SPARQL Query Answering over OWL Ontologies

Ilianna Kolliα1,2, Birte Glimm2, and Ian Horrocks2

1 National Technical University of Athens, Greece
2 Oxford University, UK

Presented By
Mohamed Sabri
Motivation:

• The query evaluation mechanism defined in the SPARQL Query specification is based on subgraph matching (simple entailment).

• SPARQL 1.1 includes more elaborate entailment regimes; including RDFS and OWL.

• Query answering under such entailment regimes is more complex as it may involve retrieving answers that only follow implicitly from the queried graph.

• While several methods and implementations for SPARQL under RDFS semantics are available, methods that use OWL semantics have not yet been well-studied.
Effects of Different Entailment Regimes:

- Green dashed lines indicate RDF-entailed triples and red dashed lines indicate triples that are also RDFS-entailed.

- SELECT ?pub WHERE { ?pub rdf:type ex:Publication }
OWL Reasoning:

SELECT ?pub WHERE { ?pub rdf:type ex:Publication }

1) Materialization approaches.

2) Query Rewriting approaches.

3) Combined approaches.
OWL Reasoning:

SELECT ?pub WHERE { ?pub rdf:type ex:Publication }

1) Materialization approaches.

2) Query Rewriting approaches.

3) Combined approaches.
OWL Reasoning:

```
SELECT ?pub WHERE { ?pub rdf:type ex:Publication }
```

1) Materialization approaches.

2) Query Rewriting approaches.

3) Combined approaches.
Motivation (cont’d):

• However, the previous techniques are only applicable for less expressive OWL 2 profiles.

• In this paper, the authors present a sound and complete algorithm for answering SPARQL queries under the OWL 2 Direct Semantics entailment regime (SPARQL-OWL).

• ...the first implementation to fully support SPARQL-OWL!

• “[Query Answering] is a very active research area, with many different techniques being developed and investigated... it is reasonable to expect that the future performance improvements in query answering systems will be even more spectacular than those achieved in the past by class reasoning systems”, Ian Horrocks.
Motivation (cont’d):

• However, the previous techniques are only applicable for less expressive OWL 2 profiles.

• In this paper, the authors present a sound and complete algorithm for answering SPARQL queries under the *OWL 2 Direct Semantics entailment regime* (SPARQL-OWL).

• ...the first implementation to fully support SPARQL-OWL!

• “[Query Answering] is a very active research area, with many different techniques being developed and investigated... it is reasonable to expect that the future performance improvements in query answering systems will be even more spectacular than those achieved in the past by class reasoning systems”, Ian Horrocks.
Web Ontology Language OWL:

SubClassOf(:DogOwner ObjectSomeValuesFrom(:owns :Dog))
SubClassOf(:CatOwner ObjectSomeValuesFrom(:owns :Cat))
ObjectPropertyDomain(:owns :Person)
ClassAssertion(ObjectUnionOf(:DogOwner :CatOwner) :mary)
ObjectPropertyAssertion(:owns :mary _:somePet)

- **We can infer:** $O \models \text{ClassAssertion(:Person :mary)}$

- However, there is no unique canonical model that could be used to answer queries. Why?!

- And in OWL it cannot be guaranteed that the models of an ontology are finite!
Web Ontology Language OWL:

SubClassOf(:DogOwner ObjectSomeValuesFrom(:owns :Dog))
SubClassOf(:CatOwner ObjectSomeValuesFrom(:owns :Cat))
ObjectPropertyDomain(:owns :Person)
ClassAssertion(ObjectUnionOf(:DogOwner :CatOwner) :mary)
ObjectPropertyAssertion(:owns :mary _:somePet)

- We can infer: O |= ClassAssertion(:Person :mary)

- However, there is no unique canonical model that could be used to answer queries. Why?!

- And in OWL it cannot be guaranteed that the models of an ontology are finite!
Definitions (simplified):

**Definition 1.** We write $I$ for the set of all IRIs, $L$ for the set of all literals, and $B$ for the set of all blank nodes. The set $T$ of RDF terms is $I \cup L \cup B$. Let $V$ be a countably infinite set of variables disjoint from $T$. A triple pattern is member of the set $(T \cup V) \times (I \cup V) \times (T \cup V)$, and a basic graph pattern (BGP) is a set of triple patterns.

