Implementing OWL 2 RL and OWL 2 QL rule-sets for OWLIM

Barry Bishop, Spas Bojanov

OWLED 2011, San Francisco, 06/06/2011

Ontotext

Ontotext is a Sirma Group company

- **Semantic technology developer**[1] established in 2000
- **Global leader** in semantic databases and semantic annotation
- **Staff: 55** employees plus contractors
- **Unique technology** portfolio:
  - **Semantic Databases**: high-performance RDF DBMS, scalable reasoning
  - **Semantic Search**: text-mining (IE), Information Retrieval (IR)
  - **Web Mining**: focused crawling, screen scraping, data fusion
  - **Web Services and BPM**: WS annotation, discovery, etc.

[1] http://www.ontotext.com/

OWLIM

- OWLIM[1] is a family of semantic repositories
  - SwiftOWLIM and BigOWLIM
  - Online user documentation[2]
- Storage and Inference Layer (SAIL) for Sesame
  - Compatible with most RDF syntaxes
- RDF storage, reasoner, query-engine
  - Forward chaining rule-entailment
  - SPARQL and SeRQL query languages

[1] http://www.ontotext.com/owlim
[2] http://owlim.ontotext.com

SwiftOWLIM

- Free to use
  - Partly open-source, but this is changing
- In memory
  - Scales to tens of millions of statements on desktop hardware
  - Persistence at shutdown/start-up
- Very fast
  - Forward chaining rule-based reasoning

BigOWLIM

- Commercially licensed
  - Enterprise grade RDF database
- File-based
  - Scales to tens of billions of statements on basic server
- Advanced features
  - Incremental retraction (without truth maintenance)
  - Full-text search
  - Geo-spatial extensions
  - RDF Rank
  - **`owl:sameAs`** optimisation
  - Replication cluster

OWLIM – Rule Language

- R-entailment (ter Horst)
  - Premises and conclusions are triple patterns
  - Variables allowed in any position
  - Inequality constraints
  - Rules applied directly to RDF graph

- Example

```
Id: prp_fp
  p <rdf:type> <owl:FunctionalProperty>
  x p y1                        [Constraint y1 != y2]
  x p y2
  -------------------------------
  y1 <owl:sameAs> y2
```

OWL2 RL

- OWL2 profile
  - Syntactic subset of OWL2
  - Scalable, expressive
  - RDF-based semantics defined by first order implications[1]
  - Designed to be amenable to implementation on rule-engines

- Straightforward to implement on OWLIM?
  - Problem 1: Data-type reasoning
  - Problem 2: Rules that use lists (RDF collections[2])

[1] http://www.w3.org/TR/owl2-profiles/#Reasoning_in_OWL_2_RL_and_RDF_Graphs_using_Rules
[2] http://www.w3.org/TR/rdf-syntax/#collections

OWL2 RL – data-type rules

- Data-type rules need special programming
  - Efficient implementation not obvious for a forward-chaining reasoner, e.g. dt-diff:

⇒ `T(lt₁, owl:differentFrom, lt₂)`

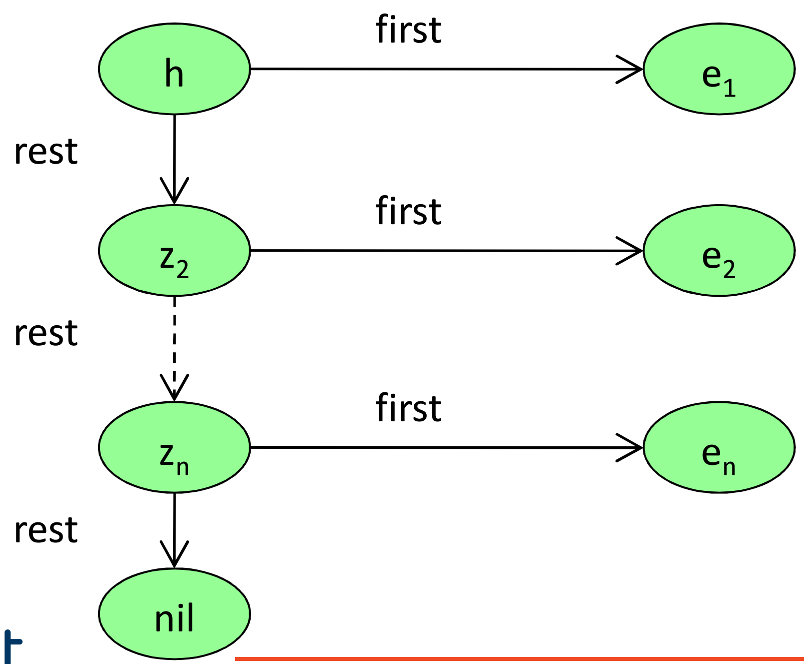
for all literals lt_1 and lt_2 with different data values

