When OWL met *DL-Lite*...

Claudio Corona, Emma Di Pasquale, Antonella Poggi, Marco Ruzzi, and Domenico Fabio Savo

Dip. di Informatica e Sistemistica, SAPIENZA University of Rome
lastname@dis.uniroma1.it

Abstract. Recent research in the area of ontology representation for the Semantic Web has led to several different language proposals. Among the others, it led to the standardization of OWL, on one hand, and to the emergence of the *DL-Lite* family of Description Logics, on the other hand. These two outcomes mainly differ on their objective: while OWL aims to be a standard, and as such, it is tailored towards expressivity, the main goal of the languages in the *DL-Lite* family is to allow accessing huge amount of data, by maintaining tractability, and delegating query processing to a DBMS. In this paper we show how *DL-Lite* can meet OWL. Specifically, we focus on *DL-Lite*$_A$, the language in the *DL-Lite* family that is closest to OWL, and introduce the *SparSQL* language, a concrete epistemic query language, inspired by both SQL and SPARQL, that allows posing expressive tractable queries over *DL-Lite*$_A$ ontologies. Finally, we introduce the main novel *DL-Lite*$_A$ features beyond OWL. The capability of handling such features, together with the *SparSQL* queries, are some of the new functionalities recently implemented in the MASTRO system.

1 Introduction

Web Ontology Language (OWL) has become the standard W3C language for the definition of ontologies for the Semantic Web community. As a standard language, no limits have been posed to its expressivity, and the complete language of OWL, OWL Full, is not decidable. Some fragments of OWL have been identified in order to meet decidability, with restricted forms of constructs. This is the case of OWL Lite and OWL DL. Although these fragments are decidable, they enforce very high computational costs even for simple reasoning tasks, not allowing a practical use of OWL.

The *DL-Lite* family [3] of Description Logics (DLs) is a family of ontology languages whose aim is to capture some of the most popular conceptual modeling formalisms, such as Entity-Relationship model [6] and UML class diagrams[1], while preserving the tractability of the most important reasoning tasks, such as ontology satisfiability and query answering. More specifically, reasoning over ontologies of the *DL-Lite* family, is LOGSPACE w.r.t. the size of data, that is, the number of instances of the ontology, and can be entirely delegated to a standard DBMS technology. Unfortunately, *DL-Lite* has no standard syntax, which makes it difficult to use the most popular tools currently available for the SW community to reason over *DL-Lite* ontologies.

[1] http://www.omg.org/uml/

In this paper we show how the two aspects mentioned above, that is, the need for a standard language and the capability of handling ontologies with a huge number of instances, can be both achieved. More precisely, the main contributions of this paper can be summarized as follows: (i) we provide an expressive query language for *DL-Lite*$_A$ ontologies, named *SparSQL*, whose syntax is inspired by both SQL and SPARQL, the W3C recommendation towards a standard query language for RDF ; *SparSQL* is actually an epistemic query language that implements the language *EQL-Lite(UCQ)* presented in [2]. As such, it is in LOGSPACE and can be reduced to evaluate first-order logic queries over the ABox; (ii) then, we stress the expressive power of *DL-Lite*$_A$ by extending its TBox, to make it able to express *data properties of object properties*, *denials* and *local identification constraints*, and adding a new component to *DL-Lite*$_A$ ontologies, called the ECBox, containing a general form of constraints called *EQL constraints*, that are based on epistemic logics. Note that this is not the first attempt to add constraints to OWL. However, previous related work, e.g. [7], are technically and semantically incomparable with the approach presented in this paper.

Notably, all the contributions discussed in this work are currently implemented within the MASTRO system, a tool for ontology representation and reasoning that has *DL-Lite*$_A$ as proprietary core language. The main feature of MASTRO is to reduce all reasoning tasks, such as consistency checking and query answering to the evaluation of standard SQL queries over a DBMS. Note that one of the major benefits of using DBMS technologies, is to allow using huge amounts of possibly pre-existing data, to populate the ontology instances [10].

