of relevant formulas built over the mapping views.
– We report (Section 5) on a first set of experiments we have carried out in real-world OBDA projects. Such experiments demonstrate the effectiveness of our approach in discovering large collections of extensional constraints entailed by the OBDA specification, and prove that the use of theorem provers for this kind of task is very promising.

2 Preliminaries

An OBDA system specification $\mathcal{B} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$ is characterized by three components: the ontology $\mathcal{T}$, the source schema $\mathcal{S}$ and the mapping $\mathcal{M}$ between the source schema and the ontology.

In this work, we assume that $\mathcal{T}$ is specified as a TBox in *DL-Lite*$_A$ [12], which is a logic of the *DL-Lite* family[1] allowing for UCQ-rewritability of query answering, i.e., answering unions of conjunctive queries (UCQ) posed over it can be reduced to evaluation of a UCQ over the underlying data. *DL-Lite*$_A$ allows for specifying *concepts*, representing sets of objects, *roles*, representing binary relations between objects, and *attributes*, representing binary relations between objects and values. The syntax of concept, role and attribute *expressions* in *DL-Lite*$_A$ is as follows:

$$\begin{array}{lll} B \longrightarrow A \mid \exists Q \mid \delta(U) & R \longrightarrow Q \mid \neg Q & E \longrightarrow \rho(U) \\ C \longrightarrow B \mid \neg B & Q \longrightarrow P \mid P^- & F \longrightarrow T_1 \mid \cdots \mid T_n \\ & & V \longrightarrow U \mid \neg U \end{array}$$

In such rules, A, P, and U denote a *concept name*, a *role name*, and an *attribute name*, respectively. P^- denotes the *inverse of a role*. B and R are called *basic concept* and *basic role*, respectively. $\neg B$ (resp. $\neg Q$, $\neg U$) denotes the *negation* of a basic concept B (resp. basic role, or attribute). The concept $\exists Q$, also called *unqualified existential restriction*, denotes the *domain* of a role Q, i.e., the set of objects that Q relates to some object. Similarly, the concept $\delta(U)$ denotes the *domain* of an attribute U, i.e., the set of objects that U relates to some value. Conversely, $\rho(U)$ denotes the *range* of an attribute U, i.e., the set of values to which U relates some object. $T_1, \ldots, T_n$ are unbounded pairwise disjoint predefined *value-domains*.

A *DL-Lite*$_A$ TBox $\mathcal{T}$ is a finite set of assertions of the form

$$B \sqsubseteq C \qquad Q \sqsubseteq R \qquad E \sqsubseteq F \qquad U \sqsubseteq V \qquad (\mathsf{funct}\ Q) \qquad (\mathsf{funct}\ U)$$

[1] Not to be confused with the set of DLs studied in [2], which form the *extended DL-Lite* family, also called the *DL-Lite*$_{bool}$.

From left to right, assertions denote inclusions between concepts, roles, value-domains, attributes, and functionality on roles and on attributes, respectively. Notice that in *DL-Lite*$_A$ TBoxes we impose that roles and attributes occurring in functionality assertions cannot be specialized (i.e., they cannot occur in the right-hand side of inclusions).

The *source schema* $\mathcal{S}$ is a relational database schema, specifying both the structure of the relational tables in the database, and the integrity constraints that the tables should obey. A database D for $\mathcal{S}$ is called *legal* if it satisfies all the integrity constraints in $\mathcal{S}$ [1]. To express queries over the source schema, we use SQL. Given an n-tuple of variables $\boldsymbol{x} = x_1, \ldots, x_n$, we use the notation $Q_{DB}(\boldsymbol{x})$ for an SQL query of arity n, where x_i denotes the i-th attribute in the target list of the query. To simplify notation, we assume that query attribute names are obtained by renaming attributes in the target list of the SQL query with variable symbols. When the specification of the attributes $\boldsymbol{x}$ is not necessary, we simply refer to the symbol Q_{DB}.

