Reasoning with the Web Ontology Language (OWL)

JESSE WEAVER, PH.D.

Fundamental & Computational Sciences Directorate, Senior Research Computer Scientist
Discovery 2020 Short Course on Semantic Data Analysis
IR# PNNL-SA-95530
Outline

- **Brief** Introduction to RDFS and OWL
  - Alphabet soup: RDFS, OWL, DL, EL, QL, RL
  - Two semantics, five variants of OWL

- Learning by Example
  - Playing with Pellet, an OWL reasoner
  - Go through the OWL 2 primer examples
    - Learn about the Open World Assumption (OWA)
    - Learn about the Non-Unique Name Assumption (NUNA)

- A Naïve Model of IP Flow Data in OWL
  - IPFIX in OWL
  - IP Flow Disambiguation
<table>
<thead>
<tr>
<th>Name</th>
<th>Native Profiles</th>
<th>Semantics</th>
<th>(Non-) Conformance</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CB</td>
<td>EL</td>
<td>direct</td>
<td>Lacks support for datatypes/values (coming soon).</td>
<td>A resolution based reasoner for OWL EL. Superseded in 2011 by ELK (see below).</td>
</tr>
<tr>
<td>CEL</td>
<td>EL</td>
<td>direct</td>
<td>Lacks support for nominals (ObjectHasValue and ObjectOneOf) and datatypes/values.</td>
<td>CEL is an open-source polynomial-time Classifier for the OWL 2 EL profile. It has demonstrated scalability and proved well suited for several biomedical ontologies.</td>
</tr>
<tr>
<td>ELK Reasoner</td>
<td>EL</td>
<td>direct</td>
<td>As of Dec 2012, no support for keys, datatype reasoning and limited support for nominals/values; see website for details.</td>
<td>ELK is a Java-based, open source reasoning for OWL EL that has demonstrated excellent performance on various life science ontologies. See the documentation for details.</td>
</tr>
<tr>
<td>ELLY</td>
<td>EL, RL</td>
<td>direct</td>
<td>OWL profile support under development.</td>
<td>Data-centric implementation of reasoning and query answering based on translating EL/RL to datalog in a way that preserves assertional entailments.</td>
</tr>
<tr>
<td>FaCT++</td>
<td>DL</td>
<td>direct</td>
<td>Fully conformant except for keys and some datatypes (coming soon).</td>
<td>FaCT++ is an open-source tableau-based OWL 2 DL reasoner. It is implemented in C++ and shows exceptional performance on expressive ontologies.</td>
</tr>
<tr>
<td>Hermit</td>
<td>DL</td>
<td>direct</td>
<td>Fully conformant</td>
<td>Based on a novel &quot;hypertableau&quot; algorithm, Hermit can determine whether or not the input ontology is consistent, identify subsumption relationships between classes, and much more.</td>
</tr>
<tr>
<td>Jena</td>
<td>RL</td>
<td>RDF-based/direct</td>
<td>Complete RL/PR1 support is in the plan</td>
<td>An experimental OWL 2 RL implementation, based on translating the premise ontology to a set of Jena inference rules, is under development by HP Labs Bristol and Aberdeen University.</td>
</tr>
<tr>
<td>Owlgros</td>
<td>QL</td>
<td>direct</td>
<td>Fully conformant</td>
<td>An open source reasoner for OWL2 QL that is designed to use the PostgreSQL RDBMS.</td>
</tr>
</tbody>
</table>
OWL has a fixed set of features, but they can be interpreted according to two different semantics.

The **RDF-based** semantics are an extension of RDF(S).

Any valid RDF graph is syntactically valid.

Undecidable reasoning in the general case.

You can do anything that RDF(S) allows... at your own risk.

The **direct** (DL-based) semantics are independent of RDF.

Some valid RDF graphs are syntactically invalid.

