Implementing an Inference Engine for RDFS/OWL Constructs and User-Defined Rules in Oracle

Zhe Wu, George Eadon, Souripriya Das, Eugene Inseok Chong, Vladimir Kolovski, Melliyal Annamalai, Jagannathan Srinivasan

ICDE 2008
Our Data Model

- Inference Engine for Semantic Web Data
- RDF: Labeled, directed graph, which may contain cycles
- Edges are called triples
  \[
  \langle \text{subject}, \text{property}, \text{object} \rangle
  \]
- Triples represent facts
  \[
  \langle :\text{ICDE2008}, :\text{beginsOn}, \text{“April_07_2008”} \rangle
  \langle :\text{ICDE2008}, :\text{locatedIn}, :\text{Cancun} \rangle
  \langle :\text{Cancun}, :\text{inCountry}, :\text{Mexico} \rangle
  \]
How is RDF Related To Inference?

• W3C defines vocabularies and semantics for use with RDF.
  • RDFS (“RDF Schema”) introduces a basic type system including class hierarchy.
    
    <:Conference, rdfs:subClassOf, :Gathering>
    
    <:ICDE2008,
    rdf:type, :Conference>
  • OWL (“Web Ontology Language”) has richer semantics.
    
    <:ICDE2008,
    owl:SameAs,
    24th_International_Conference_on_Data_Engineering>
  • Three OWL variants: OWL Lite, OWL DL, and OWL Full.

• Users have their own rules for their custom semantics.
• Inference lets you merge different data sources.
• Infer hidden facts (and inconsistencies) from existing facts.
Oracle Support for RDF

- Oracle 10gR2 (2005) supports RDF storage, query, and RDFS inference with user-defined rules.

- Oracle 11g (2007) introduces a scalable, efficient, forward-chaining based inference engine that supports a subset of OWL DL called OWL Prime.
  - Need to support 100s of millions of triples, and beyond
    - No existing reasoner handles complete DL semantics at this scale
      - Neither Pellet nor KAON2 can handle LUBM10 or ST ontologies on a setup of 64 Bit machine, 4GB Heap¹
  - Implemented as a database application

¹ The summary Abox: Cutting Ontologies Down to Size. ISWC 2006
OWL Subsets Supported

- **Three subsets for different applications**
  - RDFS++
    - RDFS plus owl:sameAs and owl:InverseFunctionalProperty
  - OWLSIF (OWL with IF semantics)
    - Based on Dr. Horst’s pD* vocabulary¹
  - OWLPrime
    - rdfs:subClassOf, subPropertyOf, domain, range
    - owl:TransitiveProperty, SymmetricProperty, FunctionalProperty, InverseFunctionalProperty,
    - owl:inverseOf, sameAs, differentFrom
    - owl:disjointWith, complementOf,
    - owl:hasValue, allValuesFrom, someValuesFrom
    - owl:equivalentClass, equivalentProperty

- **Jointly determined with domain experts, customers and partners**

¹ Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary
Semantics Characterized by Entailment Rules

- RDFS has 14 entailment rules defined in the specification.
  - E.g. rule:
    
    ```
    aaa  rdfs:domain XXX .
    uuu   aaa             yyy .  \(\Rightarrow\)  uuu rdf:type XXX .
    ```

- OWLPrime has 50+ entailment rules.
  - E.g. rule:
    
    ```
    aaa  owl:inverseOf  bbb  .
    bbb  rdfs:subPropertyOf  ccc  .
    ccc  owl:inverseOf  ddd  .  \(\Rightarrow\)  aaa rdfs:subPropertyOf ddd .
    xxx  owl:disjointWith yyy .
    a    rdf:type            xxx .  \(\Rightarrow\)  a owl:differentFrom b .
    b    rdf:type            yyy .
    ```

