OwlOntDB: A Scalable Reasoning System for OWL 2 RL Ontologies with Large ABoxes

Rokan Uddin Faruqui and Wendy MacCaull

Centre for Logic and Information
St. Francis Xavier University
Nova Scotia, Canada
{x2010mcd,wmaccaul}@stfx.ca

Abstract. Ontologies are becoming increasingly important in large-scale information systems such as healthcare systems. Ontologies can represent knowledge from clinical guidelines, standards, and practices used in the healthcare sector and may be used to drive decision support systems for healthcare, as well as store data (facts) about patients. Real-life ontologies may get very large (with millions of facts or instances). The effective use of ontologies requires not only a well-designed and well-defined ontology language, but also adequate support from reasoning tools. Main memory-based reasoners are not suitable for reasoning over large ontologies due to the high time and space complexity of their reasoning algorithms. In this paper, we present OwlOntDB, a scalable reasoning system for OWL 2 RL ontologies with a large number of instances, i.e., large ABoxes. We use a logic-based approach to develop the reasoning system by extending the Description Logic Programs (DLP) mapping between OWL 1 ontologies and datalog rules, to accommodate the new features of OWL 2 RL. We first use a standard DL reasoner to create a complete class hierarchy from an OWL 2 RL ontology, and translate each axiom and fact from the ontology to its equivalent datalog rule(s) using the extended DLP mapping. We materialize the ontology to infer implicit knowledge using a novel database-driven forward chaining method, storing asserted and inferred knowledge in a relational database. We evaluate queries using a modified SPARQL-DL API over the relational database. We show our system performs favourably with respect to query evaluation when compared to two main-memory based reasoners on several ontologies with large datasets including a healthcare ontology.

Keywords: Ontology, Knowledge Representation, Healthcare System, Scalable Reasoner, OWL 2 RL

1 Introduction

Ontologies are becoming increasingly important in large-scale information systems such as healthcare systems. Ontologies can represent knowledge from clinical guidelines, standards, and practices used in the healthcare sector and may be used to drive decision support systems for healthcare. Applications for these

types of systems use large ontologies, i.e., ontologies with a large number (millions) of instances. The W3C recommends the use of the Web Ontology Language (OWL), a semantic markup language, which provides a formal syntax and semantics to represent ontologies and paves the way for manipulating ontologies effectively [20]. However, the effective use of ontologies requires not only a well-designed and well-defined ontology language, but also adequate support from reasoning tools. Ontology reasoning is a methodology for extracting and inferring knowledge from ontologies. Description Logic (DL)-based reasoners including *RacerPro*, *FaCT++*, and *Pellet* can efficiently perform reasoning over expressive OWL ontologies. However, these reasoners perform in-memory reasoning and are not particularly suitable for reasoning over ontologies with millions of instances such as those often needed for real-world applications such as healthcare systems.

Several approaches have been applied to improve the scalability of the reasoners. One of the most widely used approaches is database integration, i.e., utilizing secondary memory to increase efficiency. A number of reasoners such as OntMinD [5] and QuOnto [4] use database integration by directly mapping ontologies to databases. In this approach, ontologies are expressed in terms of UML class diagrams or/and ER diagrams and query rewriting techniques are used to perform reasoning over information stored in relational databases [10]. However, this approach restricts the expressivity of ontologies and supports only a small fragment of DL logic called DL-Lite [9]. DL-Lite is the maximal tractable fragment that supports efficient query answering using a relational database. So scalable reasoning with more expressive DL fragments is still a challenging problem. Another approach to improve the scalability of reasoners for more expressive ontologies is the logic programming-based approach. In this approach, an ontology is translated to a logic program, then inference algorithms for logic programs are used for reasoning. The main advantage of this approach is to reuse existing efficient inference algorithms and implementations, which are suitable for large ontologies. Logic programming-based approaches improve the scalability of the reasoning systems by handling large amounts of instances but still restrict the expressivity of ontologies [16].

In this paper, we present a scalable reasoning system, OwlOntDB, for OWL 2 RL ontologies. Here, by scalability, we refer the ability to perform reasoning over ontologies with large numbers of instances. The new standardization, OWL 2, has three profiles: OWL 2 EL - based on the EL^{++} Description Logic, OWL 2 QL - based on the DL-Lite family of Description Logics, and OWL 2 RL - inspired by pD^* and Description Logic Programs (DLP) [12]. Each profile exhibits a polynomial time complexity for ontological reasoning tasks. We choose OWL 2 RL because it offers a great deal of expressivity while being suitable for rule-based implementations. Grosof et al. [12] give a DLP mapping to translate OWL 1 ontologies to datalog programs to take advantage of logic programming-based algorithms to infer knowledge. In our hybrid approach, we extend the DLP mapping to accommodate the new features of OWL 2 RL, combine this with a mapping to a relational database to develop a restrictions checker to han-

dle some OWL 2 RL axioms and concepts that cannot be handled by the logic programming-based approach, and then materialize all asserted and inferred knowledge from an ontology to a relational database. Our approach is a combination of the database mapping and the logic programming-based inferencing. However, instead of using the direct-mapping based approach to map OWL 2 RL ontologies to relational databases as in [10], we used a novel database-driven forward chaining approach to infer and store OWL 2 RL ontologies to relational databases.

The remainder of the paper is organized as follows. In section 2 we describe our scalable reasoning system, OwlOntDB. In section 3 we evaluate the performance of our system using two benchmark ontologies and a real-world ontology for healthcare. We discuss related work in section 4 and conclude in section 5.

2 A Scalable Reasoning System for Large ABoxes: OwlOntDB

We recall that OWL 2 is based on the family of Description Logics (DL) [6], a family of decidable fragments of first order logic. A DL-based ontology has two components: a TBox and an ABox. The TBox introduces vocabulary relevant to a domain and their semantics, while the ABox contains assertions about individuals using this vocabulary. Our reasoning system supports OWL 2 RL, which describes the domain of an ontology in terms of classes, properties, individuals, and datatypes and values. Individual names refer to elements of the domain; classes describe sets of individuals having similar characteristics; properties describe binary relationships between pairs of individuals. A property can be either an object property which links an individual to an individual, or a datatype property which links an individual to a data value. In OWL 2 RL, object properties can be functional, inverse functional, irreflexive, symmetric, asymmetric, or transitive; however, data properties can only be functional [20]. Note that the new features of OWL 2 RL not found in OWL 1 are qualified cardinality restrictions, irreflexive, and antisymmetric properties, and property chain inclusion axioms. The syntax of OWL 2 RL is asymmetric, i.e., the syntactic restrictions allowed for subclass expressions differ from those allowed for superclass expressions. For instance, an existential quantification to a class expression (`ObjectSomeValuesFrom`) is allowed only in subclass expressions whereas universal quantification to a class expression (`ObjectAllValuesFrom`) is allowed only in superclass expressions. These restrictions facilitate the rule-based implementation of reasoning systems for OWL 2 RL ontologies. Note that at present we assume the Unique Name Assumption (UNA) to translate OWL 2 RL ontologies into datalog programs. However, OWL 2 RL does not use the UNA i.e., it does not treat two different OWL 2 RL elements with different names as different. We are currently in the process of removing this limitation.