Decidable reasoning.
OWL Fragments

- Full – all features under the RDF-based semantics.
  - RL – a subset for rule-based reasoning.
  - DL – all features under the direct semantics.
    - EL – a subset for large schema.
    - QL – a subset for query rewriting.
OWL Worldview

- **Classes**
  - aka concepts

- **Individuals**
  - aka instances, objects

- **Object Properties**
  - aka roles; relationships between individuals

- **Object Property Assertions**
  - a single statement using an object property

- **Datatypes**
  - like primitives (e.g. integers)

- **Literals**
  - instances of datatypes

- **Datatype Properties**
  - relationships from individuals to literals

- **Datatype Property Assertions**
  - a single statement using a datatype property
Outline

- Brief Introduction to RDFS and OWL
  - Alphabet soup: RDFS, OWL, DL, EL, QL, RL
  - Two semantics, five variants of OWL

- Learning by Example
  - Playing with Pellet, an OWL reasoner
  - Go through the OWL 2 primer examples
    - Learn about the Open World Assumption (OWA)
    - Learn about the Non-Unique Name Assumption (NUNA)

- A Naïve Model of IP Flow Data in OWL
  - IPFIX in OWL
  - IP Flow Disambiguation
Pellet is an OWL DL reasoner.

What can it do for you?
Pellet is an OWL DL reasoner.

What can it do for you?
- Ontology classification.
Pellet

- Pellet is an OWL DL reasoner.
- What can it do for you?
  - Consistency checking with explanation.
Pellet is an OWL DL reasoner.

What can it do for you?
- Basic SPARQL querying with DL reasoning.
Pellet is an OWL DL reasoner.

What can it do for you?
- Entailment checking (do triples A imply triples B?).
Pellet

- Pellet is an OWL DL reasoner.
- What can it do for you?
  - Materialize inferences.
Outline

- Brief Introduction to RDFS and OWL
  - Alphabet soup: RDFS, OWL, DL, EL, QL, RL
  - Two semantics, five variants of OWL

- Learning by Example
  - Playing with Pellet, an OWL reasoner
  - Go through the OWL 2 primer examples
    - Learn about the Open World Assumption (OWA)
    - Learn about the Non-Unique Name Assumption (NUNA)

- A Naïve Model of IP Flow Data in OWL
  - IPFIX in OWL
  - IP Flow Disambiguation
# 4-1-Cases-and-Instances/data.ttl

```
:Person rdf:type owl:Class .
:Woman rdf:type owl:Class .
:Mary rdf:type :Woman .
```

- **Query 1: Who are women?**
  - Answer: Mary

- **Query 2: Who are persons?**
  - Answer: (none known)
Class Hierarchies

Query 1: Who are persons?
- Answer: Mary

Query 2: Who are humans?
- Answer: Mary

Query 3: Human is equivalent to Person, but is Person equivalent to Human?
- Answer: Yes.
Query 1: Who is not a Man?

Answer: Mary

Because of OWA, we don’t know whether Ashley is not a man.

What if we add ...

:Ashley rdf:type :Man, :Woman .
Object Properties

```ttl
# 4-4-Object-Properties/data.ttl
:hasWife rdf:type owl:ObjectProperty .
:John :hasWife :Mary .
```

► What if we add …

```ttl
[] rdf:type owl:NegativePropertyAssertion ;
owl:sourceIndividual :Mary ;
owl:assertionProperty :hasWife ;
owl:targetIndividual :John .
```

► What if we add …

```ttl
[] rdf:type owl:NegativePropertyAssertion ;
owl:sourceIndividual :John ;
owl:assertionProperty :hasWife ;
owl:targetIndividual :Mary .
```

INCONSISTENT
Property Hierarchies

```
# 4-5-Property-Hierarchies/data.ttl
:hasWife rdfs:subPropertyOf :hasSpouse .
:John :hasWife :Mary .
```

Query 1: Who is John’s spouse?
- Answer: Mary

Query 2: Who is Mary’s spouse?
- Answer: (none known)
- We have yet to codify the fact that the spousal relationship is symmetric.
  (Soon to come)
Domain and Range Restrictions

Query 1: Who are men?
- Answer: John

Query 2: Who are women?
- Answer: Mary

Does this mean John is not a woman?
- Under the OWA, not necessarily (since we have not modeled that here).
Equality and Inequality of Individuals

Query 1: Is James the same as Jimmy?