- These rules have efficient implementations in RDBMS.
Soundness

• Soundness of our OWL Prime rules verified through
  • Comparison with other well-tested reasoners
  • Proof generation
    • A proof of an assertion consists of a rule (name), and a set of assertions which together deduce that assertion.
  • Option “PROOF=T” instructs 11g OWL to generate proof

\[
\begin{align*}
\text{TripleID1} & : \text{emailAddress} \quad \text{rdf:type} \quad \text{owl:InverseFunctionaProperty} . \\
\text{TripleID2} & : \text{John} \quad \text{:emailAddress} \quad :\text{John\_at\_yahoo\_dot\_com} . \\
\text{TripleID3} & : \text{Johnny} \quad \text{:emailAddress} \quad :\text{John\_at\_yahoo\_dot\_com} . \\
:\text{John} & \quad \text{owl:sameAs} \quad :\text{Johnny} \quad (\text{proof := TripleID1, TripleID2, TripleID3, “IFP”})
\end{align*}
\]
Option 1: add user-defined rules

- Both 10gR2 RDF and 11g RDF/OWL supports user-defined rules in this form (filter is supported)

- We allow arbitrary user-defined rules
  - There’s no negation or NOT EXISTS – introducing new triples never revokes existing triples.
  - Rules (user-defined and built-in) introduce new edges in the graph, but do not create new nodes or labels.
Option 2: Separation in TBox and ABox reasoning

- TBox (Schema) tends to be small in size
  - Generate a class subsumption tree using complete DL reasoners (like Pellet, KAON2, Fact++, Racer, etc)
- ABox (Data/Individuals) can be arbitrarily large
  - Use Oracle OWL to infer new knowledge based on the class subsumption tree from TBox
Avoid incremental index maintenance for efficient INSERT

Partition data for efficient query

Maintain up-to-date statistics

Upon reaching closure, copy results to a new “exchange” table, create appropriate indexes, and exchange into IdTriplesTable.
Translating Rules to SQL

- **Rule**
  \[ ?u \text{rdfs:subClassOf} \ ?x \rightarrow ?v \text{rdf:type} \ ?x \]
  \[ ?v \text{rdf:type} \ ?u \]

- **SQL**
  
  ```sql
  select distinct T2.SID sid,
          ID(rdf:type) pid, T1.OID oid
  from (IVIEW) T1, (IVIEW) T2
  where T1.PID = ID(rdfs:subClassOf)
  AND T2.PID = ID(rdf:type)
  AND T1.SID = T2.OID
  AND NOT EXISTS
    (select null from (IVIEW) m
     where m.SID = T2.SID
     AND m.PID = ID(rdf:type)
     AND m.OID = t1.OID)
  ```
Translating Rules to SQL

- **Rule**
  \[ ?u \text{rdfs:subClassOf} ?x \Rightarrow ?v \text{rdf:type} ?x \]
  \[ ?v \text{rdf:type} ?u \]

- **SQL**
  
  ```sql
  select distinct T2.SID sid,
           ID(rdf:type) pid, T1.OID oid
  from (IVIEW) T1, (IVIEW) T2
  where T1.PID = ID(rdfs:subClassOf)
       AND T2.PID = ID(rdf:type)
       AND T1.SID = T2.OID
       AND NOT EXISTS
          (select null from (IVIEW) m
           where m.SID = T2.SID
                AND m.PID = ID(rdf:type)
                AND m.OID = t1.OID)
  ```
Translating Rules to SQL

- **Rule**
  \[ ?u \text{ rdfs:subClassOf } ?x \rightarrow ?v \text{ rdf:type } ?x \]
  \[ ?v \text{ rdf:type } ?u \]

- **SQL**
  ```sql
  select distinct T2.SID sid,
                 ID(rdf:type) pid, T1.OID oid
  from (IVIEW) T1, (IVIEW) T2
  where T1.PID = ID(rdfs:subClassOf)
  AND T2.PID = ID(rdf:type)
  AND T1.SID = T2.OID
  AND NOT EXISTS
       (select null from (IVIEW) m
        where m.SID = T2.SID
        AND m.PID = ID(rdf:type)
        AND m.OID = t1.OID)
  ```
Translating Rules to SQL