The rest of the paper is structured as follows: in Section 2 we introduce *DL-Lite*$_A$ as a fragment of the standard functional-style syntax of OWL; in Section 3 we present the *SparSQL* query language for *DL-Lite*$_A$ ontologies; in Section 4 we introduce all the new *DL-Lite*$_A$ enrichments that are beyond OWL and present some conclusions in Section 5.

2 Preliminaries

In this section we present *DL-Lite*$_A$ as a fragment of OWL, coherently with its functional-style syntax [2]. Note that *DL-Lite*$_A$ differs from the tractable fragment of OWL recently proposed and called *DL-Lite* [3], essentially because it allows using both functional object properties and sub-object properties, by appropriately restricting their use. In providing the specification of our language, we use A to denote a *named class*, P (possibly with subscripts) an *object property*, U_C a *data property*, D a *datatype*, `owl:Thing` the most general class, and `rdfs:Literal` the universal *datatype*. Then, *DL-Lite*$_A$ expressions are defined as follows:

$objectPropertyExpression ::= P \mid \mathsf{InverseObjectPropertyOf}(P)$
$dataPropertyExpressions ::= U_C$
$classExpression ::= A \mid \mathsf{DataMinCardinality}(1\, U_C) \mid$
$\quad \mathsf{ObjectMinCardinality}(1\, P) \mid$
$\quad \mathsf{ObjectSomeValueFrom}(objectPropertyExpression\ \texttt{owl:Thing}) \mid$
$\quad \mathsf{DataSomeValueFrom}(dataPropertyExpressions\ \texttt{rdfs:Literal})$

[2] From now on we will refer to OWL 2.0, see `http://www.webont.org/owl/1.1/`.
[3] See `http://www.w3.org/Submission/owl11-tractable/#3`

As usual in DLs, a *DL-Lite*$_A$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ represents the domain of discourse in terms of two components: the TBox $\mathcal{T}$, i.e. a set of axioms representing the intensional knowledge, and the ABox $\mathcal{A}$, i.e. a set of axioms representing the extensional knowledge, known as *facts*.

More precisely, *DL-Lite*$_A$ intensional axioms have the form:

$ClassAxiom$ ::= SubClassOf($classExpression$ ($classExpression$ |
ObjectComplementsOf($classExpression$))) |
EquivalentClasses(2* $classExpression$) | DisjointClasses(2* $classExpression$)

$ObjectPropertyAxiom$ ::= SubObjectPropertyOf(2*2 $objectPropertyExpression$) |
EquivalentObjectProperties(2* $objectPropertyExpression$) |
DisjointObjectProperties(2* $objectPropertyExpression$) |
InverseObjectProperties(P_1 P_2) |
ObjectPropertyDomain(P ($classExpression$ |
ObjectComplementsOf($classExpression$))) |
ObjectPropertyRange(P ($classExpression$ |
ObjectComplementsOf($classExpression$)))
FunctionalObjectProperty($objectPropertyExpression$)
SymmetricObjectProperty($objectPropertyExpression$)
AsymmetricObjectProperty($objectPropertyExpression$)

$DataPropertyAxiom$ ::= SubDataPropertyOf(2*2 $dataPropertyExpression$) |
EquivalentDataProperties(2* $dataPropertyExpression$) |
DisjointDataProperties(2* $dataPropertyExpression$) |
DataPropertyDomain($dataPropertyExpression$ ($classExpression$ |
ObjectComplementsOf($classExpression$))) |
DataPropertyRange($dataPropertyExpression$ D) |
FunctionalDataProperty($dataPropertyExpression$)

Then, a *DL-Lite*$_A$ TBox is a set of intensional axioms that must satisfy the following restriction: no functional object property α (or functional data property α') can be specialized, i.e., α (resp. α') cannot appear in axioms of the form:

SubObjectPropertyOf(β α) (resp. SubDataPropertyOf(β' α')).