Ontological reasoning tasks are related either to the TBox, or to the ABox or to both the TBox and the ABox of an ontology. Here we focus on developing a scalable reasoner for reasoning tasks related to the ABox, namely ABox queries

and mixed TBox and ABox queries. We use an existing DL-based reasoner to perform the TBox reasoning necessary to infer the complete subsumption relationship among classes (i.e., generate the class hierarchy). The overview of our system is found in Fig 1. OwlOntDB takes an OWL 2 RL ontology and materializes the datalog version of the classified ontology to the relational database using our technique which we refer to as a database-driven forward chaining and uses a modified SPARQL-DL as a query interface to extract knowledge from the database. The details of each step are explained in the following subsections.

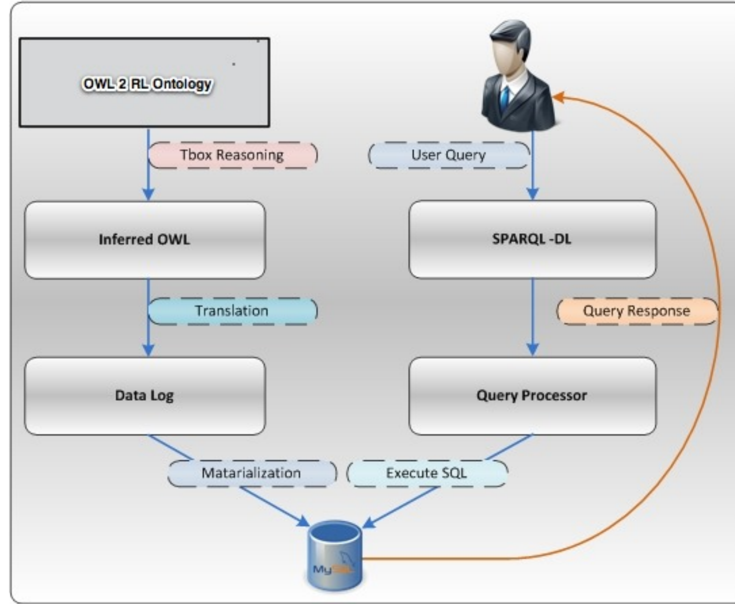

Fig. 1. The system architecture of OwlOntDB

2.1 Translation

Our approach to reasoning is to express inference tasks for the OWL 2 RL ontology in terms of inference tasks for the rule language datalog. Datalog is a simple rule language stemming from Prolog. In this step, we translate the classified ontology to a datalog programs using our extended DLP mapping. We use the OWL API to parse the classified OWL 2 RL ontology and extract all the logical axioms from the ontology. Then, we translate each logical axiom into its equivalent datalog rule(s). In OWL 2 RL, facts are described using ClassAssertions and ObjectPropertyAssertions/DataPropertyAssertions which correspond to DL axioms of the form $a : C$ and $\langle a, b \rangle : P$, respectively, where a and b are individuals, C is a class, and P is an object/data property. These assertions axioms are already in the datalog rule format with empty bodies. Translations of the OWL 2 RL axioms into datalog rules are given in Table 1. Their (straightforward) semantics may be found in [11].

OWL 2 RL Constructors	DL Syntax	Datalog Rule
`ClassAssertions`	$a : C$	$C(a)$
`PropertyAssertion`	$\langle a, b\rangle : P$	$P(a, b)$
`SubClassOf`	$C \sqsubseteq D$	$C(x) \rightarrow D(x)$
`ObjectPropertyChain`	$P \circ Q \sqsubseteq R$	$P(x, y) \wedge Q(y, z) \rightarrow R(x, z)$
`EquivalentClasses`	$C \equiv D$	$C(x) \rightarrow D(x)$, $D(x) \rightarrow C(x)$
`EquivalentProperties`	$P \equiv Q$	$Q(x, y) \rightarrow P(x, y)$
		$P(x, y) \rightarrow Q(x, y)$
`ObjectInverseOf`	$P \equiv Q^-$	$P(x, y) \rightarrow Q(y, x)$
		$Q(y, x) \rightarrow P(x, y)$
`TransitiveObjectProperty`	$P^+ \sqsubseteq P$	$P(x, y) \wedge P(y, z) \rightarrow P(x, z)$
`SymmetricObjectProperty`	$P \equiv P^-$	$P(x, y) \rightarrow P(y, x)$
`Object/DataUnionOf`	$C_1 \sqcup C_2 \sqsubseteq D$	$C_1(x) \rightarrow D(x)$, $C_2(x) \rightarrow D(x)$
`Object/DataIntersectionOf`	$C \sqsubseteq D_1 \sqcap D_2$	$C(x) \rightarrow D_1(x)$,$C(x) \rightarrow D_2(x)$
`Object/DataSomeValuesFrom`	$\exists P.C \sqsubseteq D$	$P(x, y) \wedge C(y) \rightarrow D(x)$
`Object/DataAllValuesFrom`	$C \sqsubseteq \forall P.D$	$C(x) \wedge P(x, y) \rightarrow D(y)$
`Object/DataPropertyDomain`	$\top \sqsubseteq \forall P^-.C$	$P(y, x) \rightarrow C(y)$
`Object/DataPropertyRange`	$\top \sqsubseteq \forall P.C$	$P(x, y) \rightarrow C(y)$

Table 1. Translation of OWL 2 RL axioms into datalog rules

Recall that we translate an ontology to a logic program in order to use the logic programming-based inference algorithm for ontology reasoning. However, we can not handle OWL 2 RL concepts dealing with cardinality restrictions - namely, maximum cardinality and minimum cardinality, and axioms dealing with property restrictions - namely, functional, inverse functional, irreflexive, asymmetric - using a logic programming-based approach. These concepts and axioms impose certain restrictions over the object and data properties of an ontology and any violation of these restrictions results in an inconsistent ABox. We developed a two-phase approach to the translation, using first an automated translator to translate the ontology to datalog and then a restrictions checker to check for ABox consistency with respect to the restriction concepts and axioms. We represent each restriction concept/axiom by a datalog rule and then store the restrictions of a property to a relational database by translating the datalog rule to an SQL statement. For each assertion the restrictions checker checks whether it violates any restrictions. The datalog representations of the restriction concepts and axioms are given in Table 2. We illustrate this with a brief example: Suppose we have a TBox axiom *IrreflexiveObjectProperty(hasSibling)* (*hasSibling* is an irreflexive object property) and then we infer an ABox axiom *hasSibling(Bob, Bob)*. Now the ABox of the ontology will be inconsistent with respect to the TBox axiom because *Bob* cannot be the sibling of himself (*irreflexivity*). We identify all violations according to the semantics of the axioms listed in Table 2, where $n = 0$ or 1.