As for the mapping $\mathcal{M}$ between the ontology and the source schema, we will use a form of mapping assertion that slightly generalizes the well-known GAV mappings [10]. An *extended GAV mapping assertion* m is an expression of the form

$$Q_{DB}(\boldsymbol{x}) \rightsquigarrow cq(\boldsymbol{x})$$

where $Q_{DB}(\boldsymbol{x})$ is an SQL query over $\mathcal{S}$ (called the *view* associated to m), and $cq(\boldsymbol{x})$ is a conjunctive query over $\mathcal{T}$ whose variables are those in $\boldsymbol{x}$. The intuitive meaning of a mapping assertion m of the above form is that every tuple satisfying view Q_{DB} also satisfies the ontology query cq. Note that the extended GAV mapping language has the same expressive power as ordinary GAV. However, in practice, the possibility of using more than one atom in the head of a mepping assertion greatly simplifies the task of the ontology engineer.

As for the semantics of the ontology we adopt the standard semantics based on first-order interpretations, and we refer the reader to [7] for a formal, detailed presentation of the semantics of the whole OBDA specification.

Example 1. We now present an OBDA system specification which is an excerpt of the one we used in a real-world experimental project carried out in collaboration with the Ministry of Economy and Finance.

PublicOrg $\sqsubseteq$ Organization	PublicDep $\sqsubseteq$ PublicOrg	(funct name)
$\exists$worksWith $\sqsubseteq$ Organization	$\exists$worksWith$^-$ $\sqsubseteq$ Organization	(funct address)

The concepts in the this fragment of the ontology are Organization (denoting organizations), PublicOrg (denoting public organizations), and PublicDep (denoting public departments). The inclusion assertions in the first row state that public organizations are particular organizations, and public departments are particular public organizations. The role worksWith relates organizations that work together (second row). The attributes of the ontology are name, address, prjName. The axioms in the third column specify that name and address are functional.

The source schema $\mathcal{S}$ contains four relational tables. Dept_MinistryA(dep_id,dep_name) and Dept_MinistryB(dep_id,dep_addr) store data about departments belonging to MinistryA and MinistryB respectively. Works_On(dep_id,proj_name) store data about projects carried out by departments and Cooperate(dept1,dept2) specify pairs of cooperating departments.

M_1 $SELECT dep\_id AS x, dep\_name AS y$ $FROM Dept\_MinistryA$ $\rightsquigarrow \{x, y \mid \mathsf{PublicDep}(x) \wedge \mathsf{name}(x, y)\}$

M_2 $SELECT dep\_id AS x, dep\_addr AS y$ $FROM Dept\_MinistryB$ $\rightsquigarrow \{x, y \mid \mathsf{PublicDep}(x) \wedge \mathsf{address}(x, y)\}$

M_3 $SELECT w1.dep\_id as x, w2.dep\_id as y,$ $w2.proj\_name as z FROM Works\_On w1,$ $Works\_On w2, Dept\_MinistryA d1,$ $Dept\_MinistryA d2$ $WHERE d1.dep\_id = w1.dep\_id$ $AND d2.dep\_id = w2.dep\_id$ $AND w1.proj = w2.proj$ $AND w1.dep\_id <> w2.dep\_id$ $\rightsquigarrow \{x, y, z \mid \mathsf{worksWith}(x, y) \wedge \mathsf{prjName}(x, z) \wedge \mathsf{prjName}(y, z)\}$

M_3 $SELECT d1.dep\_id as x, d2.dep\_id as y$ $FROM Cooperate c, Dept\_MinistryB d1,$ $Dept\_MinistryB d2$ $WHERE c.dept1 = d1.dep\_id$ $AND c.dept2 = d2.dep\_id$ $\rightsquigarrow \{x, y \mid \mathsf{worksWith}(x, y)\}$

Fig. 1: Mapping assertion of Example 1

The mapping $\mathcal{M}$ between the sources and the ontology contains the mapping assertions illustrated in Figure 1. In words, M_1 relates table `Dept_MinistryA` to the instances of concept PublicDep and their names. M_2 relates table `Dept_MinistryB` to the instances of PublicDep and their addresses; M_3 specifies that departments from MinistryA that work on the same project actually work together, and therefore are mapped to worksWith, together with the indication of the project they work for (attribute prjName). Finally, M_4 maps the notion of cooperation represented in table `Cooperate` to the role worksWith in the ontology.

