Answering Queries over OWL Ontologies with SPARQL

Ilianna Kollia Birte Glimm Ian Horrocks
presented by Mike Grove

OWL ED 2011
June 6, 2011

Introduction – A SPARQL Query Example

Example

Graph G: :Ilianna :owns :Tom.
:Tom rdf:type :Cat.
:owns rdfs:domain :Person.

Introduction – A SPARQL Query Example

Example

Graph G: :Ilianna :owns :Tom.
:Tom rdf:type :Cat.
:owns rdfs:domain :Person.

Query 1:

```
SELECT ?owner ?pet
WHERE {?owner :owns ?pet}
```

Introduction – A SPARQL Query Example

Example

Graph G: :Ilianna :owns :Tom.
:Tom rdf:type :Cat.
:owns rdfs:domain :Person.

Query 1: SELECT ?owner ?pet
WHERE {?owner :owns ?pet}

Answer: ?owner $\mapsto$:Ilianna, ?pet $\mapsto$:Tom

Introduction – A SPARQL Query Example

Example

Graph G: :Ilianna :owns :Tom.
:Tom rdf:type :Cat.
:owns rdfs:domain :Person.

Query 1: SELECT ?owner ?pet
WHERE {?owner :owns ?pet}

Answer: ?owner $\mapsto$:Ilianna, ?pet $\mapsto$:Tom

Query 2: SELECT ?person
WHERE {?person rdf:type :Person }

Introduction – A SPARQL Query Example

Example

Graph G: :Ilianna :owns :Tom.
:Tom rdf:type :Cat.
:owns rdfs:domain :Person.

Query 1: SELECT ?owner ?pet
WHERE {?owner :owns ?pet}

Answer: ?owner $\mapsto$:Ilianna, ?pet $\mapsto$:Tom

Query 2: SELECT ?person
WHERE {?person rdf:type :Person }

Answer: $\emptyset$

Introduction – A SPARQL Query Example

Example

Graph G: :Ilianna :owns :Tom.
:Tom rdf:type :Cat.
:owns rdfs:domain :Person.

Query 1: SELECT ?owner ?pet
WHERE {?owner :owns ?pet}

Answer: ?owner $\mapsto$:Ilianna, ?pet $\mapsto$:Tom

Query 2: SELECT ?person
WHERE {?person rdf:type :Person }

Answer: $\emptyset$

Query 2 needs RDFS or OWL entailment

Introduction – A SPARQL Query Example

Example

Graph G: :Ilianna :owns :Tom.
:Tom rdf:type :Cat.
:owns rdfs:domain :Person.

Query 1: SELECT ?owner ?pet
WHERE {?owner :owns ?pet}

Answer: ?owner $\mapsto$:Ilianna, ?pet $\mapsto$:Tom

Query 2: SELECT ?person
WHERE {?person rdf:type :Person }

Answer: $\emptyset$

The highlighted part is called a BGP (Basic Graph Pattern)

SPARQL 1.1 Entailment Regimes

SPARQL 1.1 defines several entailment regimes including

- RDF Entailment Regime
- RDFS Entailment Regime
- D-Entailment Regime
- OWL 2 RDF-Based Semantics Entailment Regime
- **OWL 2 Direct Semantics Entailment Regime**
- RIF Core Entailment

OWL Direct Semantics (OWL DS) Entailment Regime

- OWL axioms contain disjunctions
  - ⇒ the partial closure approach for RDFS does not work

OWL Direct Semantics (OWL DS) Entailment Regime

- OWL axioms contain disjunctions
  - ⇒ the partial closure approach for RDFS does not work
  - ⇒ axioms cannot be satisfied in a unique canonical model that could be used to answer queries

OWL Direct Semantics (OWL DS) Entailment Regime

- OWL axioms contain disjunctions
  - ⇒ the partial closure approach for RDFS does not work
  - ⇒ axioms cannot be satisfied in a unique canonical model that could be used to answer queries
- OWL contains existential restrictions

OWL Direct Semantics (OWL DS) Entailment Regime

- OWL axioms contain disjunctions
  - $\Rightarrow$ the partial closure approach for RDFS does not work
  - $\Rightarrow$ axioms cannot be satisfied in a unique canonical model that could be used to answer queries
- OWL contains existential restrictions
  - $\Rightarrow$ introduction of new individuals by OWL reasoners