2.2 Materialization

Materialization [8] is an approach for inferring and storing implicit knowledge from ontologies. If the ABox of an ontology is large and the query rate is high, the

`MinimumCardinality` $\geqslant nP.C$ $ObjectMinCardinality(n\ P\ C)$	`MaximumCardinality` $\leqslant nP.C$ $ObjectMaxCardinality(n\ P\ C)$
`FunctionalProperty` $\top \sqsubseteq \leqslant 1\ P$ $FunctionalObjectProperty(P)$	`InverseFunctionalProperty` $\top \sqsubseteq \leqslant 1\ P^{-}$ $InverseFunctionalObjectProperty(P)$
`Irreflexive` $\exists\ P.self \sqsubseteq \bot$ $IrreflexiveObjectProperty(P)$	`Asymmetric` $P \sqsubseteq \neg P^{-}$ $AsymmetricObjectProperty(P)$

Table 2. Datalog representation of the restrictions checker's concepts and axioms

materialization technique is faster than the approaches that perform reasoning during query evaluation. Materialization techniques are used in many scalable reasoners, including [5], [21] and [18]. In our materialization approach, we use the forward-chaining method to infer implicit knowledge and a relational database to store information. In this section, we give a formal representation of the datalog version of the translated ontology by an abstract syntax, explain how a datalog rule can be translated to an SQL statement, and discuss the inferencing over datalog programs.

The abstract syntax for our datalog program is given in Listing 1.1 using a BNF. In this notation, the terminals are quoted, the non-terminals are not quoted, alternatives are separated by vertical bars, and components that can occur zero or more times are enclosed by braces followed by a superscript asterisk symbol ($\{\ldots\}^*$). A class atom represented by *class(i-object)* in the BNF consists of a class and a single argument representing an individual. For example, an atom *Person(x)* holds if x is an instance of the class *Person*. Similarly, an individual property atom represented by *ObjectProperty(i-object,i-object)* consists of an object property and two arguments representing individuals. For example, an atom *hasDog(x,y)* holds if x is related to y by property *hasDog*. A functional object property such as *hasMother* is encoded as *FunctionalObjectProperty (hasMother)*. If an atom is a ground fact, i.e., there are no variables in its argument list, we call it a restrictive atom, because such an atom is restricted to appear only in the head of a datalog rule.

```
Program ::=Rule {Rule}*
Rule ::= Head  | Head '←' Body
Head ::= Atom  | RestrictedAtom
Body ::= Atom{∧ Atom}*
Atom ::= Class '(' i-object ')'
        | ObjectProperty '(' i-object ',' i-object ')'
        | DataProperty '(' i-object ',' d-object ')'

RestrictedAtom ::= 'InverseObjectProperty(' PropertyID ',' PropertyID')'
                 | 'FunctionalObjectProperty(' PropertyID ')'
                 | 'InverseFunctionalObjectProperty(' PropertyID ')'
                 | 'SymmetricObjectProperty(' PropertyID ')'
                 | 'AsymmetricObjectProperty(' PropertyID ')'
                 | 'TransitiveObjectProperty(' PropertyID ')'
                 | 'IrreflexiveObjectProperty(' PropertyID ')'
                 | 'FunctionalDataProperty(' PropertyID ')'
```

```
                      | 'ObjectMinCardinality(' n PropertyID ClassID ')'
                      | 'ObjectMaxCardinality(' n PropertyID ClassID')'
                      | 'ObjectPropertyDomain(' ClassID ')'
                      | 'ObjectPropertyRange(' ClassID ')'
                      | 'DataPropertyDomain(' ClassID ')'
                      | 'DataPropertyRange(' ClassID ')'

i-object ::= i-variable | individualID
d-object ::= d-variable | dataLiteral
i-variable ::= 'I-variable(' URIreference ')'
d-variable ::= 'D-variable(' URIreference ')'
```

Listing 1.1. Abstract syntax for datalog programs

As we already mentioned, storing asserted and inferred information is part of materialization and to achieve this, we translate each datalog rule to an equivalent SQL statement. We use a database structure adapted from [18] which has 33 relational tables to store OWL 2 RL ontologies. The structure uses a metamapping approach, putting all the Class assertions into one table, all the Object Property assertions into a second table and all the Data Property assertions into a third table, rather than using a separate table for each predicate. Extensions of the database corresponding to extensions of an ontology are then easy to make. A fragment of the database structure is given in Figure 2 where some tables including their column names are shown. An arrow between two tables represents a referential constraint (functional dependency) between the tables. Referential constraints are also known as foreign keys.

In our datalog program, a datalog rule has one of the following forms

$$head(h_1, \dots h_n) \tag{1}$$

$$head(h_1, \dots h_n) \leftarrow body(b_1, \dots, b_n) \tag{2}$$

$$head(h_1, \dots h_n) \leftarrow body_0(b_1, \dots, b_n) \wedge \dots \wedge body_n(b_1, \dots, b_n) \tag{3}$$

Datalog rules are closely related to operations in relational algebra, and the foundation of SQL is also relational algebra. Analogies between datalog and relational query languages such as SQL are well known and well studied [3]. We translate the three kinds of datalog rules to their corresponding SQL statements as follows:

```
(1) INSERT INTO <Table1> VALUES ( h1,...hn)
(2) INSERT INTO <Table1> SELECT
                    <Projectors> FROM <Tables> WHERE <SELECTORS>
(3) INSERT INTO <Table1> SELECT <Projectors>
            FROM <Table2> JOIN ... JOIN <TableN> WHERE <SELECTORS>
```

We use an exhaustive forward-chaining approach to infer implicit knowledge, i.e., for each class/property assertion a forward-chaining is performed. This is a novel database-driven forward chaining. We first translate all the ABox facts and the TBox rules to their corresponding SQL statements. Executing the SQL statements corresponding to the ABox stores these facts into a relational database. For each fact we determine the rules relevant for forward chaining. Executing the SQL statements for these rules stores new (inferred) facts into the database.