Query answering in current OBDA systems [4] is performed by means of a two-phase query rewriting technique: the user query is first rewritten taking into account the ontology, and then considering the mappings. The first phase, called *ontology rewriting phase*, encodes part of the knowledge expressed by the TBox in the rewriting. More precisely, given an OBDA specification $\mathcal{B} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$ and a query Q over $\mathcal{T}$, an *ontology rewriting* of Q under $\mathcal{B}$ is a query Q' over $\mathcal{T}$ such that the certain answers to $\mathcal{Q}'$ with respect to $\langle \emptyset, \mathcal{M}, \mathcal{S} \rangle$ are exactly the certain answers to Q with respect to $\mathcal{B}$. Once ontology rewriting has been performed, the TBox can be disregarded and the second phase, called *mapping rewriting phase*, can consider only mapping assertions. The outcome of this second step produces an SQL query that is evaluated directly over the sources: given and OBDA specification $\mathcal{B} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$ and a query Q' over $\mathcal{T}$, a *mapping rewriting* is a query Q'' over $\mathcal{S}$ such that, for every legal database D for $\mathcal{S}$, the answers to Q'' over D are exactly the certain answers of Q' with respect to $\langle \emptyset, \mathcal{M}, \mathcal{S} \rangle$. For further details on OBDA query rewriting we refer the reader to e.g. [7].

3 Extensional constraints

In this section we formally define the notion of extensional constraint in an OBDA specification.

Definition 1. *Let $\mathcal{B} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$ be an OBDA specification. An extensional constraint for $\mathcal{B}$ is a statement of one of the following forms*

$$\begin{array}{cccc} B_1 \sqsubseteq_{\mathsf{ext}} B_2 & B_1 \sqsubseteq_{\mathsf{ext}} \neg B_2 & Q_1 \sqsubseteq_{\mathsf{ext}} Q_2 & Q_1 \sqsubseteq_{\mathsf{ext}} \neg Q_2 \\ U_1 \sqsubseteq_{\mathsf{ext}} U_2 & U_1 \sqsubseteq_{\mathsf{ext}} \neg U_2 & (\mathsf{funct}_{\mathsf{ext}}\ Q) & (\mathsf{funct}_{\mathsf{ext}}\ U) \end{array}$$

where B_1, B_2 are basic concepts, Q, Q_1, Q_2 are roles, and U, U_1, U_2 are attributes.

Notice that, by virtue of the characteristics of *DL-Lite*$_A$, the set of all possible extensional constraints for an OBDA specification is finite. Notice also that the syntax of extensional constraints is essentially the same as the syntax of TBox assertions in *DL-Lite*$_A$. Intuitively, the difference between the two types of assertion is that, while a TBox assertion states a general condition that is to be satisfied in all the models of the TBox, an extensional constraint states a condition that the instances that have been explicitly asserted in the knowledge base must satisfy. For example, an extensional inclusion assertion between concepts (first type of assertions in the above list) specifies that the instances that have been asserted for the left-hand side concept form a subset of the instances that have been asserted for the right-hand side concept. In other words, an extensional constraint is a sort of meta-level constraint regarding the instances explicitly asserted in the knowledge base.

The question now is to specify what does it mean for an instance to be explicitly asserted in the knowledge base. Recent papers dealing with extensional constraints assume that the knowledge base is constituted by a TBox and an ABox. In that context, the explicitly asserted instances of an element of the knowledge base are those appearing in the ABox. In this paper, the notion of OBDA specification is such that the knowledge base is constituted by a TBox, a set of mappings and a database schema, and this has to be taken into account in specifying the semantics of extensional constraints.

To formally define the semantics of an extensional constraint, we make use of several notions, that we now introduce. First, let us remind the reader about the meaning of an extensional constraint in the case where we have a DL knowledge base $\langle \mathcal{T}, \mathcal{A} \rangle$ constituted by a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. If γ is an extensional constraint, let us indicate with γ_{int} the TBox axiom obtained by dropping the subscript ext from α. For example, if γ is $A_1 \sqsubseteq_{\mathsf{ext}} A_2$, then γ_{int} is simply $A_1 \sqsubseteq A_2$. We define the $\mathcal{A}$-interpetation $\mathcal{I}_{\mathcal{A}}$ for $\mathcal{T}$ as the interpretation for $\mathcal{T}$ such that a fact α is true in $\mathcal{I}_{\mathcal{A}}$ if and only if $\alpha \in \mathcal{A}$. It is now immediate to define the notion of satisfaction for extensional constraints: an extensional constraint γ is said to hold in an ABox $\mathcal{A}$ if γ_{int} is satisfied by $\mathcal{I}_{\mathcal{A}}$.