OWL Direct Semantics (OWL DS) Entailment Regime

- OWL axioms contain disjunctions
  - ⇒ the partial closure approach for RDFS does not work
  - ⇒ axioms cannot be satisfied in a unique canonical model that could be used to answer queries
- OWL contains existential restrictions
  - ⇒ introduction of new individuals by OWL reasoners
  - ⇒ models of an ontology can be infinite

OWL Direct Semantics (OWL DS) Entailment Regime

- OWL axioms contain disjunctions
  - ⇒ the partial closure approach for RDFS does not work
  - ⇒ axioms cannot be satisfied in a unique canonical model that could be used to answer queries
- OWL contains existential restrictions
  - ⇒ introduction of new individuals by OWL reasoners
  - ⇒ models of an ontology can be infinite
- OWL DS is defined in terms of structural objects

OWL Direct Semantics (OWL DS) Entailment Regime

- OWL axioms contain disjunctions
  - ⇒ the partial closure approach for RDFS does not work
  - ⇒ axioms cannot be satisfied in a unique canonical model that could be used to answer queries
- OWL contains existential restrictions
  - ⇒ introduction of new individuals by OWL reasoners
  - ⇒ models of an ontology can be infinite
- OWL DS is defined in terms of structural objects
- The mapping from an RDF graph G to structural objects can be done if the RDF graph is *well-formed*

OWL Direct Semantics (OWL DS) Entailment Regime

Example

Graph G: :Ilianna :owns :Tom.
:Tom rdf:type :Cat.
:owns rdfs:domain :Person.

OWL Direct Semantics (OWL DS) Entailment Regime

Example

O(G): ObjectPropertyAssertion(:owns :Ilianna :Tom)
ClassAssertion(:Cat :Tom)
ObjectPropertyDomain(:owns :Person)

OWL Direct Semantics (OWL DS) Entailment Regime

Example

O(G): ObjectPropertyAssertion(:owns :Ilianna :Tom)
ClassAssertion(:Cat :Tom)
ObjectPropertyDomain(:owns :Person)

Q2: SELECT ?person
WHERE {?person rdf:type :Person }

OWL Direct Semantics (OWL DS) Entailment Regime

Example

O(G): ObjectPropertyAssertion(:owns :Ilianna :Tom)
ClassAssertion(:Cat :Tom)
ObjectPropertyDomain(:owns :Person)

Q2: SELECT ?person
WHERE {ClassAssertion(:Person ?person)}

OWL Direct Semantics (OWL DS) Entailment Regime

Example

O(G): ObjectPropertyAssertion(:owns :Ilianna :Tom)
ClassAssertion(:Cat :Tom)
ObjectPropertyDomain(:owns :Person)

Q2: SELECT ?person
WHERE {ClassAssertion(:Person ?person)}

The highlighted part is called an axiom template

OWL Direct Semantics (OWL DS) Entailment Regime

Example

O(G): ObjectPropertyAssertion(:owns :Ilianna :Tom)
ClassAssertion(:Cat :Tom)
ObjectPropertyDomain(:owns :Person)

Q2: SELECT ?person
WHERE {ClassAssertion(:Person ?person)}

Answer: ?person $\mapsto$:Ilianna

The highlighted part is called an axiom template

$$O(G) \models_{OWL-DS} \text{ClassAssertion(:Person :Ilianna)}$$

OWL Direct Semantics (OWL DS) Entailment Regime

Example

O(G): ObjectPropertyAssertion(:owns :Ilianna :Tom)
ClassAssertion(:Cat :Tom)
ObjectPropertyDomain(:owns :Person)

Q2: SELECT ?person
WHERE {ClassAssertion(:Person ?person)}

Answer: ?person $\mapsto$:Ilianna

The highlighted part is called an axiom template

- Some OWL modeling constructs correspond to several RDF triples, e.g., ObjectSomeValuesFrom, complex class expressions

Anonymous Individuals and Non-Distinguished Variables

Example

O(G1): ObjectPropertyAssertion(:owns :Ilianna _:someCat)
ClassAssertion(:Cat _:someCat)

O(G2): ClassAssertion(ObjectSomeValuesFrom(:owns :Cat)
:Ilianna)

Anonymous Individuals and Non-Distinguished Variables

Example

O(G1): ObjectPropertyAssertion(:owns :Ilianna _:someCat)
ClassAssertion(:Cat _:someCat)