**Definition 2.** A solution mapping is a partial function $\mu: V \to T$ from variables to RDF terms.

**Definition 3.** An RDF instance mapping is a partial function $\sigma: B \to T$ from blank nodes to RDF terms.

We use:

$O_G$ to denote the result of mapping an OWL 2 DL graph $G$ into an OWL ontology.

$O_{BGP}^G$ to denote the result of mapping an OWL 2 DL graph $G$ into an OWL ontology.
SPARQL-OWL Query Answering:

• The SPARQL-OWL regime specifies what the answers are, but not how they can actually be computed.

• **A straightforward algorithm to realize the entailment regime:**
  Given an OWL 2 DL graph $G$ and a well-formed BGP $BGP$ for $G$,
  1) map $G$ into $O_G$, BGP into $O_{BGP}^G$
  2) and then simply tests, for each compatible pair $(μ, σ)$, whether $sk(O_G) |= μ(σ(O_{BGP}^G))$.

• **Compatible Mappings:** (intuitively) class variables can only be mapped to class names, object property variables to object properties, etc.

• but in the worst case, the number of distinct compatible pairs $(μ, σ)$ is exponential in the number of variables in the query, i.e., if $m$ is the number of terms in $\ldots$ and $n$ is the number of variables in $\ldots$, we test $O(mn)$ solutions.
SPARQL-OWL Query Answering:

- The SPARQL-OWL regime specifies what the answers are, but not how they can actually be computed.

- **A straightforward algorithm to realize the entailment regime:**
  
  Given an OWL 2 DL graph $G$ and a well-formed BGP $BGP$ for $G$,
  
  1) map $G$ into $O_G$, $BGP$ into $O_{BGP}^G$
  2) and then simply tests, for each compatible pair $(\mu, \sigma)$, whether
  
  $$sk(O_G) \models \mu(\sigma(O_{BGP}^G)).$$

- **Compatible Mappings:** (intuitively) class variables can only be mapped to class names, object property variables to object properties, etc.

- but in the worst case, the number of distinct compatible pairs $(\mu, \sigma)$ is exponential in the number of variables in the query, i.e., if $m$ is the number of terms in $O_G$ and $n$ is the number of variables in $O_{BGP}^G$, we test $O(mn)$ solutions.
SPARQL-OWL Query Answering (cont’d):

• So, the authors presents optimizations to reduce the number of:
  • entailment checks, and
  • method calls to the reasoner.

• However, optimizations cannot easily be integrated in the above simple algorithm since it uses the reasoner to check for the entailment of the query as a whole and does not take advantage of relations that may exist between axiom templates.

• For example, choosing a good execution order can significantly affect the performance;

consider the BGP { ?x rdf:type :A . ?x :op ?y . }
With 100 individuals only one of which belongs to class :A, you may perform 10,200 tests instead of just 200.
SPARQL-OWL Query Answering (cont’d):

• Also, instead of checking entailment, we can, for several axiom templates, *directly retrieve* the solutions from the reasoner. Example: `{ ?x rdfs:subClassOf :C }`

• Most methods of reasoners are *highly optimized*, which can significantly reduce the number of tests that are performed. Furthermore, if the class hierarchy is precomputed, the reasoner can find the answers simply with a cache lookup.

• However, the actual execution cost might vary significantly (depending on the *internal state* of the reasoner)

• The proposed algorithm internally uses an OWL 2 DL reasoner to check entailment (HermiT, in this implementation).
Simple and Complex Axiom Templates:

• We distinguish between *simple* and *complex* axiom templates, where:

  - *Simple axiom templates* are those that correspond to dedicated reasoning tasks.
    
    SubClassOf(?x :C)
  
  - *Complex axiom templates* are, in contrast, evaluated by iterating over the compatible mappings and by checking entailment for each instantiated axiom template.
    