Answer: Yes

What if we add...

:James owl:differentFrom :Jimmy.

Is James still the same as Jimmy?

Answer: Yes and No

Is John the same as James?

Answer: Maybe (unknown), but won’t show up in query answer.

Due to NUNA, John and James could possibly be the same.
# 4-8-Datatypes/data.ttl

```turtle
# 4-8-Datatypes/data.ttl
:hasAge rdf:type owl:DatatypeProperty ;
   rdfs:range xsd:nonNegativeInteger .
:John :hasAge "51"^^xsd:nonNegativeInteger .
```

What if we add...

```turtle
[] rdf:type owl:NegativePropertyAssertion ;
   owl:sourceIndividual :John ;
   owl:assertionProperty :hasAge ;
   owl:targetValue "51"^^xsd:nonNegativeInteger .
```

INCONSISTENT
Complex Classes: Intersection

Query 1: Who are mothers?
- Answers: Jill, Mary

Query 2: Who are women?
- Answers: Jane, Jill, Mary
Complex Classes: Union

# 5-1-Complex-Classes/data-2.ttl

```
:Parent owl:equivalentClass _:x .
_:x owl:unionOf ( :Mother :Father ) .
:John rdf:type :Father .
:Mary rdf:type :Mother .
:Ashley rdf:type :Parent .
```

- **Query 3**: Who are parents?
  - Answers: John, Mary, Ashley

- **Query 1**: Who are mothers?
  - Answers: Mary

- **Query 4**: Who are fathers?
  - Answers: John

- **Who are mothers or fathers?**
  - Answers: John, Mary, Ashley
Complex Classes: Complement

# 5-1-Complex-Classes/data-3.ttl

:ChildlessPerson owl:equivalentClass _:x .
_:x owl:intersectionOf ( :Person _:y ) .
_:y owl:complementOf :Parent .
:Ashley rdf:type :Parent .
:Rick rdf:type :ChildlessPerson .
:Joe rdf:type :Person ,

Query 5: Who is childless?

Answers: Joe, Rick
# 5-2-Property-Restrictions/data-1.ttl

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>:_x rdf:type owl:Restriction ; owl:onProperty :hasChild ; owl:someValuesFrom :Person .</code></td>
<td></td>
</tr>
<tr>
<td><code>:_x owl:equivalentClass _:x .</code></td>
<td></td>
</tr>
<tr>
<td><code>:_:x rdf:type owl:Restriction ; owl:onProperty hasChild ; owl:someValuesFrom :Person .</code></td>
<td></td>
</tr>
<tr>
<td><code>:_:Mary :hasChild :Becky .</code></td>
<td></td>
</tr>
<tr>
<td><code>:_:Jill :hasChild :John .</code></td>
<td></td>
</tr>
<tr>
<td><code>:_:John rdf:type :Person .</code></td>
<td></td>
</tr>
<tr>
<td><code>:_:Rick rdf:type :Parent .</code></td>
<td></td>
</tr>
</tbody>
</table>

**Query 1: Who are parents?**
- Answers: Jill, Rick

**Mary has a child. Why isn’t she a parent?**
- Because her child Becky is not known to be a person.

**Who is Rick’s child?**
- Answer: `_:newBlankNode (not sure who, but there is one)`
Query 3: Who are happy persons?

- Answers: John, Mary, Sue

Why John?

- Because his parent Mary is (explicitly) happy, which can only be true if John (and his siblings) are happy.

Jill’s child Sue is happy. Why isn’t Jill happy?

- Because of OWA, Jill might have other children who are not happy.
Property Restrictions: Value

Query 4: Who are of type :JohnsChildren?
- Answers: Joe, Jim

Query 5: Who :hasParent :John?
- Answers: Joe, Jim
Property Characteristics: Inverse and Symmetric

Query 1: Who are Sam’s parents?
- Answers: Mary

Query 2: Who is Rick’s spouse?
- Answers: Mary
Property Characteristics: Asymmetric and Irreflexive

What if we add ...