OWL2 RL – rules that use lists

- 12 rules use LIST[h, e1, ..., en]

$T(h,\ rdf{:}first,\ e_1)$
$T(z_2,\ rdf{:}first,\ e_2)$
...
$T(z_n,\ rdf{:}first,\ e_n)$

$T(h,\ rdf{:}rest,\ z_2)$
$T(z_2,\ rdf{:}rest,\ z_3)$
...
$T(zn,\ rdf{:}rest,\ rdf{:}nil)$



OWL2 RL – List Rule Examples

- cls-int2:

```
T(?c, owl:intersectionOf, ?x)        T(?y, rdf:type, ?c1)
LIST[?x, ?c1, ..., ?cn]        =>    T(?y, rdf:type, ?c2)
T(?y, rdf:type, ?c)                  ...
                                     T(?y, rdf:type, ?cn)
```

- prp-spo2:

```
T(?p, owl:propertyChainAxiom, ?x)
LIST[?x, ?p1, ..., ?pn]
T(?u1, ?p1, ?u2)
T(?u2, ?p2, ?u3)                =>   T(?u1, ?p, ?un+1)
...
T(?un, ?pn, ?un+1)
```

OWL2 RL – List rule solution

- One solution: Pre-process for a specific ontology
  - If the ontology is known, then rules can be re-written
  - e.g. given:

```
T(?c, owl:intersectionOf, _:b)
LIST[_:b1, :Car, :LuxuryThing]
```

  - Create the rule:

```
T(?y, rdf:type, :SuperCar)  ⇒  T(?y, rdf:type, :Car)
                               T(?y, rdf:type, :LuxuryThing)
```

- Constraints
  - Requires extra processing stage
  - Assumes a fixed ontology

OWL2 RL

- (Infinite) set of OWLIM rules?

```
T(?c, owl:intersectionOf, ?x)
T(?x, rdf:first, ?c1)
T(?x, rdf:rest, rdf:nil)
T(?y, rdf:type, ?c)
```

⇒

```
T(?y, rdf:type, ?c1)
```

(Intersection of a single class)

```
T(?c, owl:intersectionOf, ?x)
T(?x, rdf:first, ?c1)
T(?x, rdf:rest, ?x2)
T(?x2, rdf:first, ?c2)
T(?x2, rdf:rest, rdf:nil)
T(?y, rdf:type, ?c)
```

⇒

```
T(?y, rdf:type, ?c1)
T(?y, rdf:type, ?c2)
```

(Intersection of two classes)

  - And so on for 3, 4, 5, … classes

- Only practical for short lists

OWL2 RL

- Different construction of rules
  - Make necessary intermediate inferences to cope with lists of any length
  - Would allow the ontology to change

- Rule Interchange Format (RIF) working group have done this[1]
  - Translation of OWL2 RL rule implications to RIF Core
  - Starting point for OWLIM implementation

[1] http://www.w3.org/TR/rif-owl-rl/#Appendix:_OWL_2_RL_ruleset_-_presentation_syntax

RIF – Example

- prp-spo2 in RIF Core using auxiliary predicate

```
(* <#prp-spo2> *)
Forall ?p ?last ?pc ?start (
  ?start[?p->?last] :- And (
      ?p[owl:propertyChainAxiom->?pc]
      _checkChain(?start ?pc ?last) ))

Forall ?start ?pc ?last ?p ?tl (
  _checkChain(?start ?pc  ?last) :- And (
      ?pc[rdf:first->?p rdf:rest->?tl]
      ?start[?p->?next]
      _checkChain(?next ?tl ?last) ))

Forall ?start ?pc ?last ?p (
  _checkChain(?start ?pc  ?last) :- And (
      ?pc[rdf:first->?p rdf:rest->rdf:nil]
      ?start[?p->?last] ))
```

RIF – Example

- Auxiliary predicate _checkChain
  - Ternary predicate
  - Used to infer relationship from all 'links' in a chain to the end
  - If chain is complete then infer property chain property from first to last individual