We now specify the form of a *DL-Lite*$_A$ ABox. Coherently with OWL, *DL-Lite*$_A$ uses two disjoint alphabets, i.e. *individual URIs* to denote objects, and *constants* to denote values (e.g. integer, strings). Then, *DL-Lite*$_A$ ABox axioms have the form:

$fact$ ::= ClassAssertion($individualURI$ A) |
ObjectPropertyAssertion(P 2*2 $individualURI$) |
DataPropertyAssertion(U_C $individualURI$ $constant$)

Before providing an example of *DL-Lite*$_A$ ontology, we recall that the semantics of *DL-Lite*$_A$ is given in terms of FOL interpretations with the unique name assumption. Refer to [9] for details.

Example 1. Consider the following *DL-Lite*$_A$ ontology:

SubClassOf($Female\ Person$)
DisjointClasses($Female\ Male$)
ObjectPropertyDomain(MARRIES $Person$)
SubClassOf($Person$ DataMinCardinality(1 SSN))

SubClassOf($Male\ Person$)
ObjectPropertyRange(MARRIES $Person$)
SymmetricObjectProperty(MARRIES)

ClassAssertion($mary\ Female$)
ClassAssertion($ann\ Female$)
ClassAssertion($bob\ Male$)
ObjectPropertyAssertion(MARRIES $bob\ jane$)

ClassAssertion($john\ Male$)
ClassAssertion($jane\ Female$)
ObjectPropertyAssertion(MARRIES $john\ ann$)
ObjectPropertyAssertion(MARRIES $bob\ mary$)

The intensional level of the ontology asserts that males and females are disjoint sets of persons, where all persons have at least one social security number and can marry other persons. Moreover, it says that if a person x marries a person y , then also y marries x. On the other hand, the extensional level of the ontology asserts that $mary, ann,$ and $jane$ denote female individuals, while $john$ and bob males. Finally, bob marries both $jane$ and $mary$, while $john$ marries ann. □

3 The SPARSQL query language

In this section, after briefly recalling the *EQL-Lite(UCQ)* epistemic query language [2], we introduce the *SparSQL* query language, that implements *EQL-Lite(UCQ)*, and is currently used in MASTRO to pose expressive queries over *DL-Lite*$_A$ ontologies. Then, we present the strategy adopted in MASTRO to process *SparSQL* queries.

3.1 *EQL-Lite(UCQ)*

It is well-known that open-world semantics, typically adopted to interpret DL ontologies, are essential for representing incomplete information, but make FOL queries submitted over DL ontologies undecidable. To the best of our knowledge, the most expressive FOL fragment for which decidability of query answering has been proved in DLs, is the class of union of conjunctive queries (UCQs) [8, 5], which unfortunately, has limited expressive power.

Hence, a non monotonic epistemic query language, named *EQL-Lite($\mathcal{Q}$)*, was introduced in [2] to query arbitrary DL ontologies. Intuitively, this language is based on the idea that "we have complete information on what the ontology knows", which allows to explicitly referring in the query to the ontology knowledge, by adopting a closed-world semantics. Thus, *EQL-Lite($\mathcal{Q}$)* queries are FOL queries, whose atoms, expressed in the *embedded query language* $\mathcal{Q}$, are epistemic formulas that refer to "what the ontology knows", i.e. the *certain answers*. According to the results of [2], *EQL-Lite(UCQ)* is particularly suitable to express complicated queries over *DL-Lite*$_A$ ontologies which can be answered in LOGSPACE w.r.t.the number of instances of the ontology.

3.2 *SparSQL*

SparSQL is the query language currently implementing *EQL-Lite(UCQ)* in MASTRO. Its syntax, as the name of the language suggests, is based both on the SQL STANDARD syntax and the SPARQL syntax [4]. Indeed, in *SparSQL* queries, SQL plays the role of

[4] We refer to `http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/` for details about SPARQL.

FOL in *EQL-Lite(UCQ)*, while a fragment of SPARQL is used as embedded query language, i.e. to define the so-called *sparqltables*. The latter play the role of *EQL-Lite(UCQ)* atoms in *SparSQL* queries.