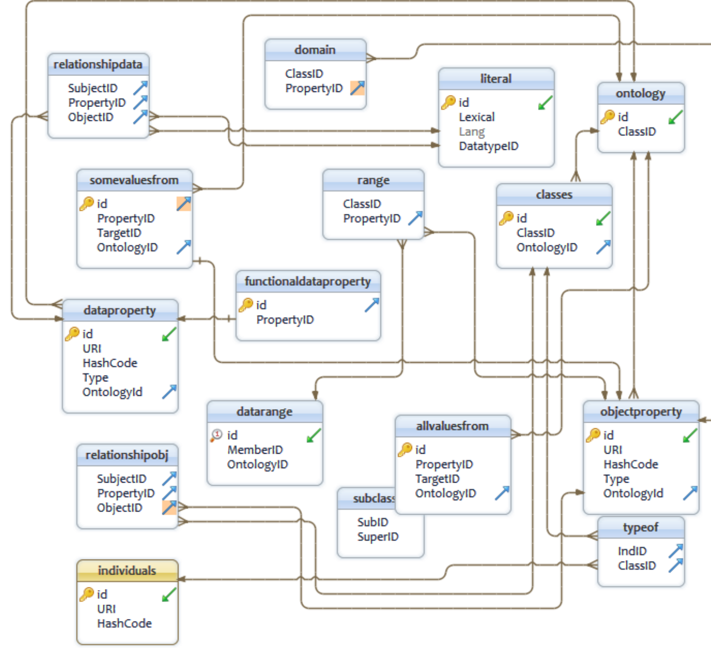

Fig. 2. A fragment of the database schema

Before storing any inferred information, we check whether it violates any restrictions listed in Table 2 by executing SQL statements corresponding to the restrictions checker's axioms and concepts.

Note that the W3C also recommends a set of rules corresponding to the OWL 2 RL profile. However, we are using set of datalog rules because the complexity of forward-chaining approach over datalog programs is polynomial and the relationship between datalog and SQL facilitates our database-driven forward-chaining approach.

Our algorithm, $Materialize(\mathcal{R},\ \mathcal{F})$, takes the datalog version of the OWL 2 RL ontology, and performs forward chaining to infer implicit knowledge. The algorithm $Materialize(\mathcal{R},\ \mathcal{F})$ (Line 1 - 8) invokes the procedure $Consequences(r, \mathcal{F})$ to populate the relational database by asserted ABox facts and to select a set of firable rules to perform database-driven forward-chaining. The $Consequences(r, \mathcal{F})$ (Line 1 -4) first checks whether the first argument is a rule or a fact. If it is a fact, then it converts to it an equivalent SQL statement and executes the SQL statement to store the asserted or inferred facts into the database. If the argument is not a fact (Line 5 - 9), this procedure checks whether the rule is firable. A firable rule is enabled if the body predicates of the rule are matched by asserted or inferred facts. (Note we do not add inferred facts to F, as our algorithm is not main-memory-based. Rather the *isFirable(r)* function accesses (asserted and inferred) facts from the database.) After getting the set of firable rules, the

$Materialize(\mathcal{R},\ \mathcal{F})$ algorithm, (Line 7), translates each rule to its equivalent SQL statement and translates all others datalog rules (i.e., not firable rules) to their equivalent SQL statements in (Line 10 -11). Before storing inferred facts by executing all the SQL statements, the $checkRestriction(s)$ (Line 12) method checks whether any assertion violates any restrictions and if not, it allows the execution of the associated SQL statement by $executeSQL(s)$ to store the information in a relational database, otherwise it raises an exception message about the inconsistency of the ABox (Line 11 - 13). For example, if $hasMother(x, y)$ is a functional object property, the restrictions checker queries the relational database to check whether x is connected to more than one different y. Full details of the $checkRestriction(s)$ method may be found in [11].

Algorithm 1: Materialize($\mathcal{R}$, $\mathcal{F}$) - materialize a datalog program into the database.

Data: $\mathcal{R}$- set of datalog rules $\mathcal{F}$- set of ABox facts
Result: $\mathcal{S}$- set of SQL statements.

```
1  repeat
2      inferred ⇐ false
3      for ∀ r ∈ R do
4          for ∀ f ∈ Consequences(r, F) do
5              if f ∉ F then
6                  inferred ⇐ true
7                  S ⇐ datalogToSQL(f)
8  until ! inferred
9  for ∀ r ∈ R do
10     S ⇐ datalogToSQL(r)
11 for ∀ s ∈ S do
12     if checkRestriction(s) then
13         executeSQL(s)
```

2.3 Query Processing

In this subsection, we describe a query interface to extract materialized knowledge from the database. SPARQL [25] is a W3C recommendation for querying RDF graphs. An RDF graph is a collection of *(subject, predicate, object)* triples. We cannot use SPARQL as it exists as a query language for OWL 2 RL for two reasons. First, it is based on the triple patterns of RDF graphs, but RDF triple patterns do not match the well-defined OWL 2 RL syntax, so a modified version of SPARQL is necessary. Second, in our framework, we materialize ontologies to relational databases. So we need a modified version of SPARQL to retrieve data from relational databases.

SQL, the query language for relational databases, includes support for large data-storage, efficient indexing schemes, and query optimization. If we directly

Procedure Consequences($r, \mathcal{F}$) - recursively applied for all the predicates of a rule body to derive the consequence.

```
Data: r - a datalog rule, F- set of ABox facts.
1 if r is a fact then
2     datalogToSQL(r)
3     executeSQL(s)
4     return r
5 inferred ⇐ ∅
6 for ∀ f ∈ F do
7     if isFirable(r) then
8         inferred ⇐ inferred ∪ r
9 return inferred
```

use SQL to extract knowledge from materialized ontologies, then users have to learn the underlying relational schemas. Many real-world semantic web-based applications need to extract data from both relational sources and ontologies. So a uniform query language is necessary for accessing both structured data (e.g., from relational databases) and semi-structured data (e.g., RDF triples, OWL ontologies).

In order to use SPARQL for querying ontologies based on OWL 1, Sirin and Parsia [27] designed a query language by modifying SPARQL called SPARQL-DL, a substantial subset of SPARQL, by mapping RDF triple patterns using OWL 1 DL semantics. Therefore, SPARQL-DL supports only the semantics of OWL 1 ontologies. The SPARQL-DL API [2] supports a query language, which we will refer to as SPARQL-DL$_E$, for OWL 2 ontologies (including OWL 2 RL ontologies). However, the SPARQL-DL API is built to interface with main-memory-based OWL 2 reasoners, so we need some modifications to support queries over the relational database-based reasoner. In this subsection, we describe the semantics of SPARQL-DL$_E$ and explain our modifications.

