Open access • Proceedings Article • DOI:10.1145/1458484.1458500

Aggregate queries over ontologies — Source link

Diego Calvanese, Evgeny Kharlamov, Werner Nutt, Camilo Thorne

Institutions: Free University of Bozen-Bolzano

Published on: 30 Oct 2008 - Ontologies and Information Systems for the Semantic Web

Topics: Conjunctive query, Ontology language, Semantics, Description logic and Semantic Web

Aggregate Queries over Ontologies

Diego Calvanese, Evgeny Kharlamov, Werner Nutt, Camilo Thorne
Free University of Bozen-Bolzano
Piazza Domenicani, 3
39100 Bozen-Bolzano, Italy
{calvanese,kharlamov,nutt,cthorne}@inf.unibz.it

ABSTRACT

Answering queries over ontologies is an important issue for the Semantic Web. Aggregate queries were widely studied for relational databases but almost no results are known for aggregate queries over ontologies. In this work we investigate the latter problem. We propose syntax and semantics for epistemic aggregate queries over ontologies and study query answering for `MAX, MIN, COUNT, CNTD, SUM, AVG` queries for the ontology language *DL-Lite*$_{\mathcal{A}}$.

Categories and Subject Descriptors

H.2.3 [**Languages**]: Query languages; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Algorithms, Languages, Theory

Keywords

Aggregate Queries, Incomplete Information, Ontology Languages, Description Logic

1. INTRODUCTION

In *ontology-based data access* (OBDA), typical of the Semantic Web [13], Enterprise Information Integration, and Data Integration [15], ontologies are used to provide a conceptual view over data repositories, and to mediate the access to the information stored therein. Typically, in OBDA, the data stored in the databases (DBs) is assumed to provide only an incomplete account of the domain of interest, and the ontology can be used to overcome such incompleteness. Indeed, an ontology does not only provide the terms through which the information can be accessed, but may also express various forms of constraints over the domain. By means of logical inference over such constraints, new information can be deduced from the explicitly given one, and contribute to the answers to queries posed to the DB through the ontology [8, 9, 12, 18].

ONISW'08, October 30, 2008, Napa Valley, California, USA.

Hence, the OBDA setting is profoundly different from that of ordinary DBs, where information is assumed to be complete, and constraints expressed over the DB schema (e.g., keys and foreign keys) are used only for validating data, but not for query answering. Instead, query answering in OBDA is tightly related to query answering under constraints over incomplete DBs interpreted under the open-world assumption [6, 5]. In both cases, query answering amounts to computing the so-called *certain answers*, i.e., those answers that hold in *all* possible DB states compatible with the data explicitly stored in the DB and with the constraints (in the ontology or in the DB schema).

Recently, the problem of answering queries over ontologies has been intensively investigated under various assumptions on the language used to express the ontology. One line of research has considered very expressive languages, i.e., significant fragments of OWL-DL[1], aiming at determining the decidability of query answering, and establishing its computational complexity [9, 10, 12, 16]. Of special interest has been characterizing the *data complexity* of the problem, i.e., the complexity measured solely in the size of the DB, which typically dominates the size of the intensional level of the ontology. It has been shown that for expressive ontology languages such as $\mathcal{SHIQ}$, answering unions of conjunctive queries (UCQs) is coNP-complete in data complexity [12, 16]; actually, the coNP lower bound already holds for much simpler languages due to the presence of disjunction [11]. This has motivated the interest in simpler ontology languages, for which query answering has low data complexity. Specifically, the *DL-Lite*-family of ontology languages has been identified recently [8, 17], which provides an interesting trade-off between expressive power and data complexity of query answering. On the one hand, *DL-Lite* (and its variants) have sufficient expressive power to capture conceptual modeling languages, such as the Entity-Relationship model or UML Class Diagrams. On the other hand, answering UCQs under the certain answer semantics over a *DL-Lite* ontology is LogSpace in data complexity, and can be reduced to standard query evaluation over the DB storing the extensional information [8, 17]. This makes it possible to leverage on standard relational technology, not only for storing the data, but also for query evaluation, thus achieving full scalability w.r.t. the size of the data.

Regarding the query language, it is worth noticing that most research so far has considered just conjunctive queries (CQs) or unions thereof (UCQs), corresponding to the (`UNION`)`-SELECT-PROJECT-JOIN` fragment of SQL. An excep-

[1] `http://www.w3.org/2007/OWL/`

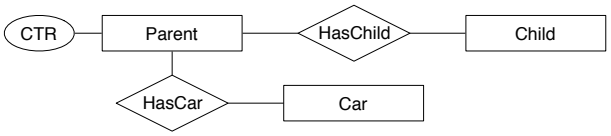

Figure 1: The family ontology.

(a) What is the maximum per child tax reduction explicitly recorded in the data?
(b) How many parents own cars?
(c) What is the total tax reduction of each parent?

Table 1: Example questions involving aggregations.

tion is [7], which allows for "closing" information through the use of an epistemic operator, and then performing first-order operations. However, such kinds of queries do not allow one to express many natural and interesting information requests, such as those expressed by *aggregate queries*, like SQL queries using the aggregation functions `COUNT`, `SUM`, or `AVG`. An important aspect of such queries is that the multiplicities of tuples in the answer set over which the aggregation is performed are of importance[2].

EXAMPLE 1.1. The ontology shown in Figure 1, rendered as an ER-diagram, models (part of) the information regarding tax payments for families, where the HasChild (resp., HasCar) relation associates Parents with their Children (resp., Cars), and CTR is an attribute representing the per-child tax reduction that a parent has right to (e.g., in Italy, this amount depends on the yearly income). Examples of typical questions over such an ontology are shown in Table 1. ∎

There are only few results related to aggregate queries in a setting of incomplete information [3, 4, 14], and none of them addresses properly the recently introduced ontology languages able to capture conceptual data models, such as those of the *DL-Lite* family. The main aim of this paper is to fill this gap, by providing the following contributions:

- We show that standard certain answer semantics is not appropriate in general for aggregate queries or for conditional aggregate queries, where part of the query is used to express a condition that should not account for multiplicities in the answer (see Section 4).
- We introduce *(conditional) epistemic aggregate queries*, and provide for them an alternative, epistemic, semantics that is better suited in the presence of incomplete information (see Section 5).
- For the ontology language *DL-Lite*$_\mathcal{A}$ [17], the most significant representative of the *DL-Lite* family, we investigate conditions on the epistemic aggregate query and the ontology that ensure that certain answers will not necessarily be empty; we develop an algorithm for computing such answers that relies on known algorithms for computing certain answers for non-aggregate queries (see Section 6).