Thus, before precisely presenting the *SparSQL* queries syntax, let us focus on sparqltables. These are defined as SPARQL queries in the fragment of the SPARQL syntax expressing UCQs. More precisely, such a fragment does not include neither the GraphGraphPattern, nor the FILTER constraints, nor the OptionalGraphPattern. Moreover, for the sake of simplicity, the *SparSQL* syntax does not require the use of the FROM clause in the SPARQL query. Specifically, a non boolean SPARQL query in *SparSQL* has the following syntax:

SELECT $listOfVariables$
WHERE $GroupGraphPattern$

On the contrary, a boolean SPARQL query in *SparSQL* has the following syntax:

ASK $GroupGraphPattern$

Now that we have introduced how sparqltables are defined, we can present the *SparSQL* syntax. A non boolean *SparSQL* query has the following form:

SELECT $listOfAttributesOrExpressions$
FROM $listOfSparqltables$
[WHERE $conditions$]
[GROUPBY $listOfGroupingAttributes$]
[HAVING $listOfGroupingConditions$]
[ORDERBY $listOfAttributesToOrder$]

where each sparqltable (defined in terms of a SPARQL query as described above) is associated to an *alias*, and no joins can occur between sparqltables defined as non boolean SPARQL queries and sparqltables defined as boolean SPARQL queries. As expected, a *SparSQL* query differs from an SQL query mainly because of the presence of one or more sparqltables in the FROM clause. Note that *SparSQL* queries, like SQL queries, can have nested queries that are again *SparSQL* queries. Next we illustrate *SparSQL* by providing two examples of non boolean queries, the first one referring to a sparqltable defined in terms of a non boolean SPARQL query, and the second one to a sparqltable defined in terms of a boolean SPARQL query.

Example 2. Consider the ontology presented in Example 1, and suppose that we want to know the persons that we *know* have married at least two persons. This query can be expressed by the following *SparSQL* query:

SELECT $spouses.x, count(spouses.y)$
FROM SparqlTable(SELECT $?x$ $?y$
WHERE{$?x$:MARRIES $?y$}) $spouses$
GROUPBY $spouses.x$
HAVING $count(spouses.y) >= 2$

The answer to this query would be the tuple $(bob, 2)$.

Now, suppose that we want to know all the *known* wifes, provided we *know* that Bob is married either with Mary or with Ann:

SELECT $wifes.x$
FROM SparqlTable(SELECT $?x$
WHERE{$?x$ rdf:type $'Female'$.
$?x$:MARRIES $?y$}) $wifes$,
SparqlTable(ASK{{$'bob'$:MARRIES $'mary'$}
UNION
{$'bob'$:MARRIES $'ann'$}}) $spouses$

Since the boolean sparqltable conditions are verified, the answers to the query are the tuples (mary), (ann) and (jane). Observe, however, that if the $spouses$ sparqltable would have been false, then the overall *SparSQL* query would have been empty. □

Let us next focus on boolean *SparSQL* queries. A boolean *SparSQL* query has the following syntax[5]:

VERIFY $conditions$

where *conditions* is defined as for non boolean *SparSQL* query. A boolean *SparSQL* query returns *true* if the *conditions* are verified, otherwise it returns *false*.

Next, we provide an example of boolean *SparSQL* query.

Example 3. Consider again the ontology of Example 1, and suppose that we want to know if there is not any person that is not *known* be a male or a female. This query can be expressed by the following boolean *SparSQL* query:

VERIFY $not\ exists$ (SELECT $persons.x$
FROM SparqlTable(SELECT $?x$
WHERE{$?x$ rdf:type $'Person'$}) $persons$
EXCEPT(SELECT $males.x$
FROM SparqlTable(SELECT $?x$
WHERE{$?x$ rdf:type $'Male'$}) $males$
UNION
SELECT $females.x$
FROM SparqlTable(SELECT $?x$
WHERE{$?x$ rdf:type $'Female'$})
$females$))

This query returns *true*, since there are no persons that are not known to be males or females. □

The example above shows that using *SparSQL*, we can express negation (which cannot be expressed in UCQ) through the use of the EXCEPT SQL operator (or NOT IN SQL operator using nested queries) .

3.3 *SparSQL* processing

In this section we sketch out the processing of *SparSQL* queries. The technique exposed below is currently implemented in the *QuOntoEQL* module, that is a client of MASTRO in charge to process *SparSQL* queries.