Let us now turn our attention to extensional constraints in an OBDA specification. Since an extensional constraint says something about the extension of predicates, the notion of satisfaction requires to fix such extensions. In an OBDA system, the extension of predicates depends upon the data at the sources. It follows that in order to check whether an extensional constraint is satisfied or not, we should resort to a database D for $\mathcal{S}$, and then verify whether the condition corresponding to the extensional constraint is satisfied by the instances that the mapping retrieves from the data sources. To make this idea precise, we introduce the notion of "ABox retrieved from a database through a mapping".

Given an OBDA specification $\mathcal{B} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$, and a database D that is legal for $\mathcal{S}$, a mapping $m : Q_{DB}(\boldsymbol{x}) \rightsquigarrow cq(\boldsymbol{x})$ in $\mathcal{M}$ is said to generate a fact $P(\boldsymbol{t})$ (where P is a predicate in $\mathcal{T}$ and $\boldsymbol{t}$ is a tuple of constants of the same arity as P) from D if (i) $\boldsymbol{t}$ is a

tuple in Q_{DB}^{D}, (ii) $cq(\boldsymbol{t})$ is the set of facts obtained by substituting $\boldsymbol{t}$ for $\boldsymbol{x}$ in $cq(\boldsymbol{x})$, (iii) $P(\boldsymbol{t})$ is an atom in $cq(\boldsymbol{t})$.

The *retrieved ABox associated to $\mathcal{B}$ and D*, denoted $\mathcal{M}(D)$, is the set of ABox assertions defined as $\{P(\boldsymbol{t}) \mid \text{there is } m \in \mathcal{M} \text{ that generates } P(\boldsymbol{t}) \text{ from } D\}$. With the notion of retrieved ABox, we can now specify when an extensional constraint is satisfied by an OBDA specification and a database.

Definition 2. *Let $\mathcal{B} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$ be an OBDA specification, D a legal database for $\mathcal{S}$, and γ an extensional constraint for $\mathcal{B}$. Then γ is* satisfied by *$(\mathcal{B}, D)$ if γ holds in $\mathcal{M}(D)$.*

The notion of satisfaction of an extensional constraint that we have just illustrated is relative to a specific database D. In OBDA, the underlying database may be subject to frequent changes. Therefore, basing our knowledge about the extensional constraints on the specific database D at hand would imply to compute the set of relevant constraints every time the database is modified. Since this is unfeasible, our approach is based on the idea of considering the problem of deriving extensional constraints as a static analysis problem, i.e., as a reasoning task over the OBDA specification. More specifically, the extensional constraints that we will use for the optimization of query rewriting will be those that are satisfied "independently" of the specific database at hand. To capture this intuition, we introduce the notion of valid extensional constraints.

Definition 3. *Let $\mathcal{B} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$ be an OBDA specification, and γ an extensional constraint for $\mathcal{B}$. Then γ is* valid *in $\mathcal{B}$ (or* entailed *by $\mathcal{B}$) if for every legal database D for $\mathcal{S}$, we have that γ is satisfied by $(\mathcal{B}, D)$.*

The main problem addressed in this paper is defined as follows.

Definition 4. *The problem of* computing the extensional closure *of an OBDA specification is the following: given an OBDA specification $\mathcal{B} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$, compute the set of all extensional constraints for $\mathcal{B}$ that are valid in $\mathcal{B}$.*

Obviously, at the basis of the problem of computing the extensional closure of an OBDA specification, there is an associated decision problem, defined as follows.

Definition 5. Extensional constraint entailment *is the following decision problem: given an OBDA specification $\mathcal{B} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$, and an extensional constraint γ for $\mathcal{B}$, check whether γ is valid in $\mathcal{B}$.*

Based on the fact the deciding containment between SQL queries is undecidable, it is immediate to verify that the extensional constraint entailment problem is undecidable.