Preliminaries on aggregate queries and on the *DL-Lite*$_\mathcal{A}$ ontology language are given in Sections 2 and 3, respectively.

[2]Note that in queries expressible in the language of [7], multiplicities are not properly taken into account.

2. AGGREGATE QUERIES OVER DBs

In this section we first recall basic notions on conjunctive and aggregate queries in SQL (see also [1] for details). We then recall nested conditional SQL queries, and introduce an extension of rule-based notation to capture such queries.

2.1 Databases and Conjunctive Queries

We assume as given a countably infinite *domain* $\Delta := \Delta_O \cup \Delta_V$, partitioned into a domain Δ_O of object constants and a domain Δ_V of values (containing numbers, e.g., $\mathbb{N}$). We call *tuple* any finite sequence $\bar{c}$ of domain elements. We will denote the empty tuple as ().

A *term* (like t) is either a variable (like v, w, x, y, z) or a constant (like c). A *relation name* R is a predicate symbol of some arity n (a nonnegative integer). A *DB schema* $\mathbf{R}$ is a finite set of relation names. An *atom* is an expression of the form $R(\bar{t})$, where R is a relation name of arity n and $\bar{t}$ is a sequence of n terms. An atom is *ground* when all its terms are constants.

A *condition* ϕ over $\mathbf{R}$ is a conjunction of atoms with relation names from $\mathbf{R}$, written in the form of a list. We denote by $Var(\phi)$ the set of variables occurring in condition ϕ.

A *conjunctive query* (CQ) over $\mathbf{R}$ is an expression

$$q(\bar{x}) \leftarrow \phi,$$

where q is a relation of some arity n that does not occur in $\mathbf{R}$, called *head relation*, and ϕ is over $\mathbf{R}$. The variables in $\bar{x}$ are called *distinguished* and ϕ is called condition of q. Distinguished variables should occur in ϕ. A CQ is *boolean* when the sequence $\bar{x}$ is empty. As usual, we denote a CQ through its head relation.

A *DB instance*, or simply DB, D of $\mathbf{R}$ is a pair $(\Delta, \cdot^D)$ where Δ is the *domain* and $\cdot^D$ is an *interpretation function* over $\mathbf{R}$, that is, a function mapping each relation name R of arity n in $\mathbf{R}$ to a subset R^D of Δ^n, i.e., to a *relation instance*. Observe that a DB is a first order logic interpretation of $\mathbf{R}$. We notice that we allow DBs to be infinite, although we consider only finite representations, by means of an ontology (see Section 3), of (sets of) such possibly infinite DBs.

EXAMPLE 2.1. Consider the DB schema with relations from Figure 1, $\mathbf{R}_f := \{\mathsf{Parent}, \mathsf{CTR}, \mathsf{HasCar}, \mathsf{HasChild}\}$, where f in $\mathbf{R}_f$ stands for "family". A DB D_f of $\mathbf{R}_f$ is

Parent

Name
Luisa
Anna

CTR

PName	Amount
Luisa	100
Anna	150

HasCar

PName	CType
Anna	VwGolf
Anna	FiatUno

HasChild

PName	CName
Anna	Mario
Luisa	Beppe
Anna	Paolo

For convenience, in examples we use attribute names to denote relation positions. ∎

An *assignment* γ over a condition ϕ is a function that maps $Var(\phi)$ to Δ and each constant in ϕ to itself. Assignments are extended to complex syntactic objects like atoms and conditions in the usual way. Whenever $\gamma(\bar{x}) = \bar{c}$ we say that $\bar{x}$ is *bound* to $\bar{c}$. An assignment γ *satisfies* ϕ over a DB

D if, for each atom $R(\bar{t})$ in ϕ, we have $\gamma(\bar{t}) \in R^D$. We denote by $Sat_D(\phi)$ the set of satisfying assignments of ϕ over D.

The *set of answers* q^D of a CQ $q(\bar{x}) \leftarrow \phi$ over a DB D is defined as $q^D := \{\bar{c} \mid \bar{c} = \gamma(\bar{x}),\ \gamma \in Sat_D(\phi)\}$.

2.2 Aggregate Queries

We enrich our syntax by considering now the standard SQL *aggregation functions* *max*, *min*, *count*, *cntd*, *sum*, and *avg*, in the following denoted by α. We call an *aggregation term* any expression of the form *count* or $\alpha(y)$, where y is called an *aggregation variable*. We use $\alpha(\bar{y})$ to denote both cases.

An *aggregate query* (AQ) over $\mathbf{R}$ is a query of the form:

$$q(\bar{x}, \alpha(\bar{y})) \leftarrow \phi,$$

where $\bar{x}$ is a (possibly empty) sequence of *grouping variables*, $\alpha(\bar{y})$ is an aggregation term and ϕ is a condition containing $\bar{x}$ and $\bar{y}$. Moreover, $\bar{y}$ does not occur in $\bar{x}$.

EXAMPLE 2.2. Question (a) in Table 1 gives rise to the following *max*-query q_1 over $\mathbf{R}_f$ from Example 2.1:

$$q_1(max(y)) \leftarrow \mathsf{Parent}(x), \mathsf{CTR}(x, y).$$

The same query in SQL notation is:

```
SELECT MAX(CTR.Amount)
FROM   Parent, CTR
WHERE  Parent.Name = CTR.PName
```

There is no `GROUP BY` clause, which reflects the fact that in q_1 there are no grouping variables. ∎

Aggregation functions in SQL are defined over bags $\{\!|\cdot|\!\}$, called *groups*, of symbolic and numerical values and return a number. Let D be a DB and $\bar{c}$ a tuple. Consider an aggregate query $q(\bar{x}, \alpha(\bar{y})) \leftarrow \phi$. The *group* $F_{\bar{c}}$ of tuple $\bar{c}$ is defined as the bag

$$F_{\bar{c}} := \{\!|\, \gamma(\bar{y}) \mid \gamma \in Sat_D(\phi) \text{ and } \gamma(\bar{x}) = \bar{c} \,|\!\}.$$

The *set of answers* q^D of an ACQ q over D is defined as

$$q^D := \{(\bar{c}, \alpha(F_{\bar{c}})) \mid \bar{c} = \gamma(\bar{x}),\ \text{for some } \gamma \in Sat_D(\phi)\}.$$