O(G2): ClassAssertion(ObjectSomeValuesFrom(:owns :Cat)
:Ilianna)

$$O(G1) \equiv_{OWL-DS} O(G2)$$

Anonymous Individuals and Non-Distinguished Variables

Example

O(G1): ObjectPropertyAssertion(:owns :Ilianna _:someCat)
ClassAssertion(:Cat _:someCat)

O(G2): ClassAssertion(ObjectSomeValuesFrom(:owns :Cat)
:Ilianna)

Q3: SELECT ?cat
WHERE { ObjectPropertyAssertion(:owns ?owner ?cat) }

Anonymous Individuals and Non-Distinguished Variables

Example

O(G1): ObjectPropertyAssertion(:owns :Ilianna _:someCat)
ClassAssertion(:Cat _:someCat)

O(G2): ClassAssertion(ObjectSomeValuesFrom(:owns :Cat)
:Ilianna)

Q3: SELECT ?cat
WHERE { ObjectPropertyAssertion(:owns ?owner ?cat) }

O(G1): ?cat $\mapsto$ _:aCat

O(G2): $\emptyset$

SPARQL-OWL vs. SPARQL-DL

SPARQL-OWL

- Queries are very powerful - variables can occur within complex class expressions and can additionally bind to class or property names apart from individuals and literals

Example

ClassAssertion(ObjectSomeValuesFrom(:op ?x) ?y)

- Does not allow for proper non-distinguished variables

SPARQL-DL - implemented in the Pellet OWL reasoner

- Variables occur in places such that queries are mapped to standard reasoning tasks e.g. subclass retrieval
- Allows non-distinguished variables

SPARQL Features

- Up until now queries consisted of one BGP
- SPARQL also supports operators such as UNION for alternative selection criteria, OPTIONAL for optional bindings, or FILTERs

Example

```
SELECT ?x WHERE {   {?x :owns :Tom }
            UNION   {?x :owns ?y. ?y rdf:type :Dog } }
```

- BGP evaluation is the fundamental task that computes solutions
- All other features correspond to operations on the solutions computed by BGP evaluation

Answering SPARQL-OWL Queries

- Replace (map) variables with names from the ontology

Answering SPARQL-OWL Queries

- Replace (map) variables with names from the ontology
- The mapping has to preserve the types
  - class variables are mapped to class names,
  - object property variables to object property names,
  - individual variables to individual names
  - etc.

Answering SPARQL-OWL Queries

- Replace (map) variables with names from the ontology
- The mapping has to preserve the types
  - class variables are mapped to class names,
  - object property variables to object property names,
  - individual variables to individual names
  - etc.

  ⇒ compatible mappings

ANSWERING SPARQL-OWL QUERIES

- Replace (map) variables with names from the ontology
- The mapping has to preserve the types
  - class variables are mapped to class names,
  - object property variables to object property names,
  - individual variables to individual names
  - etc.

  $\Rightarrow$ compatible mappings
- Use an OWL reasoner to check, for each compatible mapping, whether the instantiated ontology is entailed by the queried ontology

Answering SPARQL-OWL Queries

- Replace (map) variables with names from the ontology
- The mapping has to preserve the types
  - class variables are mapped to class names,
  - object property variables to object property names,
  - individual variables to individual names
  - etc.

  $\Rightarrow$ compatible mappings
- Use an OWL reasoner to check, for each compatible mapping, whether the instantiated ontology is entailed by the queried ontology
- Such algorithm would perform n^m entailment checks, where n:number of individuals, m:number of variables
  $\Rightarrow$ exponential in the number of variables in the query

Answering SPARQL-OWL Queries

- Replace (map) variables with names from the ontology
- The mapping has to preserve the types
  - class variables are mapped to class names,
  - object property variables to object property names,
  - individual variables to individual names
  - etc.