What if we add ...

What if we add ...

# 6-1-Property-Characteristics/data-3.ttl
:hasChild rdf:type owl:AsymmetricProperty .
:Amy :hasChild :Sally .

# 6-1-Property-Characteristics/data-6.ttl
:parentOf rdf:type owl:IrreflexiveProperty .

# 6-1-Property-Characteristics/data-8.ttl
Property Characteristics: Functional and Inverse Functional

How many husbands does Mary have?
- Answer: One
- There are two results – Jim and James – but it is inferred that :Jim owl:sameAs :James.

How many people have James as a husband?
- Answer: One
- There are two results – Mary and Rosemary – but it is inferred that :Mary owl:sameAs :Rosemary.
Who are the ancestors of Mary?

- Answers: Jill, Joan
Query 1: Who are the grandparents of whom?

Answers: Joan is a grandparent of Mary.

[:Mary :hasGrandparent :Joan]

Very powerful… like edge traversal.
Jim and James identify the same person because the values of the key properties are the same.

With only one property :hasSSN, this is equivalent to saying that [:hasSSN rdf:type owl:InverseFunctionalProperty].
Outline

- Brief Introduction to RDFS and OWL
  - Alphabet soup: RDFS, OWL, DL, EL, QL, RL
  - Two semantics, five variants of OWL

- Learning by Example
  - Playing with Pellet, an OWL reasoner
  - Go through the OWL 2 primer examples
    - Learn about the Open World Assumption (OWA)
    - Learn about the Non-Unique Name Assumption (NUNA)

- A Naïve Model of IP Flow Data in OWL
  - IPFIX in OWL
  - IP Flow Disambiguation
IP Flow Information eXport (RFC5101)

Based on Cisco Systems NetFlow Services Export Version 9

*Metering Processes* measure IP traffic and aggregate IP flow data (records), which are then sent to *Collecting Processes*.

Flow records contain some standard information called “Information Elements” about flows. For example:

- Source and destination IP address.
- Source and destination ports.
- Transport protocol.

How can we model this as an ontology to help us disambiguate flows?

- That is, Collecting Processes may receive flow records about the same Flow from different Metering Processes.
- How can we determine which records identify the same flow?
Basic Idea for Flow Disambiguation

```rml
:Flow rdf:type owl:Class ;
  owl:hasKey ( :sourceIPAddress 
  :destinationIPAddress 
  :transportProtocol 
  :sourcePort 
  :destinationPort ) .
```
With the rest of the ontology, this implies [:Flow1 owl:sameAs :Flow2].
Try it!

May 20, 2013

```
pellet query -q query-same-flows.rq -v ipfix.ttl
```

Query file: query-same-flows.rq
Start parsing query file
Finished parsing query file in 00:00:00.183
Query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ipf: <http://tools.ietf.org/html/rfc5102#>
PREFIX : <http://example.org/>

SELECT ?flow1 ?flow2 {
    ?flow1 owl:sameAs ?flow2 .
    FILTER(!sameTerm(?flow1, ?flow2))
}
```

There are 1 input files:
ipfix.ttl
Start loading
Finished loading in 00:00:00.086
Input size: Classes = 4, Properties = 25, Individuals = 9
Expressivity: EL+
Start consistency check
Finished consistency check in 00:01:06.573
Created query engine: com.hp.hpl.jena.sparql.engine.QueryExecutionBase
Start query execution
Finished query execution in 00:00:00.094
Query Results (2 answers):
flow1 | flow2
-------
Flow2 | Flow1
Flow1 | Flow2

Timer summary:
```
<table>
<thead>
<tr>
<th>Name</th>
<th>Total (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>67290</td>
</tr>
<tr>
<td>parsing query file</td>
<td>103</td>
</tr>
<tr>
<td>loading</td>
<td>86</td>
</tr>
<tr>
<td>consistency check</td>
<td>56573</td>
</tr>
<tr>
<td>query execution</td>
<td>94</td>
</tr>
</tbody>
</table>
```

```
Thank you!

Questions?