EXAMPLE 2.3. Consider the *max*-query q_1 of Example 2.2 and the DB D_f of Example 2.1. There are only two satisfying assignments, namely, $\gamma_1 := [x/\texttt{Luisa}, y/100]$ and $\gamma_2 := [x/\texttt{Anna}, y/150]$. Since q_1 has no grouping variables, the only group we have is for the empty tuple $()$ and $F_{()} = \{\!|\, 100, 150 \,|\!\}$. Therefore $q_1^{D_f} = \{(max(F_{()}))\} = \{(max(\{\!|\, 100, 150 \,|\!\}))\} = \{(150)\}$. ∎

2.3 Conditional Aggregate Queries

The standard syntax of aggregate queries can only express some SQL aggregate queries. For instance, Question (b) from Table 1 corresponds to the SQL query (over $\mathbf{R}_f$):

```
SELECT COUNT(*)
FROM   Parent
WHERE  EXISTS (
         SELECT *
         FROM   HasCar
         WHERE  HasCar.PName = Parent.Name)
```

which contains an SQL (sub)query that is *nested* within an `EXISTS` condition. Nested SQL subqueries of this kind do not return bags, but a truth value that will be `true` if and only if there is a tuple satisfying the conditions of the subquery.

To cover this feature, we must extend accordingly the syntax and semantics of aggregate queries. This nesting will be expressed by bracketing atoms in the query's condition.

A *conditional aggregate query* q (CAQ) is a query of the form:

$$q(\bar{x}, \alpha(\bar{y})) \leftarrow \phi, [\psi],$$

where ϕ and ψ are conditions, $\bar{x}, \bar{y} \subseteq Var(\phi)$ and $Var(\phi) \cap Var(\psi) \neq \emptyset$. We call ψ a *nested condition*. As before, we denote queries with their head relation.

We distinguish assignments of $Sat_D(\phi, \psi)$ that give rise to bags of tuples and assignments that do not, i.e., assignments that satisfy nested conditions. We define the latter ones as

$$Sat_D^{\psi}(\phi) := \{\gamma_{|\,Var(\phi)} \mid \gamma \in Sat_D(\phi, \psi)\},$$

where $\gamma_{|\,Var(\phi)}$ denotes the restriction of γ to $Var(\phi)$. The *group* of a tuple $\bar{c}$, denoted $G_{\bar{c}}$, is defined similarly to the group $F_{\bar{c}}$ but using assignments from $Sat_D^{\psi}(\phi)$:

$$G_{\bar{c}} := \{\!|\, \gamma(\bar{y}) \mid \gamma \in Sat_D^{\psi}(\phi) \text{ and } \gamma(\bar{x}) = \bar{c} \,|\!\}.$$

We define *the set of answers* q^D of CAQ q over DB D as

$$q^D := \{(\bar{c}, \alpha(G_{\bar{c}})) \mid \bar{c} = \gamma(\bar{x}), \text{ for some } \gamma \in Sat_D^{\psi}(\phi)\}.$$

EXAMPLE 2.4. With the new syntax we can express Question (b) from Table 1 as the following CAQ over $\mathbf{R}_f$:

$$q_2(count) \leftarrow \mathsf{Parent}(x), [\mathsf{HasCar}(x, y)].$$

For the DB D_f from Example 2.1 we obtain $q_2^{D_f} = \{(2)\}$. ∎

3. DL-LITE$_A$ ONTOLOGIES

We present now the ontology language on which we base our considerations in the rest of the paper, namely the Description Logic *DL-Lite*$_{\mathcal{A}}$. This language is one of the most expressive variants of Description Logics (DLs) of the *DL-Lite* family [8], which is a family of DLs that have been developed specifically to handle large amounts of data, possibly stored in relational DBs [17]. *DL-Lite*$_{\mathcal{A}}$ is also one of the tractable fragments of the upcoming standard OWL 2[3]. As usual in DLs, in *DL-Lite*$_{\mathcal{A}}$ the universe of discourse is represented in terms of *concepts*, denoting sets of objects, and *roles*, denoting binary relations between objects. Additionally, *DL-Lite*$_{\mathcal{A}}$ provides mechanisms to deal both with abstract objects (which are instances of concepts) and with data values (such as strings, integers, ...): *value-domains* denote sets of values, while (concept) *attributes*[4] denote binary relations between objects and values. The distinction between objects and values is especially important in the OBDA setting, where one needs to deal with the fact that (abstract) objects are maintained at the level of the ontology, while data values are stored in the underlying DB.

[3] `http://www.w3.org/2007/OWL/`

[4] For simplicity, we do not consider role attributes here.

3.1 The DL-Lite$_A$ Language

In providing the specification of *DL-Lite*$_\mathcal{A}$, we use the following notation:

- A denotes an *atomic concept*, B a *basic concept*, C a *general concept*, and $\top_C$ the *universal concept*.
- E denotes a basic value-domain, i.e., the range of an attribute, F a *value-domain expression*, and $\top_D$ the *universal value-domain*.
- P denotes an *atomic role*, Q a *basic role*, and R a *general role*. An atomic role is simply a role denoted by a name.
- U denotes an *atomic attribute* (or simply attribute), and V a *general attribute*.

An atomic concept or an atomic value-domain is simply a unary relation name, while an atomic role or an atomic attribute is simply a binary relation name. The syntax of *basic* and *general* concepts, value-domains, roles, and attributes is given below. Given an attribute U, we call *domain* of U, denoted by $\delta(U)$, the set of objects that U relates to values, and we call *range* of U, denoted by $\rho(U)$, the set of values that U relates to objects. Note that $\delta(U)$ is a concept, whereas $\rho(U)$ is a value-domain.

Formally, the syntax of *DL-Lite*$_\mathcal{A}$ expressions is defined as follows:

$$\begin{array}{lcl} B & ::= & A \mid \exists Q \mid \delta(U) \\ C & ::= & \top_C \mid B \mid \neg B \mid \exists Q.C \\ E & ::= & \rho(U) \\ F & ::= & \top_D \mid T_1 \mid \cdots \mid T_n \\ \\ Q & ::= & P \mid P^- \\ R & ::= & Q \mid \neg Q \\ V & ::= & U \mid \neg U \end{array}$$

The semantics of *DL-Lite*$_\mathcal{A}$ is specified, as usual in DLs, in terms of first-order logic interpretations, i.e., DB instances, over the domain $\Delta = \Delta_O \cup \Delta_V$ (cf. Section 2). All such DB instances agree on the semantics assigned to each value-domain T_i. In particular, we assume that Δ_V coincides with the union of the interpretations of all T_i. Given a DB instance $D = (\Delta, \cdot^D)$ interpreting unary and binary atomic relation symbols, the interpretation is extended to complex expressions in the usual way. For lack of space we don't provide here the details of the semantics of the constructs, and refer instead to [17].