The semantics of SPARQL-DL$_E$. SPARQL-DL$_E$ is an expressive query language that can combine TBox and ABox queries. Here we briefly describe the semantics of SPARQL-DL$_E$ which we extended from [27].

Let $\mathcal{O}$ be an OWL 2 ontology, let $V_O = (\mathcal{V}_{cls}, \mathcal{V}_{op}, \mathcal{V}_{dp}, \mathcal{V}_{ind}, \mathcal{V}_D, \mathcal{V}_{lit})$ be a vocabulary for $\mathcal{O}$ and let $I = (\Delta^I, .^I)$ be an interpretation for $\mathcal{O}$. The list of SPARQL-DL query atoms for OWL 1 and their corresponding semantics may be found in [27]. Two new query atoms are required to deal with OWL 2: Reflexive(p) and Irreflexive(p). Their semantics is given in Table 3. Here $a_{(i)} \in \mathcal{V}_{uri} \cup \mathcal{V}_{var} \cup \mathcal{V}_{bnode}$, $d \in \mathcal{V}_{uri} \cup \mathcal{V}_{var} \cup \mathcal{V}_{bnode} \cup \mathcal{V}_{lit}$, $C_{(i)} \in \mathcal{V}_{var} \cup \mathcal{S}_c$, $p_{(i)} \in \mathcal{V}_{uri} \cup \mathcal{V}_{var}$, $\mathcal{V}_{cls}$ is the set of classes, $\mathcal{V}_{op}$ is the set of object properties, $\mathcal{V}_{dp}$ is the set of data properties, $\mathcal{V}_{ind}$ is the set of individuals, $\mathcal{V}_{lit}$ is the set of literals and $\mathcal{V}_D$ is the set of data types of $\mathcal{O}$. Note that OWL 2 RL does not

include reflexivity, so our reasoning system does not support queries involving reflexive properties.

Query atom q	$\mathcal{I} \models_\delta q$ if
$Type(a, C)$	$\delta(a) \in C^I$
$Reflexive(p)$	$< a, b > \in p^I$ implies $a = b$
$Irreflexive(p)$	$< a, b > \in p^I$ implies $a \neq b$

Table 3. Satisfaction of a SPARQL-DL$_E$ query atom with respect to an interpretation

An evaluation $\delta : \mathcal{V}_{ind} \cup \mathcal{V}_{bnode} \cup \mathcal{V}_{lit} \rightarrow \Delta^{\mathcal{I}}$ is a mapping from the individual names, blank nodes, and literals used in the query to the elements of the interpretation domain $\Delta^{\mathcal{I}}$ subject to the requirement $\delta(a) = a^{\mathcal{I}}$ if $a \in \mathcal{V}_{ind}$ or $a \in \mathcal{V}_{lit}$. The interpretation $\mathcal{I}$ satisfies a query atom q, $\mathcal{I} \models_\delta q$, if q is compatible with the corresponding condition for the query atom. $\mathcal{I}$ satisfies a query $Q = q_1 \wedge \ldots \wedge q_n$. w.r.t. an evaluation δ **iff** $\mathcal{I} \models_\delta q_i$ for every $i = 1, \ldots, n$.

A solution to a query Q is a mapping $\mu : \mathcal{V}_{var} \rightarrow \mathcal{V}_{cls} \cup \mathcal{V}_{op} \cup \mathcal{V}_{dp} \cup \mathcal{V}_{lit}$ such that when all the variables in Q are substituted with the corresponding value from μ we get a ground query $\mu(Q)$ (i.e., an atom having no variables) compatible with $\mathcal{V}_{\mathcal{O}}$ and $\mathcal{O} \models \mu(Q)$.

Implementation of SPARQL-DL$_E$ in OwlOntDB. We modified the SPARQL-DL API to extract knowledge from the relational database and implemented it in our system. The SPARQL-DL API is built on top of the OWL API [15]. The SPARQL-DL API was designed in such a way that it can answer mixed TBox and ABox queries by invoking interfaces, such as $allC(\mathcal{O})$, $allDP(\mathcal{O})$, $allOP(\mathcal{O})$, $allI(\mathcal{O})$, etc., provided by the ontology reasoner. We modified these interfaces (see the full list of interfaces that required modification below) so that this API can evaluate queries using our persistent reasoning system.

1. $allC(\mathcal{O})$, $allDP(\mathcal{O})$, $allOP(\mathcal{O})$, $allI(\mathcal{O})$ return all classes, data properties, object properties, and individuals, respectively, defined in $\mathcal{O}$.
2. $subC(\mathcal{O}, \mathcal{C})$, $supC(\mathcal{O}, \mathcal{C})$, $eqC(\mathcal{O}, \mathcal{C})$ return all sub classes, super classes, and equivalent classes, respectively, of class C in $\mathcal{O}$.
3. $subOP(\mathcal{O}, \mathcal{P})$, $supOP(\mathcal{O}, \mathcal{P})$, $eqOP(\mathcal{O}, \mathcal{P})$, $subDP(\mathcal{O}, \mathcal{P})$, $supDP(\mathcal{O}, \mathcal{P})$, $eq DP(\mathcal{O}, \mathcal{P})$ return all sub object properties, super object properties, equivalent object properties, sub data properties, super data properties, and equivalent data properties, respectively, of properties p in $\mathcal{O}$.
4. $en(\mathcal{O}, q)$ checks whether $\mathcal{O} \models q$ for a SPARQL-DL$_E$ atom q.

Recall we stored asserted and inferred information from ontologies into databases. Therefore, we need an SQL query for each interface described in (1)-(3) to retrieve relevant information from corresponding tables of the relational database. For example, the SQL queries for $subC(\mathcal{O}, \mathcal{C})$ and $supOP(\mathcal{O}, \mathcal{P})$ are:

```
SELECT SubID FROM SubClassOf WHERE SuperID = C
SELECT SuperPropertyID FROM SuperPropertyOf WHERE SubPropertyID = P
```

The SQL queries retrieve all subclasses for a given class and all super properties for a given object property, respectively. The query given in 4. is evaluated by the

SPARQL-DL API by invoking the appropriate interfaces discussed in 1.-3. For instance, if we consider the query $q = SubClassOf('' Person'', c)$, the SPARQL-DL API will invoke the interface $subC(\mathcal{O},\ Person)$ to retrieve all the subclasses of "Person" from the database.