Theorem 1. *Extensional constraint entailment is undecidable.*

So, on the one hand we aim at computing all the extensional constraints that are valid in an OBDA specification; on the other hand, this task is undecidable. To address this problem, we could in principle decide to resort to a less powerful query language in the mapping $\mathcal{M}$. For example, if we use conjunctive queries in $\mathcal{M}$, then it is easy to see that computing the closure of extensional constraints becomes feasible (since query containment becomes decidable). Unfortunately, real world OBDA projects show

that one needs the full expressive power of SQL (or, at least, the fragment of SQL corresponding to first-order logic queries) in modeling the mapping between the data sources and the ontology. In order to provide a solution to this problem, we propose to use a "best effort" approach, with the goal of deriving as many valid extensional constraints as possible. In the rest of this paper, we show an approach based on the use of a first-order theorem prover.

4 A method for synthesizing extensional constraints

We now describe our method for synthesizing valid extensional constraints from an OBDA specification.

Translation of source schema constraints Let $\mathcal{S}$ be a relational schema with integrity constraints. We translate (a subset of) the integrity constraints of $\mathcal{S}$ into FO-sentences. For ease of exposition, we consider the most used integrity constraints in relational databases, i.e., key constraints and foreign key constraints [1] (other kinds of constraints that can be captured by first-order sentences could be considered as well):

- every key constraint $key(r) = 1, \ldots, k$ is translated into the sentence
$$\forall x_1, \ldots, x_n, y_{k+1}, \ldots, y_n \\ (r(x_1, \ldots, x_n) \wedge r(x_1, \ldots, x_k, y_{k+1}, \ldots, y_n) \rightarrow x_{k+1} = y_{k+1} \wedge \ldots \wedge x_n = y_n)$$
- every foreign key constraint $r_1[a_1, \ldots, a_k] \subseteq r_2[b_1, \ldots, b_k]$ is translated into the sentence $\forall x_1, \ldots, x_n (r_1(x_1, \ldots, x_n) \rightarrow \exists y_1, \ldots, y_{n-k}(r_2(t_1, \ldots, t_m)))$, where, for every $j \in \{1, \ldots, k\}$, $t_{b_j} = x_{a_j}$ (and $t_i = y_i$ for the other positions).

Let $\tau_{ic}(\mathcal{S})$ be the first-order sentence corresponding to the conjunction of all the above sentences.

Translation of extensional constraints In the following, and without loss of generality, we will consider a normalized, "pure GAV" version of the set of mappings, in which the head of every mapping assertion is constituted by a single atom (and the SQL query is projected onto the attributes mentioned in such an atom). In fact, given an extended GAV mapping $\mathcal{M}$, it is easy to compute (in polynomial time) a pure GAV mapping $split(\mathcal{M})$ that is equivalent to $\mathcal{M}$.

Given a first-order formula ϕ whose free variables are $x_1, \ldots, x_n$, and given an n-tuple of terms $\boldsymbol{t}$, we denote by $\phi(\boldsymbol{t})$ the formula obtained from ϕ by replacing the free variables $x_1, \ldots, x_n$ with the terms $\boldsymbol{t}$.

Given an SQL query $Q_{DB}(\boldsymbol{x})$ used in the mapping $m \in \mathcal{M}$, there are two cases: (i) $Q_{DB}(\boldsymbol{x})$ corresponds to a first-order query, i.e., a first-order formula with free variables $\boldsymbol{x}$. In this case, let $\tau_{fol}(Q_{DB}(\boldsymbol{x}))$ be such a first-order query; (ii) $Q_{DB}(\boldsymbol{x})$ does not correspond to any first-order query (e.g., because of aggregation operations). In this case, we make use of a new relation name r_m (i.e., a relation name not used in $\mathcal{S}$) and define $\tau_{fol}(Q_{DB}(\boldsymbol{x})) = r_m(\boldsymbol{x})$. The intuition behind this forced translation of non-first-order SQL queries is to encode them with new, generic predicates (using a different predicate for every such query) that have no special property and that are used to encode queries whose semantics is "unknown". Then, given a mapping assertion ξ of the form $Q_{DB}(\boldsymbol{x}) \rightsquigarrow A(\boldsymbol{x})$, we define $\tau_{fol}(\xi)$ as $\tau_{fol}(Q_{DB}(\boldsymbol{x}))$.