  $\Rightarrow$ compatible mappings
- Use an OWL reasoner to check, for each compatible mapping, whether the instantiated ontology is entailed by the queried ontology
- Such algorithm would perform n^m entailment checks, where n:number of individuals, m:number of variables
  $\Rightarrow$ exponential in the number of variables in the query
- More efficient to evaluate axiom by axiom in a “good” order

Cost-Based Ordering Example

Example

(1) ?x rdf:type :A.
(2) ?x :op ?y

Assumptions:

- 100 individuals, one of them in :A
- The :A instance has 1 :op-successor
- :op has 200 instances

Cost-Based Ordering Example

Example

(1) ?x rdf:type :A.
(2) ?x :op ?y

Assumptions:

- 100 individuals, one of them in :A
- The :A instance has 1 :op-successor
- :op has 200 instances

Order (1) → (2)	Total=200 mappings to be tested
?x rdf:type :A.	test 100 possible mappings, 1 satisfies template
?x :op ?y.	test 100 possible mappings, 1 satisfies template

Order (2) → (1)	Total=10200 mappings to be tested
?x :op ?y.	test 100 * 100 possible mappings, 200 satisfy template
?x rdf:type :A	test 200 possible mappings, 1 satisfies template

Use Dedicated Reasoner Tasks

- Use reasoner to retrieve solutions instead of checking entailment

Example

BGP : ?x rdfs:subClassOf :C

- Use highly optimised methods of reasoners to retrieve the subclasses instead of checking entailment for each possible mapping
- If the class hierarchy is precomputed only a cache lookup is enough to find the solutions

Simple and Complex Axiom Templates

- *Simple axiom templates* - correspond to dedicated reasoning tasks
- *Complex axiom templates* - are evaluated by iterating over the compatible mappings and by checking entailment for each instantiated axiom template

Example

Simple: SubClassOf(?x :C)
TransitiveObjectProperty(?x)
ObjectPropertyRange(:op ?y)

Complex: SubClassOf(:C ObjectIntersectionOf(?z
ObjectSomeValuesFrom(?x ?y)))
ClassAssertion(ObjectSomeValuesFrom(:op ?x) ?y)

Axiom Template Rewriting

Intuition: Rewrite costly complex axiom templates into equivalent ones that can be evaluated more efficiently

Example

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

- Requires a quadratic number of consistency checks in the number of classes in the ontology (?x, ?y are class variables)

Axiom Template Rewriting

Intuition: Rewrite costly complex axiom templates into equivalent ones that can be evaluated more efficiently

Example

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

Rewritten: (1) SubClassOf(?x :C)
(2) SubClassOf(?x ObjectSomeValuesFrom(:op ?y))

- Requires a quadratic number of consistency checks in the number of classes in the ontology (?x, ?y are class variables)
- (1) requires a cheap cache lookup (assuming that the class hierarchy is precomputed)
- (2) requires an entailment check for the usually few resulting bindings for ?x combined with all other class names for ?y

Axiom Template Reordering

- Complex axiom templates can only be evaluated with costly entailment checks
  - $\Rightarrow$ We evaluate simple axiom templates first
- Cost of simple templates: weighted sum of the estimated number of required consistency checks and the estimated result size
  - Estimates are based on statistics provided by the reasoner
  - In case it cannot give estimates we work with explicitly stated information
- Cost of complex axiom templates: ordered based on the number of bindings that have to be tested, i.e. the number of needed consistency checks

Class-Property Hierarchy Exploitation

- Cached hierarchies can be used to prune the search space of solutions in the evaluation of certain axiom templates

Example

SubClassOf(:Infection
ObjectSomeValuesFrom(:hasCausalLinkTo ?x))

Class-Property Hierarchy Exploitation

- Cached hierarchies 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.

Class-Property Hierarchy Exploitation

- Cached hierarchies 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.
- Thus, when searching for solutions for ?x, we traverse the class hierarchy topdown

Class-Property Hierarchy Exploitation

- Cached hierarchies 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.
- Thus, when searching for solutions for ?x, we traverse the class hierarchy topdown
- When we find a non-solution :C, we prune the subtree of the class hierarchy rooted in :C

Class-Property Hierarchy Exploitation

- Cached hierarchies 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.
- Thus, when searching for solutions for ?x, we traverse the class hierarchy topdown
- When we find a non-solution :C, we prune the subtree of the class hierarchy rooted in :C
- Queries over ontologies with many classes and deep hierarchies can gain the maximum advantage from this optimization

Exploiting the Domain and Range Restrictions

- The explicit and/or inferred domains and ranges of properties in the queried ontology can be used to reduce the number of required entailment checks

Example

O(G): ObjectPropertyRange(:takesCourse :Course)
BGP: SubClassOf(:GraduateStudent
    ObjectSomeValuesFrom(:takesCourse ?x))