3 Evaluation

We evaluated OwlOntDB using an OWL 2 RL pain management ontology constructed from the guidelines for the management of cancer related pain in adults, which provides a standard approach in assessing and managing cancer related pain in adults across Nova Scotia, Canada [7]. We evaluated OwlOntDB using this pain ontology because there are no widely accepted benchmarks for OWL 2. In [22], the authors discussed this problem and identified that while there are some ontologies that can be used as standards for testing TBox reasoning, there are no such standards for ABox reasoning. Evaluation was done on a laptop computer with 2.4 GHz Intel Core 2 Duo processor, 4 GB of RAM running Mac OS X version 10.6.8.

We use the pain management ontology [26] that includes the terminology and concepts of health and medicine used in the Guysborough Antigonish Strait Health Authority (GASHA) and some terms from SNOMED-CT [1], ICNP [14], and the guidelines for cancer pain treatment. A fragment of the pain management ontology is depicted in Figure 3. Our ontology includes several classes including *Pain*, *Person*, *Patient*, *PainIntensityType*, *SpecialPainProblem*, *SideEffects*; some object properties including *hasPainIntensity*, *Domain:Pain*, *Range:PainIntensityType*, and data properties including *hasPainLevel*, *Domain :Pain*, *Range:xsd:int*, inverse object properties such as *isFeeling* and *isFeltBy*, and functional object properties including *hasPainLevel*, i.e., each pain level belongs to an instance of *Pain* class. We also use propositional connectives to create complex class expressions (e.g., persons who feel pain are patients, in DL $Person \sqcap \exists isFeeling.Pain \sqsubseteq Patient$). We developed a data generator similar to that developed for the LUBM benchmark [13] to synthetically generate large numbers of instances for the pain management ontology. We generated five test datasets, PM_{250}, PM_{500}, PM_{1000}, PM_{2000}, and PM_{3000}, where the number of patients n = 250, 500, 1000, 2000, and 3000, respectively, and evaluated the following two queries to evaluate the performance of our system. The SPARQL-DL$_E$ formulation of each query appears below its natural language formulation.

PM Q_1. *Determine the medication information of all patients who feel "Mucositis" pain.*

```
PREFIX pm: <http://logic.stfx.ca/ontologies/PainOntology.owl#>
 SELECT ?i ?j WHERE {
  Type(?i,pm:Patient), PropertyValue(?i, pm:isFeeling, pm:MucositisPain),
    PropertyValue(?i, pm:hasMedication, ?j) }
```

PM Q_2. *Find the names of those relatives (of patients) who serve as informal care givers.*

```
PREFIX pm: <http://logic.stfx.ca/ontologies/PainOntology.owl#>
```

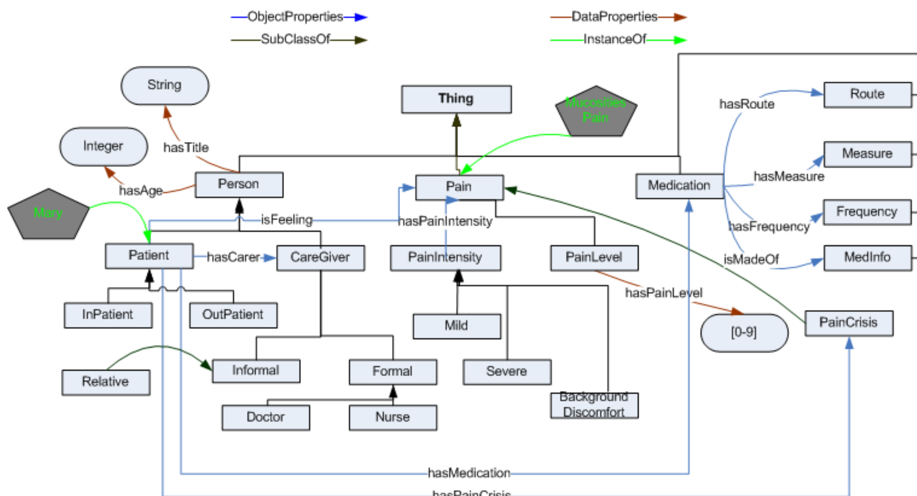

Fig. 3. A fragment of the pain management ontology

```
SELECT ?i ?j WHERE {
  Type(?i,pm:Patient), PropertyValue(?i, pm:hasCarer , ?j),
    SubClassOf(?j, pm:Relative) }
```

The first query is a conjunctive ABox query and the second query is a conjunctive mixed TBox and ABox query. We evaluated these queries over the corresponding ontologies using OwlOntDB and also using two highly optimized in-memory reasoners *Pellet* and *RacerPro* 2.0. Our goal is to show that in-memory reasoners cannot deal with ontologies with large ABoxes. The OwlOntDB materializes the information to a database, so it needs an initial processing before query evaluation. The initial processing time (i.e., materialization time) for five datasets required for OwlOntDB and total number of axioms in each ontology are given in Table 4.

	PM_{250}	PM_{500}	PM_{1000}	PM_{2000}	PM_{3000}
No. of Axioms	20344	40396	77600	156308	231555
Time (sec.)	20.71	45.94	90.34	263.24	345.28

Table 4. Time required for materialization for the PM ontology

We also evaluated our system using two well-known benchmark ontologies for OWL 1: LUBM - an ontology about organizational structures of universities developed to test the performance of ontology management and reasoning systems [1], and the Wine ontology - an ontology containing a classification of wines, taken from the KAON2 site [2]. We used two LUBM datasets from LUBM namely, $lubm_1$, and $lubm_{10}$, where 1 and 10 are the number of universities used to generate test data and two Wine datasets $wine_1$ - the original wine ontology,

[1] http://swat.cse.lehigh.edu/downloads/index.html
[2] http://kaon2.semanticweb.org/download/test_ontologies.zip

and $wine_5$ - which is synthetically generated by replicating 2^5 times the ABox of $wine_1$. The details of both ontologies can be found in [22]. We evaluated the following queries over the appropriate LUBM and Wine ontologies:

LUBM Q_1. *Find names of the students who are university employees along with their type of employment.* (Note this is a mixed ABox and TBox query.)

```
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
 SELECT * WHERE {
  Type(?x, ub:Student), Type(?x, ?C), SubClassOf(?C, ub:Employee)}
```

LUBM Q_2. *Find the names of all students.*

```
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
 SELECT * WHERE {
  Type(?x, ub:Student)}
```

Wine Q_1 *Determine all instances of "AmericanWine".*

```
PREFIX wine: <http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#>
 SELECT ?i WHERE {
  Type(?i, wine:AmericanWine)}
```

Wine Q_2 *Determine all the instances of wine which are "Dry".*

```
PREFIX wine: <http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#>
  SELECT ?i WHERE {
     Type(?i,?x), SubClassOf(?x, wine:DryWine) }
```

The materialization time for the LUBM and Wine ontologies required for OwlOntDB and total number of axioms in each ontology are given in Table 5.