- However, OWLIM (R-entailment) applies rules directly to RDF statements
  - No auxiliary predicates

OWLIM – auxiliary predicate solution

- OWLIM is a quad store
  - It stores graph name for every triple
  - Called 'context' in Sesame
- By using quads it is possible to store
  - The name of an auxiliary ternary predicate
  - The three members of tuples
- Solution
  - Extend OWLIM rule language to specify context
  - Hide these 'special' RDF statements from the query engine

OWLIM – Rule language extensions for context

- prp-spo2 in OWLIM

```
Id: prp_spo2_1
    p <owl:propertyChainAxiom> pc
    start pc last                    [Context <onto:_checkChain>]
    ----------------------------
    start p last

Id: prp_spo2_2
    pc <rdf:first> p
    pc <rdf:rest> t                  [Constraint t != <rdf:nil>]
    start p next
    next t last                      [Context <onto:_checkChain>]
    ----------------------------
    start pc last                    [Context <onto:_checkChain>]

Id: prp_spo2_3
    pc <rdf:first> p
    pc <rdf:rest> <rdf:nil>
    start p last
    ----------------------------
    start pc last                    [Context <onto:_checkChain>]
```

OWLIM – Rule language extensions for context

- OWLIM prp-spo2 rules example
  - Input data:

```
uncle owl:propertyChainAxiom x1
x1 rdf:first parent
x1 rdf:rest x2
x2 rdf:first brother
x2 rdf:rest rdf:nil

Lola parent Birgit
Birgit brother Klaus
```

  - Leads to inferences:

```
prp_spo2_3 =>
  Birgit x2 Klaus _checkChain

prp_spo2_2 =>
  Lola x1 Klaus _checkChain

prp_spo2_1 =>
  Lola uncle Klaus
```

OWLIM – OWL2 RL Support

- OWLIM was modified to use context in rules
- OWL2 RL fully supported, except some data-type rules missing:
  - dt-type2
  - dt-eq
  - dt-diff
  - dt-not-type
- Performance
  - Small loading degradation for data-sets that don't use OWL2-RL features, e.g. property chains

OWL2 QL

- OWL2 profile[1]
  - Syntactic subset of OWL2
  - Designed for querying assertions via an ontology through query-rewriting (LOGSPACE wrt. number of assertions)
  - Effectively backward-chaining

- Doesn't look suitable for OWLIM
  - Problem 1: OWLIM uses (mostly) forward-chaining reasoning
  - Problem 2: OWL2 QL permits existential quantification

[1] http://www.w3.org/TR/owl2-profiles/#OWL_2_QL

OWL2 QL – Existential Quantification

- Consider this example:

```
Prefix ( : = <http://example.org/> )
Ontology (
SubClassOf (:GrandPa
          ObjectSomeValuesFrom (:fatherOf owl:Thing))
ClassAssertion ( :GrandPa :Tom ) )
```

OWL2 QL Existential Quantification

- Which is supported in OWLIM with this rule:

```
Id: exst1
y <owl:onProperty> p
y <owl:someValuesFrom> <owl:Thing>
a <rdfs:subClassOf> y
x <rdf:type> a [Constraint x != blank]
---------------------------
x p b
```

- Exploits OWLIM's behaviour that unbound head variables make new blank nodes

OWL2 QL Existential Quantification

- However, this does not work in all cases
  - And it's dangerous in some

- Consider:

  SubclassOf(Person ObjectSomeValuesFrom(hasParent owl:Thing))
  SubclassOf(ObjectSomeValuesFrom(hasChild owl:Thing) Person)
  InverseOf(hasChild hasParent)
  Person(tom)

- Rule exst1 infers:

  tom hasParent b1

- and then

  b1 hasChild tom, b1 type Person, and if b1 is a Person there must exist…

OWL2 QL – other non-conformance

- If two classes are declared disjoint
  - Should infer that all pairs of members of each class are different from each other – Cartesian product
  - Instead, implemented with a consistency check – fires if an individual is a member of both classes
- For every class C there is a class that is the union of {C}
  - Forward-chaining -> infinite recursive class definition
- An individual related via disjoint properties to {a, b, c}
  - Should infer a set of mutually exclusive individuals
  - Instead, differentFrom pairs are inferred

OWL2 QL – conclusion

- No modifications required to OWLIM
  - Apart from changes already made for OWL2 RL
- However, not complete
  - Existential problems
  - Some inferences too expensive or not possible
  - Not complete
- But still passes most of the positive entailment tests