- In case at least one solution mapping exists for ?x, the class :Course and its super-classes can immediately be considered solution mappings for ?x

Algorithm Overview

1. Map the graph and BGP to OWL structural objects (possibly with variables)
2. Rewrite axioms templates
3. Order the axiom templates
4. Evalute simple axiom templates and prune untested solutions
5. Evaluate complex axiom templates and prune untested solutions

SYSTEM ARCHITECTURE

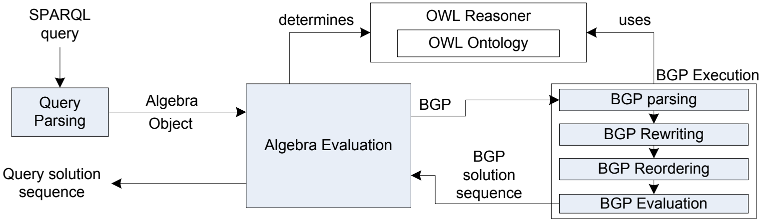

FIGURE: The main phases of query processing in our system

Evaluation – LUBM(1,0)

- LUBM queries belong to the class of conjunctive ABox queries
- LUBM ontology contains 43 classes, 25 object properties and 7 data properties
- LUBM(1,0) contains 16283 individuals and 8839 literals
- The ontology took 3.8 s to load and 22.7 s for classification and realization
- The reordering optimization had the biggest impact on queries 2,7,8 and 9

Query	Time in ms
1	20
2	46
3	19
4	19
5	32
6	58
7	42
8	353
9	4,475
10	23
11	19
12	28
13	16
14	45

Evaluation – GALEN

- The Galen ontology consists of 2 748 classes and 413 object properties
- The ontology took 1.6 s to load and 4.8 s for classification

Evaluation – GALEN

- The Galen ontology consists of 2 748 classes and 413 object properties
- The ontology took 1.6 s to load and 4.8 s for classification

Query 1

SubClassOf(:Infection
ObjectSomeValuesFrom(:hasCausalLinkTo ?x))

- Time without optimizations: 2.1 s
- Time with Hierarchy Exploitation 0.1 s

Evaluation – GALEN

- The Galen ontology consists of 2 748 classes and 413 object properties
- The ontology took 1.6 s to load and 4.8 s for classification

Query 2

SubClassOf(:Infection
    ObjectSomeValuesFrom(?y ?x))

- Time without optimizations: 780.6 s
- Time with Hierarchy Exploitation 4.4 s

EVALUATION – GALEN

- The Galen ontology consists of 2 748 classes and 413 object properties
- The ontology took 1.6 s to load and 4.8 s for classification

QUERY 3

```
SubClassOf( ?x
            ObjectIntersectionOf(:Infection
                  ObjectSomeValuesFrom(:hasCausalAgent ?y)))
```

- Time without optimizations: $>$30 min
- Time with Hierarchy Exploitation: 119.6 s
- Time with Rewriting: 204.7 s
- Time with Hierarchy Exploitation and Rewriting: 4.9 s

Evaluation – GALEN

Query 5

SubClassOf(?x :NonNormalCondition)
SubClassOf(:Bacterium ObjectSomeValuesFrom(?z ?w))
SubClassOf(?w :AbstractStatus)
SubClassOf(?x ObjectSomeValuesFrom(?y :Status))
SubObjectPropertyOf(?z :ModifierAttribute)
SubObjectPropertyOf(?y :StatusAttribute)

- Time with Reordering or Hierarchy Exploitation: $>$30 min
- Time with Reordering and Hierarchy Exploitation: 5.6 s
  - $\Rightarrow$ Add as many as possible restrictive axiom templates for query variables

Conclusions

- An outline of a query answering algorithm and novel optimizations have been presented for SPARQL's OWL Direct Semantics Entailment Regime
- Our prototypical system uses existing tools such as ARQ, the OWL API and the HermiT OWL reasoner
- Apart from the query reordering optimization which uses statistics provided by HermiT, the system is independent of the reasoner used
- We evaluated the algorithm and the proposed optimizations on the LUBM benchmark and on a custom benchmark containing queries that make use of the very expressive features of the regime
- The optimizations can improve query answering time by up to three orders of magnitude

QUESTIONS?

Contact information of the authors:

- b.glimm@cs.ox.ac.uk
- ilianna2@mail.ntua.gr