Given a set of mappings $\mathcal{M}$ and a concept name A, the set of *relevant mappings for* A (denoted by $RelMap(A, \mathcal{M})$) is the set of mapping assertions $\xi \in \mathcal{M}$ such that the head of ξ has the form $A(x)$. Analogously, given a role name P, $RelMap(P, \mathcal{M})$) is the set of mapping assertions $\xi \in \mathcal{M}$ such that the head of ξ has the form $P(x_1, x_2)$, and given an attribute name U, $RelMap(U, \mathcal{M})$) is the set of mapping assertions $\xi \in \mathcal{M}$ such that the head of ξ has the form $U(x_1, x_2)$.

Given a concept A, the *FOL-unfolding* $\tau_{unf}(\alpha, \mathcal{M})$ of a basic concept atom, basic role atom, or attribute atom α with respect to a set of mappings $\mathcal{M}$ is defined as follows:

$$\begin{aligned}
\tau_{unf}(A(x), \mathcal{M}) &= \bigvee_{\xi \in RelMap(A,\mathcal{M})} \tau_{fol}(\xi)(x) \\
\tau_{unf}(\exists P(x), \mathcal{M}) &= \bigvee_{\xi \in RelMap(P,\mathcal{M})} \exists y(\tau_{fol}(\xi)(x, y)) \\
\tau_{unf}(\exists P^-(x), \mathcal{M}) &= \bigvee_{\xi \in RelMap(P,\mathcal{M})} \exists y(\tau_{fol}(\xi)(y, x)) \\
\tau_{unf}(P(x_1, x_2), \mathcal{M}) &= \bigvee_{\xi \in RelMap(P,\mathcal{M})} \tau_{fol}(\xi)(x_1, x_2) \\
\tau_{unf}(P^-(x_1, x_2), \mathcal{M}) &= \bigvee_{\xi \in RelMap(P,\mathcal{M})} \tau_{fol}(\xi)(x_2, x_1) \\
\tau_{unf}(U(x_1, x_2), \mathcal{M}) &= \bigvee_{\xi \in RelMap(U,\mathcal{M})} \tau_{fol}(\xi)(x_1, x_2)
\end{aligned}$$

Then, we define the function τ_{ec} that translates *DL-Lite*$_A$ extensional constraints with respect to a set of mappings $\mathcal{M}$ as follows.

- *inclusion axioms between basic concepts*: let ϕ be the axiom $B_1 \sqsubseteq B_2$. Then, $\tau_{ec}(\phi, \mathcal{M}) = \forall x(\tau_{unf}(B_1(x), \mathcal{M}) \rightarrow \tau_{unf}(B_2(x), \mathcal{M}))$;
- *disjointness axioms between basic concepts*: let ϕ be the axiom $B_1 \sqsubseteq \neg B_2$. Then, $\tau_{ec}(\phi, \mathcal{M}) = \forall x(\tau_{unf}(B_1(x), \mathcal{M}) \rightarrow \neg\tau_{unf}(B_2(x), \mathcal{M}))$;
- *inclusion axioms between basic roles (and attributes)*: let ϕ be the axiom $R_1 \sqsubseteq R_2$. Then, $\tau_{ec}(\phi, \mathcal{M}) = \forall x, y(\tau_{unf}(R_1(x, y), \mathcal{M}) \rightarrow \tau_{unf}(R_2(x, y), \mathcal{M}))$ (analogously for inclusions axioms between attributes);
- *disjointness axioms between basic roles (and attributes)*: let ϕ be the axiom $R_1 \sqsubseteq \neg R_2$. Then, $\tau_{ec}(\phi, \mathcal{M}) = \forall x, y(\tau_{unf}(R_1(x, y), \mathcal{M}) \rightarrow \neg\tau_{unf}(R_2(x, y), \mathcal{M}))$ (analogously for disjointness axioms between attributes);
- *role (and attribute) functionality axioms*: let ϕ be the axiom $(\mathsf{funct}\ R)$. Then, $\tau_{ec}(\phi, \mathcal{M}) = \forall x, y, z(\tau_{unf}(R(x, y), \mathcal{M}) \wedge \tau_{unf}(R(x, z), \mathcal{M}) \rightarrow y = z)$ (analogously for attribute functionality axioms).