    SubClassOf(:C ObjectIntersectionOf(?z ObjectSomeValuesFrom(?x ?y)))
    ClassAssertion(ObjectSomeValuesFrom(:op ?x) ?y)
Algorithm 1. Query Evaluation Procedure

**Input:** \( G \): the active graph, which is an OWL 2 DL graph  
\( \text{BGP} \): an OWL 2 DL BGP  

**Output:** a multiset of solutions for evaluating \( \text{BGP} \) over \( G \) under OWL 2 Direct Semantics

1: \( O_G := \text{map}(G) \)
2: \( O^G_{\text{BGP}} := \text{map}(\text{BGP}, O_G) \)
3: \( \text{Axt} := \text{rewrite}(O^G_{\text{BGP}}) \) \{ create a list \( \text{Axt} \) of simplified axiom templates from \( O^G_{\text{BGP}} \) \}
4: \( \text{Axt}^1, \ldots, \text{Axt}^m := \text{connectedComponents}(\text{Axt}) \)
5: \( \text{for } j = 1, \ldots, m \text{ do} \)
6: \( R_j := \{(\mu_0, \sigma_0) \mid \text{dom}(\mu_0) = \text{dom}(\sigma_0) = \emptyset\} \)
7: \( \text{axt}_1, \ldots, \text{axt}_n := \text{reorder}(\text{Axt}^i) \)
8: \( \text{for } i = 1, \ldots, n \text{ do} \)
9: \( R_{\text{new}} := \emptyset \)
10: \( \text{for } (\mu, \sigma) \in R_j \text{ do} \)
11: \( \text{if isSimple(axt}_i) \text{ and } ((\text{V(axt}_i) \cup \text{B(axt}_i))) \setminus (\text{dom}(\mu) \cup \text{dom}(\sigma)) \neq \emptyset \text{ then} \)
12: \( R_{\text{new}} := R_{\text{new}} \cup \{(\mu \cup \mu', \sigma'\cup \sigma') \mid (\mu', \sigma') \in \text{callReasoner}(\mu(\sigma(axt)_i)))\} \)
13: \( \text{else} \)
14: \( B := \{(\mu \cup \mu', \sigma \cup \sigma') \mid \text{dom}(\mu') = \text{V}(\mu(axt)_i), \text{dom}(\sigma') = \text{B}(\sigma(axt)_i), (\mu \cup \mu', \sigma \cup \sigma') \text{ is compatible with } axt_i \text{ and } sk(O_G)\} \)
15: \( B := \text{prune}(B, axt_i, O_G) \)
16: \( \text{while } B \neq \emptyset \text{ do} \)
17: \( (\mu', \sigma') := \text{removeNext}(B) \)
18: \( \text{if } O_G \models \mu'(\sigma'(axt)_i) \text{ then} \)
19: \( R_{\text{new}} := R_{\text{new}} \cup \{(\mu', \sigma')\} \)
20: \( \text{else} \)
21: \( B := \text{prune}(B, axt_i, (\mu', \sigma')) \)
22: \( \text{end if} \)
23: \( \text{end while} \)
24: \( \text{end if} \)
25: \( \text{end for} \)
26: \( R_j := R_{\text{new}} \)
27: \( \text{end for} \)
28: \( \text{end for} \)
29: \( R := \{(\mu_1 \cup \ldots \cup \mu_m, \sigma_1 \cup \ldots \cup \sigma_m) \mid (\mu_j, \sigma_j) \in R_j, 1 \leq j \leq m\} \)
30: \( \text{return } \{(\mu, m) \mid m > 0 \text{ is the maximal number with } \{(\mu, \sigma_1), \ldots, (\mu, \sigma_m)\} \subseteq R\} \)
Algorithm 1. Query Evaluation Procedure