	$LUBM_1$	$LUBM_{10}$	$Wine_1$	$Wine_5$
No. of Axioms	84562	1316410	649	5576
Time (sec.)	117.58	830.53	21.422	217.5

Table 5. Time required for materialization for the LUBM and Wine ontologies

The query evaluation time for *Pellet*, *RacerPro*, and OwlOntDB is given in Table 6. Standard tableau-based reasoners support more expressive fragments of DL and efficiently perform reasoning over ontologies with small ABoxes. From our experiments, we found that for ontologies with large ABoxes, our reasoning outperformed its tableau counterpart. Although we first used a tableau-based DL reasoner for the TBox reasoning required for classification, we get better performance for the query evaluation than these tableau-based reasoners because we first materialized the inferred information into a database. After the materialization, reasoning over a materialized ontology is simply an SQL query into a relational database. Main-memory based reasoners perform inferencing for each query, so they take a longer time when the ABox is large. Indeed, we can see from the Table 6 that we do not get any query result from either *Pellet* or *RacerPro* for a large ontology like $LUBM_{10}$, but, the query response time

for this ontology using our OwlOntDB is very low. The disadvantage of the materialization technique is that it takes a long time initially to materialize the ontology.

	Q_1			Q_2		
	Pellet	*RacerPro*	OwlOntDB	*Pellet*	*RacerPro*	OwlOntDB
PM_{250}	41.65	27	7.53	65.85	...	9.79
PM_{500}	91.83	70	11.71	127.038	...	14.89
PM_{1000}	179.50	105.5	16.89	259.678	...	19.01
PM_{2000}	718.18	225	20.78	959.32	...	23.21
PM_{3000}	-	430	29.21	-	...	34.01
$LUBM_1$	129.02	...	3.43	127	73	0.79
$LUBM_{10}$	-	...	29.07	-	-	15.03
$Wine_1$	2.95	24	0.047	2.9	...	0.11
$Wine_5$	6.08	37	0.3435	386.59	...	1.171

Table 6. A comparison of query answering times (in seconds). "-" means that the reasoner failed to return the result and "..." means that the reasoner does not support the query. Note that *RacerPro* supports only TBox queries, a limitation not due to in-memory problems but due to the nature of *RacerPro*.

Recall we use a standard DL reasoner for the TBox reasoning which creates a complete class hierarchy if the corresponding TBox is consistent. Therefore, OwlOntDB is complete for TBox reasoning. The ABox reasoning is based on a database-driven forward chaining approach. The empirical completeness of ABox reasoning was checked by comparing the ABox reasoning results with the results of the OWL 2 reasoners *Pellet* and *RacerPro*. A similar empirical approach is used in [23], to compare their in-memory-based OWL 2 RL reasoners with Hermit. While efficient for the TBox reasoning, their in-memory-based implementation performed poorly on ontologies with large ABoxes. We are still working on the algorithm to deal with the situation where the set of axioms is cyclic; currently our algorithm may not terminate if the set of axioms is cyclic.

4 Related Work

There has recently been considerable interest in developing scalable persistent reasoning systems for Semantic Web applications. The integration of relational databases and DL-based reasoners has been realized in many research initiatives including [18], [29], [4]. Most scalable reasoning systems such as Minerva [29], SOAR [18], and DLDB2 [24] combine existing DL reasoners with logic programming-based approaches. However, these reasoners are based on DLP, providing only incomplete coverage of OWL 2 RL reasoning. We use a 2-phase approach to deal with all OWL 2 RL axioms.

OWLIM [16] is an in-memory reasoner. It also uses the logic programming based approach (i.e., forward-chaining for inferencing) and focuses only on the DLP fragment, hence it covers a subset of OWL 2 RL. Another logic programming-based DL reasoner is KAON2 [21]. In KAON2, the ontology is translated into a

logic program and then it is materialized into a deductive database for querying and storing the information. This approach is similar to our approach, except we develop a scalable reasoner for OWL 2 RL, a more expressive fragment than that supported by KAON2, and materialize the information to a relational database rather than to a deductive database.

Another database-driven reasoning system is Orel [17], which covers the full profile of OWL 2 RL as well as the OWL 2 EL profile, using an algorithm based on DLP. However, this system supports only TBox reasoning; it does not support (conjunctive) query answering (i.e., ABox reasoning). Another limitation of this system is that it does not support a standard query language for the extraction of knowledge from materialized ontologies, therefore, users have to know the detailed structure of the underlying database schema to extract knowledge using SQL. To extract knowledge from the database OwlOntDB supports SPARQL-DL$_E$, so users of our system have to know only about the ontology.

DLEJena [19] is an OWL 2 RL reasoner that also combines a forward-chaining-based rule engine Jena and a DL reasoner *Pellet*. It supports a practical subset of OWL 2 RL. A pair of OWL 2 RL reasoners is described in recent work, [23] using two existing rule systems Jess and Drools. However, these reasoners are all in-memory-based reasoners; they are not scalable: they cannot handle ontologies with large ABoxes. We could not find any scalable OWL 2 RL reasoners to use for comparison with our approach.

5 Conclusion and Future Work

Scalable reasoning is crucial for the development of large-scale ontology-driven applications. In this paper, we propose a practical scalable ontology reasoning approach. The combination of DL reasoners with logic-based inferencing using datalog exploits the particular advantages of each method in order to support expressive ontologies, such as those which use OWL 2 RL in their TBoxes, and large ABoxes. Logic-based approaches give us scalable reasoning strategies, and database systems are a well-known technology for handling large amounts of data. We develop a hybrid approach by applying database-driven forward chaining approach over logic-based translated ontologies that allows us to perform scalable reasoning over ontologies with large ABoxes. There is a number of advantages and disadvantage for materialization techniques. However, they are good for many applications where query answering is more frequent and updating is less frequent.

Our approach is still preliminary and some improvements can be made. One of the future directions to improve our system is to remove the Unique Name Assumption (UNA) because UNA is not made in OWL 2 semantics. The initial processing time for the materialization is very high. Parallel and distributed computing may be applied to reduce the materialization time. However, this will not be fast enough for applications that require frequent update and real-time query answering, such as healthcare applications, where ontologies are used to drive decision support systems. The current strategy is to rematerialize the

whole ontology if the ontology is updated, but this brings a heavy overhead as the time required for materialization must be added to the time for query-answering. Incremental materialization is anticipated to be an efficient solution for the update problem. We are working to reduce materialization time by replacing our exhaustive forward chaining inferencing approach by an incremental approach that rematerializes relevant axioms. We note that *Pellet* supports incremental materialization but only for concept assertions. There are also some works in deductive database areas for incremental maintenance of truth in materialization [28]; a further investigation can be made to check whether these techniques can be used for relational databases. Efficient handling of frequent updates in an ontology with large number of instances is an important aspect of developing large-scale ontology-driven systems such as healthcare systems.