- **Rule**
  
  \[ ?u \ rdfs:subClassOf \ ?x \rightarrow ?v \ rdf:type \ ?x \]
  \[ ?v \ rdf:type \ ?u \]

- **SQL**
  
  ```sql
  select distinct T2.SID sid,
                   ID(rdf:type) pid, T1.OID oid
  from (IVIEW) T1, (IVIEW) T2
  where T1.PID = ID(rdfs:subClassOf)
       AND T2.PID = ID(rdf:type)
       AND T1.SID = T2.OID
       AND NOT EXISTS
         (select null from (IVIEW) m
          where m.SID = T2.SID
             AND m.PID = ID(rdf:type)
             AND m.OID = T1.OID)
  ```
Translating Rules to SQL

- **Rule**
  \[ \text{?u rdfs:subClassOf } \text{?x } \Rightarrow \text{?v rdf:type } \text{?x} \]
  \[ \text{?v rdf:type } \text{?u} \]

- **SQL**
  
  ```sql
  select distinct T2.SID sid,
           ID(rdf:type) pid, T1.OID oid
  from (IVIEW) T1, (IVIEW) T2
  where T1.PID = ID(rdfs:subClassOf)
  AND T2.PID = ID(rdf:type)
  AND T1.SID = T2.OID
  AND NOT EXISTS
    (select null from (IVIEW) m
     where m.SID = T2.SID
     AND m.PID = ID(rdf:type)
     AND m.OID = t1.OID)
  ```
Translating Rules to SQL with Proof

- Rule
  \[ \text{?u rdfs:subClassOf ?x } \quad \rightarrow \quad \text{?v rdf:type ?x} \]
  \[ \text{?v rdf:type ?u} \]

- SQL
  \[
  \text{select T2.SID, ID(rdf:type), T1.OID oid,}
  \min(T1.TripleID||' '||T2.TripleID||': RDFS9')
  \text{from (IVIEW) T1, (IVIEW) T2}
  \text{where T1.PID = ID(rdfs:subClassOf)}
  \text{AND T2.PID = ID(rdf:type)}
  \text{AND T1.SID = T2.OID}
  \text{AND NOT EXISTS}
  \text{(select null from (IVIEW) m}
  \text{where m.SID = T2.SID}
  \text{AND m.PID = ID(rdf:type)}
  \text{AND m.OID = t1.OID)}
  \text{group by T2.SID, T1.OID}
  \]
Transitive Closure

- Builtin rules are hand optimized – see paper for details.
- Computing transitive closure is a common operation.
- Example:
  
  ```
  ?a owl:sameAs ?b \rightarrow ?a owl:sameAs ?c
  ?b owl:sameAs ?c
  ```

- Our system optionally finds length of the shortest path connecting two nodes.
- Oracle has CONNECT BY for hierarchical queries
  - Finds “all-pairs, all-paths” – we only need “all-pairs, shortest-path.”
- Iterative procedural approach is preferred.
Transitive Closure Alternatives

<table>
<thead>
<tr>
<th>Approach</th>
<th>Iterative Step</th>
<th># of Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>naïve</td>
<td>( {1..2^i} \times {1..2^i} \Rightarrow {2 \ldots 2^i, 2^i+1 \ldots 2^{i+1}} )</td>
<td>logarithmic</td>
</tr>
<tr>
<td>smart</td>
<td>( {1..2^i} \times {2^i} \Rightarrow {2^i+1 \ldots 2^{i+1}} )</td>
<td>logarithmic</td>
</tr>
<tr>
<td>semi-naïve</td>
<td>( {1} \times {i} \Rightarrow {i+1} )</td>
<td>linear</td>
</tr>
</tbody>
</table>

- We use “semi-naïve”.
  - Requires more iterations, but each iteration is cheaper.
  - Buffer-cache friendly.
  - 47x faster than naïve.