**Input:** G: the active graph, which is an OWL 2 DL graph  
BGP: an OWL 2 DL BGP

**Output:** a multiset of solutions for evaluating BGP over G under OWL 2 Direct Semantics

1: \(O_G := \text{map}(G)\)
2: \(O_{BGP}^G := \text{map}(\text{BGP, } O_G)\)
3: \(\text{Axt} := \text{rewrite}(O_{BGP}^G)\) {create a list \(\text{Axt}\) of simplified axiom templates from \(O_{BGP}^G\)}
4: \(\text{Axt}^1, \ldots, \text{Axt}^m := \text{connectedComponents}(\text{Axt})\)
5: for \(j=1, \ldots, m\) do
6: \(R_j := \{(\mu_0, \sigma_0) \mid \text{dom}(\mu_0) = \text{dom}(\sigma_0) = 0\}\)
7: \(\text{axt}_1, \ldots, \text{axt}_n := \text{reorder}(\text{Axt}^i)\)
8: for \(i = 1, \ldots, n\) do

27: end for
28: end for
29: \(R := \{(\mu_1 \cup \ldots \cup \mu_m, \sigma_1 \cup \ldots \cup \sigma_m) \mid (\mu_j, \sigma_j) \in R_j, 1 \leq j \leq m\}\)
30: return \{(\mu, m) \mid m > 0 \text{ is the maximal number with } \{(\mu, \sigma_1), \ldots, (\mu, \sigma_m)\} \subseteq R\}
Algorithm 1. Query Evaluation Procedure

Input: $G$: the active graph, which is an OWL 2 DL graph  
        $BGP$: an OWL 2 DL BGP  

Output: a multiset of solutions for evaluating $BGP$ over $G$ under OWL 2 Direct Semantics

1: $O_G := \text{map}(G)$  
2: $O^G_{BGP} := \text{map}(BGP, O_G)$  
3: $Axt := \text{rewrite}(O^G_{BGP})$  
   {create a list $Axt$ of simplified axiom templates from $O^G_{BGP}$}  
4: $Axt^1, \ldots, Axt^m := \text{connectedComponents}(Axt)$  
5: for $j=1,\ldots, m$ do  
6: \hspace{1em} $R_j := \{(\mu_0, \sigma_0) \mid \text{dom}(\mu_0) = \text{dom}(\sigma_0) = \emptyset\}$  
7: \hspace{1em} $axt_1, \ldots, axt_n := \text{reorder}(Axt^j)$  
8: \hspace{1em} for $i = 1,\ldots, n$ do  
9: \hspace{2em} $\text{	extbullet For a simple axiom template, we then call a specialized reasoner method to retrieve entailed results (line 12).}$  
10: \hspace{2em} $\text{	extbullet Otherwise, we check which compatible solutions yield an entailed axiom (lines 13 to 24).}$  
9: \hspace{2em} end for  
8: \hspace{1em} end for  
7: $R := \{(\mu_1 \cup \ldots \cup \mu_m, \sigma_1 \cup \ldots \cup \sigma_m) \mid (\mu_j, \sigma_j) \in R_j, 1 \leq j \leq m\}$  
6: return $\{(\mu, m) \mid m > 0 \text{ is the maximal number with } \{(\mu, \sigma_1), \ldots, (\mu, \sigma_m)\} \subseteq R\}$
Proposed Optimizations:

1) Axiom Template Reordering:
   - The *simple axiom* templates are ordered by their cost, which is computed as the weighted sum of the estimated number of required consistency checks and the estimated result size. *(reasoner-depandan)*
   - The *complex templates* are ordered based only on the number of bindings that have to be tested.

2) Axiom Template Rewriting:
   - Some costly to evaluate axiom templates can be rewritten into axiom templates that can be evaluated more efficiently and yield an equivalent result.
   - Example:

     SubClassOf(?x ObjectIntersectionOf(ObjectSomeValuesFrom(:op ?y) :C))

     SubClassOf(?x :C) and SubClassOf(?x ObjectSomeValuesFrom(:op ?y))
Proposed Optimizations:

1) **Axiom Template Reordering:**
   - The *simple axiom* templates are ordered by their cost, which is computed as the weighted sum of the estimated number of required consistency checks and the estimated result size. *(reasoner-dependent)*
   - The *complex templates* are ordered based only on the number of bindings that have to be tested.