In particular, *QuOntoEQL* takes as input a SparSQL query q_{EQL} and a *DL-Lite*$_A$ ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ and performs the following steps:

[5] Note that this syntax simplifies the SQL syntax used to express boolean queries.

- every sparqltable γ occurring in the q_{EQL} query is expanded according to the TBox $\mathcal{T}$: the expansion process amounts to compute the perfect reformulation (see [1]) γ_{REF} of the sparqltable;
- the γ_{REF} query is unfolded and rewritten as γ_{UNF}, so to be expressed in terms of the database where the ABox $\mathcal{A}$ is stored (cf. details in [1]);
- every γ sparqltable is then replaced with the corresponding γ_{UNF} query SQL in the original *SparSQL* query.

Hence we obtain a whole query expressed in SQL that can be executed by any DBMS. We refer to [2] for the proof of correctness of the query processing described above.

4 Extending *DL-Lite*$_A$ beyond OWL

This section aims to present new features of *DL-Lite*$_A$, that are absent in OWL. These amount to (i) extend the intensional level of *DL-Lite*$_A$ ontologies, and (ii) introduce a new component of *DL-Lite*$_A$ ontologies, called ECBox (*Epistemic Constraints Box*), embracing a useful expressive kind of epistemic constraints.

4.1 Enriching *DL-Lite*$_A$ intensional level

We next present a series of extensions of *DL-Lite*$_A$ that are beyond OWL and concern the intensional level of ontologies. More precisely, we first introduce *object property data*, that allow handling data properties of object properties. Then, we present *denial* and *local identification constraints*, that were formally introduced in [4, 2].

Object property data To be closer to ER and UML formalisms, *DL-Lite*$_A$ has been enriched with *object property data*, representing binary relations between pairs of objects and values [1].

In the following we use U_R to denote an object property data. Then, the new axioms involving object property data are defined as follows:

$ObjectPropertyDataAxiom$ ::= SubObjectPropertyDataOf(2*2 U_R) |
EquivalentObjectPropertyData(2* U_R) | DisjointObjectPropertyData(2* U_R) |
ObjectPropertyDataDomain(U_R $objectPropertyExpression$) |
ObjectPropertyDataRange(U_R D) | FunctionalObjectPropertyData(U_R)

For the semantics of these axioms refer to [1].

Denial constraints Now, we pass considering the second extension to the intensional level of *DL-Lite*$_A$ ontologies: the *denial constraints*. Denial constraints (for short, DCs) are TBox assertions defined as follows:

$DCAxiom$ ::= deny(q)

where q is a boolean union of conjunctive queries. The semantics of denial constraints is given in terms of FOL interpretations. Specifically, an interpretation $\mathcal{I}$ satisfies the DC deny(q) if q is false in $\mathcal{I}$. Thus, the semantics of ontologies with denial constraints is the set of interpretations that satisfy all assertions in $\mathcal{A}$ and $\mathcal{T}$, including every denial constraint in $\mathcal{T}$.

Local identification constraints Finally, we present the third and last extension: the *local identification constraints*. Local identification constraints (LIdCs) are TBox assertions defined as follows:

$LIdCAxiom$ ::= KeyFor($simplePropertyExpression$ *(, $path$) $classExpression$)

where

$path$ ::= $simplePropertyExpression$ | $testRoleExpression$ | CompositionOf(2*($path$))
$simplePropertyExpression$ ::= $keyDataPropertyExpression$ | $objectPropertyExpression$
$testRoleExpression$ ::= TestRole($classExpression$ | $valueDomainExpression$)
$valueDomainExpression$ ::= D | RangeOf(U_C) | RangeOf(U_R)
$keyDataPropertyExpression$ ::= U_C | InverseDataPropertyOf(U_C)

As for DCs, the semantics of LIdCs is given in terms of FOL interpretations [6]. Intuitively, let KeyFor($\pi_1, ..., \pi_n$ A) be an LIdC, $n \geq 1$, where A denotes a *classExpression*, π_1 denotes a path of length 1, i.e. a $simplePropertyExpression$, and π_i denotes a path of arbitrary length, for $i = 2, ..., n$. Then an interpretation satisfies such an LIdC if for any two individuals a, a' of A, there exists at least one path π_j, $j \in \{1, n\}$, such that a, a' differ on the set of individuals and values that are reachable resp. from a, a' by means of π_j (for more details about the formal semantics of LIdCs, please refer to [4]).