- Partition by path-length for semi-naïve and smart approaches, no index required.

---

\(^1\) Y.E. Ioannidis, “On the Computation of the Transitive Closure of Relational Operators”, VLDB 1986
Experimental Setup

- Linux based PC (1 CPU, 3GHz, 2GB RAM)
- One database machine, plus two machines serving storage via NFS over private gigabit network.
- Two Datasets:
  - LUBM (“Lehigh University Benchmark”): a synthesized dataset modeling an university domain.
  - LUBM\(<N>\) represents \(N\) different universities
  - UniProt (“Universal Protein Resource”): a repository of protein data.
## Inference Performance

<table>
<thead>
<tr>
<th>Ontology ( Millions of Triples)</th>
<th>RDFS # Triples Inferred (Millions)</th>
<th>RDFS Time</th>
<th>OWLPrime # Triples Inferred (Millions)</th>
<th>OWLPrime Time</th>
<th>OWLPrime + Used-Defined Rules # Triples Inferred (Millions)</th>
<th>OWLPrime + Used-Defined Rules Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUBM50 (6.65)</td>
<td>1.08</td>
<td>9 min 14 sec</td>
<td>3.25</td>
<td>9 min 53 sec</td>
<td>3.34</td>
<td>24 min 16 sec</td>
</tr>
<tr>
<td>LUBM500 (69.64)</td>
<td>11.33</td>
<td>2 hr 59 min</td>
<td>31.93</td>
<td>6 hr 52 min</td>
<td>32.83</td>
<td>9 hr 30 min</td>
</tr>
<tr>
<td>LUBM1000 (133.61)</td>
<td>21.73</td>
<td>6 hr 34 min</td>
<td>61.25</td>
<td>14 hr 42 min</td>
<td>62.89</td>
<td>18 hr 31 min</td>
</tr>
<tr>
<td>UniProt (5.00)</td>
<td>0.88</td>
<td>8 min</td>
<td>3.4</td>
<td>5 min 42 sec</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### Graph

- **Input Size (Millions of Triples)**
- **Inference Time (minutes)**

- **RDFS**: Blue Diamonds
- **OWL'**: Green Diamonds
- **OWL' & Rules**: Red Triangles

---

*Oracle*
Since We Wrote the Paper...

- Tuned the database
- LUBM8000: 1 Billion + Triples
  - One 3GHz CPU (as before)
  - 4 GB RAM, local disks

<table>
<thead>
<tr>
<th>Ontology (Millions of Triples)</th>
<th>OWLPrime</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># Triples Inferred (Millions)</td>
</tr>
<tr>
<td>LUBM50 (6.65)</td>
<td>3.25</td>
</tr>
<tr>
<td>LUBM1000 (133.61)</td>
<td>61.25</td>
</tr>
<tr>
<td>LUMB8000 (1,068.00)</td>
<td>521.7</td>
</tr>
</tbody>
</table>

Inference Time (hours)

Input Size (Millions of Triples)
Summary

• Implemented forward-chaining inference engine for Semantic Web data as database application.
  • Optionally generate proofs
  • Optionally compute path length for transitive properties
  • Supports user-defined rules in addition to builtins

• Carefully constructed a subset of OWL DL called OWL Prime, and builtin rules for OWL Prime semantics.
  • Support OWL constructs that users need.
  • Scalable and efficient inference.

• Performance evaluation using LUBM data demonstrates scalability claim.
Future Work

• Implement more rules to cover richer semantics
  • OWL 1.1 Rule-Based Fragment

• Seek a standardization of the set of rules.
  • To promote interoperability among vendors

• Further improve inference performance

• Look into incremental maintenance as explicit data changes
For More Information

http://search.oracle.com

semantic technologies

or

http://www.oracle.com/