Acknowledgments This work is supported by an NSERC Discovery Grant, an NSERC Industrial Post Graduate Fellowship and ACOA. We would like to thank Fazle Rabbi for the help to develop benchmark data generator, Jocelyne Faddoul and Fazle Rabbi for support on *RacerPro* and Rachel Embree and Mary Heather Jewers for the fruitful discussions about ontologies and the guidelines for the management of cancer related pain in adults. We thank the anonymous referees for their comments and corrections.

References

1. SNOMED-CT Systematized Nomenclature of Medicine-Clinical Terms. http://www.ihtsdo.org/snomed-ct/ (2007)
2. SPARQL-DL API. http://www.derivo.de/en/resources/sparql-dl-api/ (2011)
3. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
4. Acciarri, A., Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Palmieri, M., Rosati, R.: QuOnto: Querying Ontologies. In: Veloso, M.M., Kambhampati, S. (eds.) AAAI. pp. 1670–1671. AAAI Press / The MIT Press (2005)
5. Al-Jadir, L., Parent, C., Spaccapietra, S.: Reasoning with large ontologies stored in relational databases: The OntoMinD approach. Data & Knowledge Engineering 69(11), 1158–1180 (November 2010)
6. Baader, F., McGuinness, D.L., Nardi, D., (Eds.), P.F.P.S.: The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
7. Broadfield, L., Banerjee, S., Jewers, H., Pollett, A.J., Simpson, J.: Guidelines for the management of cancer-related pain in adults. Supportive care cancer site team, cancer care Nova Scotia, Canada. (2005)
8. Broekstra, J.: Storage, Querying and Inferencing for Semantic Web Languages. Ph.D. thesis, VU Amsterdam (2005)
9. Calvanese, D., Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite Family. Journal of Automated Reasoning 39, 385–429 (October 2007)
10. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R.: Ontologies and Databases: The DL-Lite Approach. In: Tessaris, S., Franconi, E., Eiter, T., Gutierrez, C., Handschuh, S., Rousset, M.C., Schmidt, R.A. (eds.) Reasoning Web. LNCS, vol. 5689, pp. 255–356. Springer (2009)

11. Faruqui, R.U.: Scalable reasoning over large ontologies. MSc thesis, St. Francis Xavier University, 2012, Available at http://logic.stfx.ca/thesis/
12. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: Combining logic programs with description logic. In: Proceedings of the 12th international conference on World Wide Web. pp. 48–57. ACM Press (2003)
13. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. J. Web Sem. 3(2-3), 158–182 (2005)
14. Hardiker, N., Coenen, A.: A formal foundation for ICNP. Journal of Stud Health Technol Inform 122, 705–709 (2006)
15. Horridge, M., Bechhofer, S.: The OWL API: A java API for working with OWL 2 Ontologies. In: 6th OWL Experienced and Directions Workshop (OWLED) (October 2009)
16. Kiryakov, A., Ognyanov, D., Manov, D.: OWLIM - A Pragmatic Semantic Repository for OWL. In: Dean, M., Guo, Y., Jun, W., Kaschek, R., Krishnaswamy, S., Pan, Z., Sheng, Q.Z. (eds.) WISE Workshops. Lecture Notes in Computer Science, vol. 3807, pp. 182–192. Springer (2005)
17. Krötzsch, M., Mehdi, A., Rudolph, S.: Orel : Database-Driven reasoning for OWL 2 Profiles. In: 23rd Int. Workshop on Description Logics (DL2010). pp. 114–124 (2010)
18. Lu, J., Ma, L., Zhang, L., Brunner, J.S., Wang, C., Pan, Y., Yu, Y.: SOR: a practical system for ontology storage, reasoning and search. In: Proceedings of the 33rd international conference on Very large data bases. pp. 1402–1405. VLDB '07, VLDB Endowment (2007)
19. Meditskos, G., Bassiliades, N.: DLEJena: A practical forward-chaining OWL 2 RL reasoner combining Jena and Pellet. Web Semant. 8(1), 89–94 (Mar 2010), `http://dx.doi.org/10.1016/j.websem.2009.11.001`
20. Motik, B., Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology Language: Profiles, W3C Recommendation. http://www.w3.org/TR/owl2-profiles/ (October 2009)
21. Motik, B.: KAON2 - Scalable Reasoning over Ontologies with Large Data Sets. ERCIM News 2008(72) (2008)
22. Motik, B., Sattler, U.: A Comparison of Reasoning Techniques for Querying Large Description Logic ABoxes. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS, vol. 4246, pp. 227–241. Springer, Phnom Penh, Cambodia (2006)
23. O'Connor, M.J., Das, A.: A Pair of OWL 2 RL Reasoners. In: Klinov, P., Horridge, M. (eds.) OWLED. CEUR Workshop Proceedings, vol. 849. CEUR-WS.org (2012)
24. Pan, Z., Zhang, X., Heflin, J.: DLDB2: A Scalable Multi-perspective Semantic Web Repository. In: Web Intelligence. pp. 489–495. IEEE (2008)
25. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C Recommendation. Available at http://www.w3.org/TR/rdf-sparql-query/ (2008)
26. Rakib, A., Faruqui, R.U., MacCaull, W.: Verifying resource requirements for ontology-driven rule-based agents. In: Lukasiewicz, T., Sali, A. (eds.) FoIKS. Lecture Notes in Computer Science, vol. 7153, pp. 312–331. Springer (2012)
27. Sirin, E., Parsia, B.: SPARQL-DL: Sparql query for OWL-DL. 3rd OWL Experiences and Directions Workshop (OWLED-2007) (2007)
28. Volz, R., Staab, S., Motik, B.: Incrementally Maintaining Materializations of Ontologies Stored in Logic Databases. Journal of Data Semantics II 3360, 1–34 (2005), lNCS, Springer
29. Zhou, J., Ma, L., Liu, Q., Zhang, L., Yu, Y., Pan, Y.: Minerva: A Scalable OWL Ontology Storage and Inference System. In: Mizoguchi, R., Shi, Z., Giunchiglia, F. (eds.) The Semantic Web AWSC, LNCS, vol. 4185, pp. 429–443. Springer (2006)