Now, a *DL-Lite*$_A$ ontology with LIdCs is a *DL-Lite*$_A$ ontology whose TBox $\mathcal{T}$ can also include LIdCs, such that for each LIdC γ in $\mathcal{T}$, every $simplePropertyExpression$ in a path of γ is not specialized in $\mathcal{T}$. Then, as for DCs, the semantics of a *DL-Lite*$_A$ ontology with LIdCs is the set of interpretations that satisfy all assertions in $\mathcal{A}$ and $\mathcal{T}$, including every LIdC in $\mathcal{T}$.

Let us now illustrate by an example the extensions presented above.

Example 1 (cont.) The ontology of Example 1 can be now enriched with these axioms:

ObjectPropertyDataDomain($WeddingDate$ MARRIES)
ObjectPropertyDataRange($WeddingDate$ `rdf:date`)
KeyFor(SSN $Person$)
deny($q() \leftarrow WeddingDate(X, X)$)
ObjectPropertyDataAssertion($WeddingDate$ bob $mary$ '06/02/06')

The new intensional assertions state that MARRIES is the domain of the object property data $WeddingDate$, while its range is `date`. Moreover, they state that SSN is an identifier for $Person$, and that nobody is married with himself. Finally, the extensional assertions specify the date of the wedding of bob and $mary$. □

4.2 Expressing epistemic constraints in *DL-Lite*$_A$: the ECBox

In this section we present a new component of a *DL-Lite*$_A$ ontology, called ECBox, consisting of a set of *epistemic constraints*, named *EQL* constraints, formally introduced in [2]. It is worth noting that, as opposed to rules and other kinds of assertions, constraints are not interpreted as assertions allowing inferring the set of models of the ontology. On the contrary, constraints are interpreted as simple "checks" over the ontology set of models.

The syntax for the EQL constraints (or EQLCs) is the following:

[6] Note that OWL 2.0 Easy Keys fundamentally differ from LIdCs, since they are interpreted with an epistemic semantics.

$EQLCAxiom ::= \mathsf{EQLC}(q)$

where q is a boolean *SparSQL* query.

The semantics of such constraints is given in terms of *epistemic interpretations* [2]. An epistemic interpretation E, w satisfies $\mathsf{EQLC}(q)$ if q is true in E, w. Notice that such constraints are a particular form of the constraints introduced in [11].

With this notion in place, we can revise the notion of *DL-Lite*$_A$ ontology: a *DL-Lite*$_A$ *ontology with ECBox* is a triple $(\mathcal{T}, \mathcal{A}, \mathcal{C})$, where $\mathcal{T}$ and $\mathcal{A}$ resp. denote as usual a *DL-Lite*$_A$ TBox and ABox, and $\mathcal{C}$ is a ECBox.

Now, we say that an ontology $(\mathcal{T}, \mathcal{A})$ *satisfies an ECBox* $\mathcal{C}$, if for each $\mathsf{EQLC}(q) \in \mathcal{C}$, the boolean *SparSQL* query q is true over the ontology $(\mathcal{T}, \mathcal{A})$. Then, the semantics of an ontology with ECBox $(\mathcal{T}, \mathcal{A}, \mathcal{C})$ is defined as the set of models of $(\mathcal{T}, \mathcal{A})$, if $(\mathcal{T}, \mathcal{A})$ satisfies $\mathcal{C}$, and the empty set of models, otherwise. Also, we say that an ontology with ECBox is *satisfiable* if its set of models is non-empty. We remind that the idea of epistemic constraints, follows the research line introduced in [11].

Let us now illustrate the role of the ECBox.