3.2 Knowledge Bases

A *DL-Lite*$_\mathcal{A}$ *knowledge base* (KB), or ontology, $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is constituted by two components: a TBox $\mathcal{T}$, representing intensional knowledge, and an ABox $\mathcal{A}$, representing extensional information. The *TBox* is constituted by a set of assertions of the form:

B	$\sqsubseteq$ C	*(concept inclusion assertion)*
Q	$\sqsubseteq$ R	*(role inclusion assertion)*
E	$\sqsubseteq$ F	*(value-domain inclusion assertion)*
U	$\sqsubseteq$ V	*(attribute inclusion assertion)*
(funct Q)		*(role functionality assertion)*
(funct U)		*(attribute functionality assertion)*

A concept inclusion assertion expresses that a (basic) concept B is subsumed by a (general) concept C. Analogously for the other types of inclusion assertions. A role functionality assertion expresses the (global) functionality of an atomic role. Analogously for an attribute functionality assertion.

Formally, a DB instance D satisfies an inclusion assertion $X \sqsubseteq Y$ if the corresponding set inclusion $X^D \subseteq Y^D$ holds, and it satisfies a functionality assertion (funct Q) (resp., (funct U)), if the relation Q^D (resp., U^D) is a function.

A *DL-Lite*$_\mathcal{A}$ ABox is a finite set of *membership assertions* of the form

$$A(a), \qquad D(c), \qquad P(a,b), \qquad U(a,c)$$

where a and b are object constants in Δ_O, and c is a value in Δ_V[5]. To define its semantics, we specify when a DB instance $D = (\Delta, \cdot^D)$ satisfies a membership assertion α in $\mathcal{A}$, written $D \models \alpha$. Namely, D satisfies $A(a)$ if $a \in A^D$, satisfies $P(a,b)$ if $(a,b) \in P^D$, and satisfies $U(a,b)$ if $(a,b) \in U^D$.

D is a *model* of a *DL-Lite*$_\mathcal{A}$ KB $\mathcal{K}$ or, equivalently, D *satisfies* $\mathcal{K}$, written $D \models \mathcal{K}$ iff D satisfies all assertions in $\mathcal{K}$. A KB is *satisfiable* if it has at least one model. A KB $\mathcal{K}$ *logically implies* an assertion α if all models of $\mathcal{K}$ are also models of α. (Similar definitions hold for a TBox $\mathcal{T}$ or ABox $\mathcal{A}$.)

3.3 Query Answering in DL-Lite$_A$

A query over a KB $\mathcal{K}$ is a query whose predicates are the *atomic* concepts, value domains, roles, and attributes of $\mathcal{K}$. The reasoning service we are interested in is *query answering*: given $\mathcal{K}$ and a query $q(\bar{x})$ over $\mathcal{K}$, return the *certain answers* $Cert(q, \mathcal{K})$ to $q(\bar{x})$ over $\mathcal{K}$, i.e., all tuples $\bar{c}$ of elements of $\Delta_O \cup \Delta_V$ s.t. $\bar{c} \in q^D$ for every model D of $\mathcal{K}$.

Note that query answering over an ontology is a form of reasoning over incomplete DBs, and as such, in general, a more complex task than plain query answering over a DB. Indeed, from the results in [8] it follows that, in general, already answering CQs over *DL-Lite*$_\mathcal{A}$ KBs as introduced above, is PTIME-hard in *data complexity* (i.e., the complexity measured w.r.t. the size of the ABox only). As a consequence, to solve query answering over such KBs, we need at least the power of general recursive Datalog.

However, as shown in [8], the data complexity of answering CQs (and unions thereof) in *DL-Lite*$_\mathcal{A}$ drops to LOGSPACE, provided a suitable restriction is imposed on the interaction between role (resp., attribute) inclusion assertions and functionality assertions. Intuitively, the restriction requires that no functional role or attribute can be specialized, i.e., appears in the left-hand side of an inclusion assertion, when in the right-hand side there is a non-negated role or attribute (we refer to [8] for the details). Moreover, if this condition is satisfied, computing the certain answers of a CQ with respect to a satisfiable *DL-Lite*$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ can be reduced, through a process called *perfect reformulation*, to the evaluation over $\mathcal{A}$ (now viewed as a complete DB) of a suitable union of CQs [8]. This query answering technique has also been implemented in the QuOnto system [2], where the ABox is stored directly in a relational DB, containing one relation for each atomic symbol. In the following, we will also assume that the ABox of the ontology over which queries are answered is stored in the form of DB relations, one for each atomic concept, attribute, and role symbol. Note that domains need not be stored explicitly in the DB, since they are actually used only to represent the range of attributes.

[5]We assume to have *standard names*, i.e., we identify constants with domain elements.

4. LIMITATIONS OF CERTAIN ANSWER SEMANTICS

As we discussed in the previous section, the problem of answering queries over ontologies is a special case of the one of answering queries over sets of DBs. This problem has not yet been studied systematically for aggregate queries.

Lechtenbörger et al. [14] investigated aggregate queries over conditional tables, which are a formalism to specify incomplete DBs, and showed that in this case the answers to an aggregate query can be represented again by a conditional table. Arenas et al. [4] studied aggregate queries over the set of repairs of an inconsistent DB. They introduced a so-called *range-semantics*, according to which a query is evaluated over each individual instance (that is, each repair), and returns the minimal and the maximal value thus obtained. Afrati and Kolaitis [3] gave a semantics and algorithms for aggregate queries in data exchange, which crucially exploit the specific characteristics of this setting.

As a preparation for the following sections, we will argue that an approach that generalizes the standard certain answer semantics to aggregate queries will not be satisfactory, since in most cases the set of certain answers will be empty.

EXAMPLE 4.1. Consider the KB $\mathcal{K}_1$ that has an empty TBox and whose ABox is D_f from Example 2.1. Because of the open-world semantics of ontologies, $\mathcal{K}_1$ expresses that, for instance, `Luisa` has at least two children, `Beppe` and `Paolo`, and that it is unclear whether she has a car.