2) **Axiom Template Rewriting:**
   - Some costly to evaluate axiom templates can be rewritten into axiom templates that can be evaluated more efficiently and yield an equivalent result.
   - Example:
     
     ```
     SubClassOf(?x ObjectIntersectionOf(ObjectSomeValuesFrom(:op ?y) :C))
     
     SubClassOf(?x :C) and SubClassOf(?x ObjectSomeValuesFrom(:op ?y))
     ```
Proposed Optimizations:

1) Axiom Template Reordering:
   - The *simple axiom* templates are ordered by their cost, which is computed as the weighted sum of the estimated number of required consistency checks and the estimated result size. *(reasoner-dependant)*
   - The *complex templates* are ordered based only on the number of bindings that have to be tested.

2) Axiom Template Rewriting:

<table>
<thead>
<tr>
<th>Axiom Template</th>
<th>Rewriting</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClassAssertion(ObjectIntersectionOf(:C_1 \ldots :C_n) :a)</td>
<td>{ClassAssertion(:C_i :a)</td>
</tr>
<tr>
<td>SubClassOf(:C ObjectIntersectionOf(:C_1 \ldots :C_n))</td>
<td>{SubClassOf(:C :C_i)</td>
</tr>
<tr>
<td>SubClassOf(ObjectUnionOf(:C_1 \ldots :C_n) :C)</td>
<td>{SubClassOf(:C_i :C)</td>
</tr>
<tr>
<td>SubClassOf(ObjectSomeValuesFrom(:op owl:Thing :C))</td>
<td>ObjectPropertyDomain(:op :C)</td>
</tr>
<tr>
<td>SubClassOf(owl:Thing ObjectAllValuesFrom(:op :C))</td>
<td>ObjectPropertyRange(:op :C)</td>
</tr>
</tbody>
</table>
Proposed Optimizations (cont’d):

3) **Class-Property Hierarchy Exploitation:**
   - The hierarchies are stored in the reasoner’s internal structures.
   - Can be used to prune the search space of solutions in the evaluation of certain axiom templates.
   - Example:
     
     SubClassOf(:Infection ObjectSomeValuesFrom(:hasCausalLinkTo ?x))

     [If :C is not a solution and SubClassOf(:B :C) holds, then :B is also not a solution.]

4) **Exploiting the Domain and Range Restrictions:**
   - Can be exploited to further restrict the mappings for class variables.
   - Example:
     
     ObjectPropertyRange(:takesCourse :Course)
     SubClassOf(:GraduateStudent ObjectSomeValuesFrom(:takesCourse ?x))

     [Only the class :Course and its subclasses can be solutions for x and we can immediately prune other mappings]
Proposed Optimizations (cont’d):

3) *Class-Property Hierarchy Exploitation:*
   - The hierarchies are stored in the reasoner’s internal structures.
   - Can be used to prune the search space of solutions in the evaluation of certain axiom templates.
   - Example:
     \[ \text{SubClassOf(:Infection ObjectSomeValuesFrom(:hasCausalLinkTo ?x))} \]
     
     [If :c is not a solution and SubClassOf(:B :c) holds, then :B is also not a solution.]

4) *Exploiting the Domain and Range Restrictions:*
   - Can be exploited to further restrict the mappings for class variables.
   - Example:
     \[ \text{ObjectPropertyRange(:takesCourse :Course)} \]
     \[ \text{SubClassOf(:GraduateStudent ObjectSomeValuesFrom(:takesCourse ?x))} \]
     
     [Only the class :Course and its subclasses can be solutions for x and we can immediately prune other mappings]
• Since entailment regimes only change the evaluation of basic graph patterns, standard SPARQL algebra processors can be used that allow for custom BGP evaluation.
• Uses the ARQ library of the Jena Semantic Web Toolkit for parsing the query and for the SPARQL algebra operations apart from our custom BGP evaluation.
• The BGP is parsed and mapped into axiom templates by our extension of the OWL API.
• We use the HermiT reasoner for OWL reasoning.
Experimental Results:

Table 3. Query answering times in milliseconds for LUBM(1,0) and in seconds for the queries of Table 4 with and without optimizations

<table>
<thead>
<tr>
<th>LUMB(1, 0)</th>
<th>GALEN queries from Table 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td>Time</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>46</td>
</tr>
<tr>
<td>3</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>19</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>58</td>
</tr>
<tr>
<td>7</td>
<td>42</td>
</tr>
<tr>
<td>8</td>
<td>353</td>
</tr>
<tr>
<td>9</td>
<td>4,475</td>
</tr>
<tr>
<td>10</td>
<td>23</td>
</tr>
<tr>
<td>11</td>
<td>19</td>
</tr>
<tr>
<td>12</td>
<td>28</td>
</tr>
<tr>
<td>13</td>
<td>16</td>
</tr>
<tr>
<td>14</td>
<td>45</td>
</tr>
</tbody>
</table>

- Evaluate the 14 conjunctive ABox queries provided in the LUBM.
- Without optimization, queries 2, 7, 8, and 9 required 758.9 s, 14.7 s, >30 min, and >30 min, respectively.
Experimental Results:

Table 4. Sample complex queries for the GALEN ontology

<table>
<thead>
<tr>
<th></th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SubClassOf(:Infection ObjectSomeValuesFrom(:hasCausalLinkTo ?x))</td>
</tr>
<tr>
<td>2</td>
<td>SubClassOf(:Infection ObjectSomeValuesFrom(?y ?x))</td>
</tr>
<tr>
<td>3</td>
<td>SubClassOf(?x ObjectIntersectionOf(:Infection ObjectSomeValuesFrom(:hasCausalAgent ?y)))</td>
</tr>
<tr>
<td>4</td>
<td>SubClassOf(:NAMEDLigament ObjectIntersectionOf(:NAMEDInternalBodyPart ?x) SubClassOf(?x ObjectSomeValuesFrom(:hasShapeAnalagousTo ObjectIntersectionOf(?y ObjectSomeValuesFrom(?z :linear))))</td>
</tr>
<tr>
<td>5</td>
<td>SubClassOf(?x :NonNormalCondition) SubObjectPropertyOf(?z :ModifierAttribute) SubClassOf(:Bacterium ObjectSomeValuesFrom(?z ?w)) SubObjectProperty(?y :StatusAttribute) SubClassOf(?w :AbstractStatus) SubClassOf(?x ObjectSomeValuesFrom(?y :Status))</td>
</tr>
</tbody>
</table>
Conclusion & Future Work:

• A sound and complete query answering algorithm and novel optimizations for SPARQL’s OWL Direct Semantics entailment regime.

• Prototype combines existing tools; ARQ, the OWL API, and the HermiT.

• Apart from the query reordering optimization, the system is independent of the reasoner used.

• The optimizations can improve query execution time by up to three orders of magnitude.

• Future work will include the creation of more accurate cost estimates for the cost-based query reordering. *(The next paper on course webpage!)*

• The implementation of caching strategies that reduce the number of tests for different instantiations of a complex axiom template. *(No reported work yet!)*

• Although the proposed optimizations can significantly improve the query execution time, the required time can still be quite high. In practice, it is, therefore, advisable to add as many restrictive axiom templates for query variables as possible.
Conclusion & Future Work:

- A sound and complete query answering algorithm and novel optimizations for SPARQL’s OWL Direct Semantics entailment regime.
- Prototype combines existing tools; ARQ, the OWL API, and the HermiT.
- Apart from the query reordering optimization, the system is independent of the reasoner used.
- The optimizations can improve query execution time by up to three orders of magnitude.
- Future work will include the creation of more accurate cost estimates for the cost-based query reordering. *(the next paper on course webpage!)*
- The implementation of caching strategies that reduce the number of tests for different instantiations of a complex axiom template. *(no reported work yet!)*
- Although the proposed optimizations can significantly improve the query execution time, the required time can still be quite high. In practice, it is, therefore, advisable to add as many restrictive axiom templates for query variables as possible.
Thank You!