Example 1 (cont.) Through the ECBox, we can now enrich the ontology of Example 1 and model the following:

- if someone, let us say X, is married with someone else, let us say Y, and Y is married with X, then their wedding dates have to be the same: $\mathsf{EQLC}(q')$, where q' is the boolean *SparSQL* query asking for the nonexistence of two persons X, Y such that it is *known* that X is married with Y, that Y is married with X, and that their wedding dates are different. For shortness, we don't show the actual form of q'.
- a person is a male or a female: $\mathsf{EQLC}(q'')$, where q'' is the boolean *SparSQL* query showed in Example 3.

Let us focus on the last constraint. It is well-known that *DL-Lite*$_A$, being conceived to keep data complexity of the main reasoning tasks within LOGSPACE, has renounced to express few constructs typically used in the ER (or UML) formalisms, such as complete generalization and minimal cardinality (greater than 1) on relations. Hence, for example, it is not possible to assert in a *DL-Lite*$_A$ ontology that a person is either a male or a female (and nothing else). However, by weakening the semantics and using the last EQLC, it is possible to guarantee that if the ontology with constraints is satisfiable, then a person is either a male or a female, in all the models of the ontology. □

5 Conclusion

The main contribution of this paper is to show how to implement efficient reasoning tasks over ontologies while keeping compliant with OWL. To this aim, we presented a new query language, called *SparSQL*, that is an implementation of the *EQL-Lite(UCQ)* epistemic query language presented in [2], whose syntax is inspired by both SQL and SPARQL. Last but not least, we enriched the ontology language introduced so far with a new set of constructs, beyond OWL, to handle data properties of object properties,

and to express constraints that might be particularly useful for modeling ontologies of practical interest.

It is worth noting that all along this paper we considered *DL-Lite*$_A$ ontologies, having traditional ABoxes as for data layer. Actually, MASTRO is able to access through an ontology and an appropriate set of *mappings* [9] any data layer, provided that it is accessible through a standard SQL engine. Notably, both the functional-style syntax for *DL-Lite*$_A$, the *SparSQL* query language and the expressive *DL-Lite*$_A$ constraints that were presented, are currently implemented within the MASTRO system, and keep working when MASTRO is used to access a general data layer. Furthermore, the first experiments with the overall system are very encouraging.

As future work we plan to follow two main directions. On one hand, we plan to investigate the use of *SparSQL* to query ontologies written in DLs, other than *DL-Lite*$_A$, e.g. OWL-DL, and in RDFS. On the other hand, we plan to run experiments with actual users, in order to compare the usability of MASTRO with the other tools for ontology management reasoning that are currently available.

Acknowledgments. This research has been partially supported by the MIUR FIRB 2005 project "Tecnologie Orientate alla Conoscenza per Aggregazioni di Imprese in Internet" (TOCAI.IT), and by the FET project TONES (Thinking ONtologiES), funded by the EU under contract number FP6-7603.

References

1. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati. Linking data to ontologies: The description logic *DL-Lite*$_A$. In *Proc. of OWLED 2006*, volume 216 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/Vol-216/`, 2006.
2. 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 IJCAI 2007*, pages 274–279, 2007.
3. 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.
4. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Path-based identification constraints in description logics. In *Proc. of KR 2008*, 2008. To appear.
5. D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proc. of PODS'98*, pages 149–158, 1998.
6. P. P. Chen. The Entity-Relationship model: Toward a unified view of data. *ACM Trans. on Database Systems*, 1(1):9–36, Mar. 1976.
7. B. Motik, I. Horrocks, and U. Sattler. Adding integrity constraints to OWL. In *Proceedings of the Workshop OWLED*, 2007.
8. M. M. Ortiz de la Fuente, D. Calvanese, and T. Eiter. Data complexity of answering unions of conjunctive queries in $\mathcal{SHIQ}$. Technical report, Fac. of Computer Science, Free Univ. of Bozen-Bolzano, Mar. 2006. Available at `http://www.inf.unibz.it/~calvanese/papers/orti-calv-eite-TR-2006-03.pdf`.
9. 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.
10. A. Poggi and M. Ruzzi. Ontology-based data access with MASTRO (demo). In *Proceedings of the Workshop OWLED*, 2007.
11. R. Reiter. What should a database know? *J. of Logic Programming*, 14:127–153, 1990.