Recall Question (c) from Table 1. It can be expressed as

$$q_3(x, sum(y)) \leftarrow \mathsf{CTR}(x, y), \mathsf{HasChild}(x, z). \qquad (1)$$

In SQL syntax, the query is

```
SELECT   CTR.PName, SUM(CTR.Amount)
FROM     CTR, HasChild
WHERE    CTR.PName = HasChild.PName
GROUP BY CTR.PName
```

The certain answer semantics for queries over a KB requires one to compute the intersection of the answer sets over all possible DBs (models) of the KB. Since $\mathcal{K}_1$ does not "know" all children of, e.g., `Luisa`, in different models of $\mathcal{K}_1$ the numbers of her children differ and, consequently, the total tax reductions differ. Therefore, the set of certain answers $Cert(q_3, \mathcal{K}_1)$ is empty. ∎

The proposition below generalizes the example to KBs where either the Tbox or ABox are empty.

PROPOSITION 4.2. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a KB and q an aggregate query. Then $Cert(q, \mathcal{K}) = \emptyset$ if $\mathcal{T} = \emptyset$ or if $\mathcal{A} = \emptyset$.*

Clearly, $Cert(q, \mathcal{K}) = \emptyset$ holds not only in the two extreme cases of the proposition. It holds, intuitively, whenever the TBox does not restrict the instances extending an ABox so much that at least one group has the same aggregate value in each instance.

5. EPISTEMIC AGGREGATE QUERIES OVER ONTOLOGIES

We now introduce *epistemic aggregate queries* and give intuitions why they solve the problems with the certain answers semantics discussed above.

5.1 Motivation

EXAMPLE 5.1. Reconsidering Example 4.1, we notice that certain answer semantics does not exploit the fact that in every instance D that is a model of $\mathcal{K}_1$, `Luisa` has two children for sure, namely `Mario` and `Paolo` named in the ABox, and `Anna` has one child for sure, namely `Beppe`. Consequently, `Luisa` will receive a total tax reduction of 300 EUR for `Mario` and `Paolo` and Anna one of 100 EUR for `Beppe`. Clearly, in a specific DB instance, the total reduction for `Anna` or `Luisa` may be higher, since the DB instance may contain additional children to the ones explicitly listed in the ABox. If we could tell our query engine that we only want the sum of the known reductions for the known children, then the engine could return as answers to query q_3 the set $\{(\texttt{Luisa}, 300), (\texttt{Anna}, 100)\}$.

Note that in this case the answers to the query could be obtained by first computing the certain answers of the conjunctive query

$$q_3'(x, y, z) \leftarrow \mathsf{CTR}(x, y), \mathsf{HasChild}(x, z),$$

then grouping the result according to values of x, and finally computing the aggregate $sum(y)$ for each group (projecting away z). ∎

The next example shows this semantics is too restrictive.

EXAMPLE 5.2. Consider a KB $\mathcal{K}_2$, with the following TBox:

$$\exists \mathsf{CTR} \sqsubseteq \exists \mathsf{HasChild}, \qquad (\mathsf{funct\ HasChild}), \qquad (\mathsf{funct\ CTR}),$$

and consider a DB instance D for $\mathcal{K}_2$. The first constraint is satisfied by D if every Parent, with an associated value for the (optional) CTR attribute, has also a child in D. The second constraint is satisfied by D if every Parent has at most one child in D. The last constraint says that the CTR attribute is functional. One could imagine that DBs satisfying the TBox reflect the situation of a country with a birth control policy where only one child would give the parents a tax reduction. Consequently, a DB instance may contain at most one child for each parent.

Suppose the ABox of $\mathcal{K}_2$ is as follows:

CTR

PName	Amount
Luisa	150
Anna	100

HasChild

PName	CName
Anna	Mario

According to the KB, there is one known child of `Anna`, namely `Mario`, but no known child of `Luisa`. However, due to the first constraint, it is known that `Luisa` has a child, although it is not known who that child is.

If we want to identify the tax reductions that necessarily hold for every DB instance that is a model of $\mathcal{K}_2$, it is not sufficient to consider only the known children, as in the previous example, but also which children are known to exist for each person.

Reasoning with the information in the ABox and in the TBox, a query processor could determine that `Luisa` receives a total reduction of 150 EUR and `Anna` one of 100 EUR. Actually, in this case $\{(\texttt{Luisa}, 150), (\texttt{Anna}, 100)\}$ is the set of certain answers to the query q_3 over the KB $\mathcal{K}_2$. ∎

To summarize the intuitions of the semantics for aggregate queries we presented in the examples, in Example 5.1, we (1) calculated certain answers for the query q_3', the body of which contains the same condition as q_3, and the distinguished variables of which are $Var(q_3)$ and (2) aggregated over the certain answers. In Example 5.2, we (1) used certain answers for the grouping variables $\bar{x}$ and the aggregation variable y and (2) applied reasoning over the ontology for the variable z, which varies over the children of tax reduction holders, in order to perform the aggregation.

5.2 Epistemic Aggregate Queries

In order to capture both of the above cases, we introduce epistemic aggregate queries. An *epistemic aggregate query* (EAQ) is a query of the form:

$$q(\bar{x}, \alpha(\bar{y})) \leftarrow \mathbf{K}\bar{x}, \bar{y}, \bar{z}.\ \phi, [\psi], \tag{2}$$

where $\bar{z}$ is a possibly empty list of distinct existential variables of ϕ and ψ such that $\bar{z}$ is disjoint from $\bar{x}$ and $\bar{y}$.

We call $\mathbf{K}$ an *epistemic operator* (where the letter "K" stands for "known") and the variables in the list following $\mathbf{K}$ are the *epistemic* variables, or $\mathbf{K}$-variables, of the query.

Example 5.3. We introduce query q_4 as an epistemic variant of query q_3 in (1)

$$q_4(x, sum(y)) \leftarrow \mathbf{K}x, y.\ \mathsf{CTR}(x, y), \mathsf{HasChild}(x, z),$$

Note that this query has no nested conditions and the only epistemic variables are x and y. ∎

5.3 Semantics of EAQs over Ontologies

We define the semantics of epistemic aggregate queries for arbitrary KBs.

Consider an EAQ q as in (2) and a KB $\mathcal{K}$. Let $\bar{w} := \bar{x} \cup \bar{y} \cup \bar{z}$. A tuple of constants $\bar{c}$ is a *known solution* for $\bar{w}$ in ϕ, ψ over $\mathcal{K}$ if $\bar{c}$ is a certain answer for the query

$$aux_q(\bar{w}) \leftarrow \phi, \psi \tag{3}$$

over $\mathcal{K}$. The semantics of q over $\mathcal{K}$ will be defined in such a way that, for every model D of $\mathcal{K}$, the query q is evaluated considering only assignments that map $\bar{w}$ to known solutions. The epistemic certain answers for q over D are then obtained by taking the intersection of the answer sets for all models D of $\mathcal{K}$.