Verification of extensional constraints The above first-order translation of both database integrity constraints and extensional constraints is related to the entailment of extensional constraints by the following theorem.

Theorem 2. *Let $\mathcal{B} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$ be an OBDA specification and let ϕ be an extensional constraint defined over the alphabet of $\mathcal{T}$. If $\tau_{ic}(\mathcal{S}) \rightarrow \tau_{ec}(\phi, \mathcal{M})$ is a valid first-order sentence, then ϕ is valid in $\mathcal{B}$.*

Notice that the converse of above theorem holds in the case when: (i) all constraints on the database schema $\mathcal{S}$ are keys or foreign key constraints; (ii) all the SQL queries used in the mapping $\mathcal{M}$ correspond to first-order queries.

An (incomplete) algorithm for synthesizing extensional constraints Based on Theorem 2, we are now able to define a technique for automatically deriving a set of extensional constraints entailed by an OBDA specification $\mathcal{B} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$.

The algorithm for the automatic processing of the OBDA specification and generation of the extensional constraints is the following.

Algorithm computeObdaEC
Input: an OBDA specification $\mathcal{B} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$
Output: a set Φ of extensional constraints entailed by $\mathcal{B}$
begin
  $\Phi = \emptyset$; compute $\mathcal{M}' = split(\mathcal{M})$; compute $\tau_{ic}(\mathcal{S})$;
  for each extensional constraint ϕ over the alphabet of $\mathcal{T}$ **do**
  begin
    compute $\tau_{ec}(\phi, \mathcal{M}')$;
    ask a FO theorem prover whether the sentence $\tau_{ic}(\mathcal{S}) \rightarrow \tau_{ec}(\phi, \mathcal{M}')$ is valid;
    if the FO theorem prover answers YES before a fixed timeout
    then $\Phi = \Phi \cup \{\phi\}$
  end;
  return Φ
end

The algorithm computeObdaEC starts by computing the pure GAV version $\mathcal{M}'$ of the mapping $\mathcal{M}$ough the function $Split$. Then, it generates *all* the possible extensional constraints that can be formulated in *DL-Lite*$_A$ over the alphabet of the TBox $\mathcal{T}$; such a set is finite and can be computed in polynomial time with respect to the size of $\mathcal{T}$. For every such extensional constraint ϕ, the algorithm builds the first-order sentence $\tau_{ic}(\mathcal{S}) \rightarrow \tau_{ec}(\phi, \mathcal{M}')$ that encodes the entailment of ϕ with respect to $\mathcal{B}$, and verifies the validity of such a formula. To solve the first-order validity problems mentioned in Theorem 2, the algorithm resorts to an external first-order theorem prover.

Of course, Theorem 1 implies that we cannot hope to actually derive the *whole* extensional closure of the OBDA specification: however, we can compute a sound approximation (i.e., a subset) of such an extensional closure. This incomplete solution is certainly acceptable, since extensional constraints are used to optimize reasoning about the OBDA, and even an incomplete set of such constraints can effectively be used for such a goal.

5 Experiments

We have implemented the algorithm computeObdaEC in a tool for discovering extensional constraints entailed by an OBDA specification. This tool provides a multi-threading implementation which takes advantage of the modern multi-core architectures, performing several entailment tests at the same time. With respect to the algorithm presented in Section 4, our implementation only considers extensional inclusions.

We experimented the tool within a joint project between Sapienza University of Rome and the Italian Ministry of Economy and Finance. The project includes an ontology defined over an alphabet containing 164 concept names, 47 role names, 86 attribute names and a set of 263 mapping assertions involving around 60 relational tables. The full list of mapping assertions used for the experiments, along with the source database schema, are available at `http://hyper.dis.uniroma1.it/~quonto/dl2013/`. In the same page it is possible to find the whole set of newly discovered extensional constraints (expressed in OWL functional-style syntax). We remark that no constraints are asserted on the original database. Therefore the discovery of extensional constraints is based only on mapping assertions.