Formally, let $\bar{w} := \bar{x} \cup \bar{y} \cup \bar{z}$ and $\bar{v} := \bar{w} \cap Var(\phi)$, that is, $\bar{w}$ consists of all epistemic variables of q, and $\bar{v}$ of all epistemic variables of ϕ. For a model D of $\mathcal{K}$ we define the set $KSat_{D,\mathcal{K}}(\bar{w}; \phi, \psi)$ of *satisfying* $\mathbf{K}$*-assignments* of ϕ, ψ over D as

$$KSat_{D,\mathcal{K}}(\bar{w}; \phi, \psi) := \{\gamma \in Sat_D(\phi, \psi) \mid \gamma(\bar{w}) \in Cert(aux_q, \mathcal{K})\}.$$

Note that all satisfying $\mathbf{K}$-assignments map $\bar{w}$ to known solutions. Analogously to the standard semantics, we define the set $KSat^{\psi}_{D,\mathcal{K}}(\bar{v}; \phi)$ of all *satisfying* $\mathbf{K}$*-assignments* of ϕ over D *that respect* ψ as

$$KSat^{\psi}_{D,\mathcal{K}}(\bar{v}; \phi) := \{\gamma_{|_{Var(\phi)}} \mid \gamma \in KSat_{D,\mathcal{K}}(\bar{w}; \phi, \psi)\}.$$

We define the multiset $H_{\bar{d}}$, which we call $\bar{d}$*-group*, consisting of the satisfying assignments from $KSat^{\psi}_{D,\mathcal{K}}(\bar{z}; \phi)$ as follows:

$$H_{\bar{d}} := \{\!\{\gamma(y) \mid \gamma \in KSat^{\psi}_{D,\mathcal{K}}(\bar{z}; \phi) \text{ and } \gamma(\bar{x}) = \bar{d}\}\!\}. \tag{4}$$

We say that $(\bar{d}, d)$ is a $\mathbf{K}$*-answer* for an EAQ q over D if there is $\gamma \in KSat^{\psi}_{D,\mathcal{K}}(\bar{v}; \phi)$ such that

$$\bar{d} = \gamma(\bar{x}) \qquad \text{and} \qquad d = \alpha(H_{\bar{d}}).$$

We denote the set of *all* $\mathbf{K}$-answers for q over a model D of $\mathcal{K}$ as $q(D, \mathcal{K})$.

We define *epistemic certain answers* for an EAQ q over a knowledge base $\mathcal{K}$, denoted $ECert(q, \mathcal{K})$, as follows:

$$ECert(q, \mathcal{K}) := \bigcap_{D \in \mathcal{K}} q(D, \mathcal{K}).$$

Example 5.4. Consider again the epistemic *sum*-query q_4 from the knowledge base represented by $\mathcal{K}_1$ as in Example 4.1. The certain answers $Cert(q_3', \mathcal{K}_1)$ are

$$\{(\texttt{Luisa}, 100, \texttt{Beppe}), (\texttt{Anna}, 150, \texttt{Mario}), (\texttt{Anna}, 150, \texttt{Paolo})\}.$$

Then $KSat_{D,\mathcal{K}}(\bar{w}; \phi, \psi)$ is $\{\gamma_1, \gamma_2, \gamma_3\}$ where

$$\begin{array}{lcl} \gamma_1 & := & [x/\texttt{Luisa}, y/100, z/\texttt{Beppe}], \\ \gamma_2 & := & [x/\texttt{Anna}, y/150, z/\texttt{Mario}], \\ \gamma_3 & := & [x/\texttt{Anna}, y/150, z/\texttt{Paolo}]. \end{array}$$

Since x is q_4's grouping variable, this means we get two $\mathbf{K}$-groups, the group of `Luisa` and the group of `Anna`, namely, $G_{\texttt{Luisa}} = \{\!\{\, 100 \,\}\!\}$ and $G_{\texttt{Anna}} = \{\!\{\, 150, 150 \,\}\!\}$. Recall that *sum* returns the sum of the values of the groups. Therefore,

$$ECert(q_4, \mathcal{K}_1) = \{(\texttt{Luisa}, 100), (\texttt{Anna}, 300)\}.$$ ∎

6. EVALUATING EAQs OVER DL-LITE$_{\mathrm{A}}$ ONTOLOGIES

As we discussed in Section 4, with the classical approach, for an ACQ q and TBox $\mathcal{T}$, the set of certain answers $Cert(q, \langle \mathcal{T}, \mathcal{A} \rangle)$ may be empty for every ABox $\mathcal{A}$. The same can happen for EAQs. We say that an EAQ q is *trivial* for a TBox $\mathcal{T}$ if for all ABoxes $\mathcal{A}$ it holds that $ECert(q, \langle \mathcal{T}, \mathcal{A} \rangle) = \emptyset$. We say that a CAQ of the form (2) is *coherent with* $\mathcal{T}$ if the condition ϕ, ψ is satisfiable together with $\mathcal{T}$. Obviously, every query that is non-coherent with $\mathcal{T}$ is trivial for $\mathcal{T}$. In this section, we consider only coherent pairs of queries and *DL-Lite*$_{\mathcal{A}}$ TBoxes, and identify conditions under which these queries are non-trivial for the TBox. We also present algorithms for computing epistemic certain answers for such queries.

In the following we consider a generic EAQ q of form (2).

6.1 General Algorithm

We introduce now an algorithm that we will use as the basis for computing epistemic certain answers $ECert(q, \mathcal{K})$ for the various aggregation functions α. We denote the algorithm `GA` (for General Algorithm). `GA` takes as input an EAQ q and a KB $\mathcal{K}$, and computes a set of tuples $O = \texttt{GA}(q, \mathcal{K})$.

We denote the sequence of non-distinguished epistemic variables of ϕ as $\bar{z}^{\phi}$. In the algorithm, we denote a predicate (view) of arity $|\bar{x} \cup \bar{y} \cup \bar{z}|$ that "stores" the tuples of certain answers $Cert(aux_q, \mathcal{K})$ for aux_q defined in (3) as $\mathsf{Cert}(aux_q, \mathcal{K})$, consequently, $\mathsf{Cert}(aux_q, \mathcal{K})(\bar{x}, \bar{y}, \bar{z})$ is an atom with this predicate name. We also denote as $\mathtt{q}_0$ (resp., $\mathtt{q}_1$) a predicate (view) that "stores" outputs of q_0 (resp., q_1).