We ran the tests over an 8 Core CPU machine equipped with 8 GB RAM: such an architecture allowed us to run 8 threads simultaneously. The theorem prover used was *E-Prover*[2], configured with 1024 MB of maximum memory and a time-out of 30 seconds. We used the computeObdaEC algorithm to generate all the possible entailment problems for inclusion dependencies and submitted them to the theorem prover. The algorithm ran 798336 tests in total, and only in the 5,45% of the cases the proof exceeded the 30 seconds time out and was terminated, whereas in the 0.019% of cases the presence of non-FOL operators (e.g., use of the `group by` clause) prevented the formulation of the containment problem.

The experimental results show the effectiveness of the proposed methodology with respect to the goal of discovering extensional constraints: 177 extensional inclusion dependencies were discovered. The whole analysis took 46h 47m 51s. We consider this amount of time reasonable, although considerably high, because this formal analysis of mappings is conducted off-line; moreover, the set of derived extensional constraints can be easily adapted to changes in the set of mappings, by adding incrementally new entailed constraints when new mapping assertions are added. Similarly, extensional constraints can be deleted when mappings that generated it are removed from the set of mappings.

6 Conclusions

We have introduced the notion of extensional constraints in a full-fledged OBDA setting, and we have illustrated a technique for synthesizing a set of extensional constraints that are valid in an OBDA specification. Our technique is based on the use of a first-order theorem prover, and the experiments we have carried out in real-world projects show that the approach is very promising.

In the future, we aim at continuing our work along several directions. First, we want to extend our approach to deal with the integrity contraints defined in the database schema, based on the observation that such constraints may help in deriving new extensional constraints. We also plan to check how the extensional constraints derived by our method affect the whole query answering process (using, e.g., the Prexto query rewriting algorithm [15]). The first experiments in this respect are promising. For example the rewriting of the query $q(x) : -senza\_underlying(x), ISIN(x, y)$ computed without taking into account extensional constraints contains 186 disjuncts, while such number drops to 18 if we consider the derived extensional constraints (the two rewritings are shown in `http://hyper.dis.uniroma1.it/~quonto/dl2013/`). Finally, we want to consider even more features of SQL, and study mechanisms for using first-order theorem provers for approximating the reasoning tasks needed to derive extensional constraints.

Acknowledgments. This research has been partially supported by the EU under FP7 project Optique – Scalable End-user Access to Big Data (grant n. FP7-318338).

[2] `http://www.eprover.org`

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., 1995.
2. A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev. The *DL-Lite* family and relations. *J. of Artificial Intelligence Research*, 36:1–69, 2009.
3. A. Calì, G. Gottlob, and A. Pieris. New expressive languages for ontological query answering. In *Proc. of AAAI 2011*, pages 1541–1546, 2011.
4. 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.
5. 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.
6. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Conceptual modeling for data integration. In A. T. Borgida, V. Chaudhri, P. Giorgini, and E. Yu, editors, *Conceptual Modeling: Foundations and Applications – Essays in Honor of John Mylopoulos*, volume 5600 of *LNCS*, pages 173–197. Springer, 2009.
7. F. Di Pinto, D. Lembo, M. Lenzerini, R. Mancini, A. Poggi, R. Rosati, M. Ruzzi, and D. F. Savo. Optimizing query rewriting in ontology-based data access. In *Proc. of EDBT 2013*, 2013.
8. G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization. In *Proc. of ICDE 2011*, pages 2–13, 2011.
9. R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The combined approach to query answering in *DL-Lite*. In *Proc. of KR 2010*, pages 247–257, 2010.
10. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of PODS 2002*, pages 233–246, 2002.
11. M. Lenzerini. Ontology-based data management. In *Proc. of CIKM 2011*, pages 5–6, 2011.
12. 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.
13. M. Rodríguez-Muro and D. Calvanese. Dependencies: Making ontology based data access work in practice. In *Proc. of AMW 2011*, volume 749 of *CEUR,* `ceur-ws.org`, 2011.
14. M. Rodriguez-Muro and D. Calvanese. High performance query answering over *DL-Lite* ontologies. In *Proc. of KR 2012*, pages 308–318, 2012.
15. R. Rosati. Prexto: Query rewriting under extensional constraints in DL-Lite. In *Proc. of ESWC 2012*, 2012.