The algorithm `GA` is defined in Figure 2 and it basically applies two steps on the tuples in $Cert(aux_q, \mathcal{K})$: *(i)* it projects them on components corresponding to epistemic variables of ϕ and *(ii)* it aggregates over the result.

Input:	epistemic aggregate query	q
	DL-Lite$_\mathcal{A}$ knowledge base	$\mathcal{K}$
Output:	set of tuples	O
Do:	build q_0:	$\mathsf{q}_0(\bar{x}, \bar{y}, \bar{z}^\phi) \leftarrow \mathsf{Cert}(aux_q, \mathcal{K})(\bar{x}, \bar{y}, \bar{z})$
	build q_1:	$\mathsf{q}_1(\bar{x}, \alpha(y)) \leftarrow \mathsf{q}_0(\bar{x}, \bar{y}, \bar{z}^\phi)$
	compute certain answers:	$D_1 := Cert(aux_q, \mathcal{K})$
	project on **K**-variables:	$D_2 := \mathsf{q}_0^{D_1}$
	aggregate:	$O := \mathsf{q}_1^{D_2}$

Figure 2: General algorithm

Notice that GA projects out non-epistemic variables of $Var(\phi)$ and, consequently, GA loses multiplicity of their values. This may obviously lead to an aggregation that is wrong according to the epistemic certain answers semantics. In the following we show that this does not lead to wrong aggregation if (1) the function α is not sensitive to multiplicities or (2) all the projected out variables do not contribute to multiplicities due to involvement in functional dependencies.

6.2 Queries with Restricted Variables

Let v and w be variables from $Var(q)$ and $\mathcal{T}$ a *DL-Lite*$_\mathcal{A}$ TBox. We say that v *directly (functionally) depends* on w in $\mathcal{T}$ if either

- there is $R(w, v)$ in q and (funct R) is in $\mathcal{T}$, or
- there is $R(v, w)$ in q and (funct R^-) is in $\mathcal{T}$.

We say that v *(functionally) depends* on w in $\mathcal{T}$ if there is a sequence of variables $v = v_1, \ldots, v_n = w$, with each $v_i \in Var(q)$, such that each v_i directly depends on v_{i+1}. Moreover, the variable v *is in bijection with* w in $\mathcal{T}$ if v depends on w and w depends on v in $\mathcal{T}$.

We say that a variable in ϕ is *restricted* by $\mathcal{T}$ if (1) it is epistemic or (2) depends in $\mathcal{T}$ on an epistemic variable occurring in ϕ or (3) is in bijection in $\mathcal{T}$ with an epistemic variable occurring in ψ. Finally, an epistemic aggregate query q is *restricted* by $\mathcal{T}$ if all the variables in ϕ are restricted by $\mathcal{T}$. The intuition is that in a restricted query, non-epistemic variables cannot give rise to additional multiplicities of group elements.

The following theorem states that GA computes *ECert* for restricted queries.

THEOREM 6.1 (RESTRICTED QUERIES). *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{A}$ TBox and q a coherent epistemic aggregate query for $\mathcal{T}$. If q is restricted by $\mathcal{T}$, then*

- *q is not trivial for $\mathcal{T}$, and*
- *$ECert(q, \langle\mathcal{T}, \mathcal{A}\rangle) = \mathtt{GA}(q, \langle\mathcal{T}, \mathcal{A}\rangle)$, for every ABox $\mathcal{A}$.*

We conjecture that an epistemic *count*-query is non-trivial if and only if it is restricted, that is, Theorem 6.1 gives a complete characterization. For other aggregation functions, there are further possibilities for queries to be non-trivial.

6.3 Min, Max, and Count Distinct Queries

GA computes all epistemic certain answers for *min*-, *max*- and *cntd*-queries even if they are non-restricted.

THEOREM 6.2 (MIN, MAX, COUNT DISTINCT QUERIES). *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{A}$ TBox and q a coherent epistemic min-, max- or cntd-query for $\mathcal{T}$. Then*

- *q is not trivial for $\mathcal{T}$ and*
- *$ECert(q, \langle\mathcal{T}, \mathcal{A}\rangle) = \mathtt{GA}(q, \langle\mathcal{T}, \mathcal{A}\rangle)$, for every $\mathcal{A}$.*

In fact, a more general statement holds, that is, $ECert(q, \mathcal{K}) = \mathtt{GA}(q, \mathcal{K})$ holds for every KB $\mathcal{K}$ and every EAC q with α insensitive to multiplicities of the values for y.

6.4 Sum Queries

A non-restricted *sum*-query may return an answer over some ABox, however, in this case all answers have the form $(\bar{d}, 0)$ and all values in the group $H_{\bar{d}}$ (cf. Equation 4) are 0. For a given q and $\langle\mathcal{T}, \mathcal{A}\rangle$, existence of such a group $H_{\bar{d}}$ can be found out by the following two queries. The first query $trash(\bar{x})$ accumulates grouping values of groups that contain non-zero values for y:

$$trash(\bar{x}) \leftarrow \mathsf{Cert}(aux_q, \mathcal{K})(\bar{x}, y, \bar{z}), y \neq 0.$$

The second query $ans(\bar{x}, 0)$ drops the values returned by *trash* from the set of all grouping values in $Cert(aux_q, \mathcal{K})$:

$$ans(\bar{x}, 0) \leftarrow \mathsf{Cert}(aux_q, \mathcal{K})(\bar{x}, y, \bar{z}), \quad \texttt{not } trash(\bar{x}). \qquad (5)$$

In the queries, inequality "$\neq$" and negation not intuitively work as "$\neq$" and MINUS in SQL (see [1] for details). The following theorem shows how to compute $ECert(q, \mathcal{K})$ for non-restricted epistemic *sum*-queries.

THEOREM 6.3 (SUM QUERIES). *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{A}$ TBox and q a coherent epistemic sum-query that is non-restricted for $\mathcal{T}$. Then $ECert(q, \mathcal{K}) = ans^{Cert(aux_q, \mathcal{K})}$, where ans is defined in Equation 5.*

When y varies over non-negative numbers only, the only groups that do not contain non-zero values for y are those for which the sum over y itself is 0. Hence, the computation of $ECert(q, \mathcal{K})$ can be simplified.

PROPOSITION 6.4. *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{A}$ TBox and q a coherent epistemic sum query that is non-restricted for $\mathcal{T}$ with the sum ranging over non-negative numbers only. Then*

$$ECert(q, \mathcal{K}) = \{(\bar{d}, 0) \mid (\bar{d}, 0) \in \mathtt{GA}(q, \mathcal{K})\}.$$

6.5 Average Queries

It turns out that for the epistemic *avg*-queries, the computation of epistemic certain answers is more involved than in all the previous cases. The reason is that the average of several copies of a number k is always k, independently of the number of copies.

Let M be a multi-set and k a natural number. We denote as $k * M$ the multi-set obtained from M by duplicating k times each element in M. For example, if $M = \{\!\{1, 1, 3\}\!\}$ then $2 * M = \{\!\{1, 1, 1, 1, 3, 3\}\!\}$. We denote the average of all the elements occurring in M as $avg(M)$. The following proposition says that *avg* is insensitive for uniform enlargements of multisets.

PROPOSITION 6.5. *Let M be a multi-set. Then*

$$avg(M) = avg(k * M), \text{ for every natural number } k.$$

Let q' be the query that projects the certain answers of the auxiliary query aux_q on the grouping variables, i.e.,

$$q'(\bar{x}) \leftarrow \mathsf{Cert}(aux_q, \mathcal{K})(\bar{x}, \bar{y}, \bar{z}).$$

We say that q has a *uniform grouping* w.r.t. $\mathcal{T}$ if for every ABox $\mathcal{A}$ and every $\bar{d} \in q'^{Cert(ans_q, \langle \mathcal{T}, \mathcal{A} \rangle)}$, there exists a multiset M such that for every DB instance D compatible with $\langle \mathcal{T}, \mathcal{A} \rangle$, there is a natural number k such that $H_{\bar{d}} = k * M$ (where $H_{\bar{d}}$ defined as in Equation (4)).

THEOREM 6.6 (UNIFORM GROUPING). *Let q be a coherent epistemic avg-query and $\mathcal{T}$ a DL-Lite$_\mathcal{A}$ TBox. If q has a uniform grouping w.r.t. $\mathcal{T}$, then q is non-trivial for $\mathcal{T}$.*

We now identify a syntactic conditions on q and $\mathcal{T}$ that guarantees that q has a uniform grouping w.r.t. $\mathcal{T}$, and, hence, is non-trivial for $\mathcal{T}$. We say that an EAQ q is *decomposable* for $\mathcal{T}$ if ϕ can be partitioned into ϕ_1, ϕ_2 such that:

- y occurs only in ϕ_1,
- all variables in ϕ_1 are restricted by $\mathcal{T}$,
- $Var(\phi_1) \cap Var(\phi_2) \subseteq \bar{x}$.

THEOREM 6.7 (DECOMPOSABLE AVERAGE QUERIES). *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{A}$ TBox and q an epistemic avg-query. If q is decomposable for $\mathcal{T}$, then*

- *q has uniform grouping w.r.t. $\mathcal{T}$ and*
- *$ECert(q, \langle \mathcal{T}, \mathcal{A} \rangle) = \texttt{GA}(q, \langle \mathcal{T}, \mathcal{A} \rangle)$, for every ABox $\mathcal{A}$.*

7. CONCLUSIONS AND FUTURE WORK

We have investigated the problem of answering aggregate queries, for which multiplicities in the computed answers need to be taken into account, over ontologies, i.e., in a setting of incomplete information. We have shown that standard certain answer semantics is not appropriate in this setting, and have proposed an alternative semantics based on the notion of epistemic aggregate query (EAQ). For the case of ontologies expressed in *DL-Lite*$_\mathcal{A}$, we have provided several conditions on the form of the EAQ and of the ontology that guarantee that answers exist and can be computed. In all the identified cases, the computation can be reduced to a standard computation of certain answers for non-aggregate queries; hence, certain answers to EAQs ca be computed in LOGSPACE as is the case for CQs over *DL-Lite*$_\mathcal{A}$-knowledge bases. As future work, we plan to extend our results to more expressive ontology languages.

Acknowledgements

This research has been partially supported by FET project TONES (Thinking ONtologiES), funded within the EU 6th Framework Programme under contract FP6-7603, and by the PRIN 2006 project NGS (New Generation Search), funded by MIUR.

8. REFERENCES

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

[2] A. Acciarri, D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, and R. Rosati. QUONTO: QUerying ONTOlogies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 1670–1671, 2005.

[3] F. Afrati and P. Kolaitis. Answering aggregate queries in data exchange. In *Proc. of the 27th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2008)*, 2008.

[4] M. Arenas, L. E. Bertossi, J. Chomicki, X. He, V. Raghavan, and J. Spinrad. Scalar aggregation in inconsistent databases. *Theoretical Computer Science*, 296(3):405–434, 2003.

[5] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. Data integration under integrity constraints. *Information Systems*, 29:147–163, 2004.

[6] A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. of the 22nd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2003)*, pages 260–271, 2003.

[7] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. EQL-Lite: Effective first-order query processing in description logics. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, pages 274–279, 2007.

[8] 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.

[9] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.

[10] D. Calvanese, G. De Giacomo, and M. Lenzerini. Conjunctive query containment and answering under description logics constraints. *ACM Trans. on Computational Logic*, 9(3):22.1–22.31, 2008.

[11] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Deduction in concept languages: From subsumption to instance checking. *J. of Logic and Computation*, 4(4):423–452, 1994.

[12] B. Glimm, I. Horrocks, C. Lutz, and U. Sattler. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. *J. of Artificial Intelligence Research*, 31:151–198, 2008.

[13] J. Heflin and J. Hendler. A portrait of the Semantic Web in action. *IEEE Intelligent Systems*, 16(2):54–59, 2001.

[14] J. Lechtenbörger, H. Shu, and G. Vossen. Aggregate queries over conditional tables. *J. of Intelligent Information Systems*, 19(3):343–362, 2002.

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

[16] M. Ortiz, D. Calvanese, and T. Eiter. Data complexity of query answering in expressive description logics via tableaux. *J. of Automated Reasoning*, 41(1):61–98, 2008.

[17] 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.

[18] R. Rosati. The limits of querying ontologies. In *Proc. of the 11th Int. Conf. on Database Theory (ICDT 2007)*, volume 4353 of *Lecture Notes in Computer Science*, pages 164–178. Springer, 2007.