OWL
The Web Ontology Language
part II: beyond the basics

Pavel Klinov
Quickly about (**decidable**) reasoning

OWL and RDF

- OWL DL and OWL Full
- global restrictions

Modeling trickiness

- N-ary predicates
- meta-modeling
- OWA, UNA, and integrity constraints
- pain points: **time** and **uncertainty**
Why reasoning?

Important for:

- **quality assurance during ontology design**
  - detects *false* entailments and non-entailments
  - esp. in case of *multiple* authors

- **semantic integrations**
  - errors during ontology re-use (remember imports)?
  - errors during ontology *mapping* and *alignment*

- **deployment**
  - any schema violations in my *data*?
  - is my data *under*-described?
Reasoning

Typical reasoning problems given an ontology O:

- is O consistent?
- does O entail an axiom?
- classification: all class inclusions for named classes
- query answering
  - DL query (querying with arbitrary class expressions)
    \[
    \text{ObjectIntersectionOf}(:\text{Person} 
    \text{ObjectSomeValue}(:\text{hasParent} :\text{Peter}))
    \]
  - conjunctive queries (tomorrow)

Reduces to consistency
(as you know from DLs)
On decidability

OWL is based on 20+ years of DL research

- largely about finding practical decision procedures
- decidability means restrictions on the language
- do we care?

Well, sort of yes

- optimizations are typically easier to develop
- semi-decidability insufficient!
  - O entails $\alpha$ iff $O \cup \{\alpha\}$ is inconsistent
  - if consistency is semi-decidable, entailment is not
What does “OWL is based on RDF” mean?

- each OWL axiom maps to a set of RDF triples
- which require extra vocabulary (owl: namespace)
OWL and RDF

What does “OWL is based on RDF” mean?

- each OWL axiom maps to a set of RDF triples
- which require extra vocabulary (owl: namespace)
The OWL to RDF Mapping

Entities are mapped to RDF resources
Data values become literals
Expressions and axioms are mapped to sets of triples
The OWL to RDF Mapping

Entities are mapped to RDF resources
Data values become literals
Expressions and axioms are mapped to sets of triples
Class expression example:
  - OWL: ObjectAllValuesFrom( :hasRelative :Griffins )
  - RDF:
    _:x rdf:type owl:Restriction
    _:x owl:onProperty :hasRelative
    _:x owl:allValuesFrom :Griffins
What about semantics?

RDF(S) has its own model-theoretic semantics
What about semantics?

RDF(S) has its own model-theoretic semantics

RDF graph

Interpretation
What about semantics?

RDF(S) has its **own** model-theoretic semantics

RDF graph

```
:Stewie :hasParent :Lois rdf:type :Person
```

Interpretation
What about semantics?

RDF(S) has its own model-theoretic semantics.
What about semantics?

RDF(S) has its own model-theoretic semantics

RDF graph

Interpretation
Extra semantic conditions

For RDFS resources:

\[(c_1, c_2) \in \text{IEXT}(\text{IS(rdfs:subClassOf)}), \text{ then} \]
\[c_1, c_2 \text{ are classes, } \text{ICEXT}(c_1) \subseteq \text{ICEXT}(c_2)\]
Extra semantic conditions

For **RDFS** resources:

\[(c_1, c_2) \in \text{IEXT}(\text{IS}(\text{rdfs}:\text{subClassOf})), \text{ then} \]
\[c_1, c_2 \text{ are classes, ICEXT}(c_1) \subseteq \text{ICEXT}(c_2)\]

Similar for all **OWL** resources

\[(z, c) \in \text{IEXT}(\text{IS}(\text{owl}:\text{someValuesFrom})), \text{ and} \]
\[(z, p) \in \text{IEXT}(\text{IS}(\text{owl}:\text{onProperty})), \text{ then} \]
\[\text{ICEXT}(z) = \{x \mid \exists y : (x, y) \in \text{IEXT}(p) \text{ and } y \in \text{ICEXT}(c)\} \]

essentially encodes “\(z \sqsubseteq \exists p. c\)”
Semantic correspondence

Ontology O can be interpreted in two ways:

- directly, via the DL model theory
- indirectly, as an RDF graph via the RDF model theory

Natural question: are the semantics equivalent?
Semantic correspondence

Ontology O can be interpreted in two ways:

- directly, via the DL model theory
- indirectly, as an RDF graph via the RDF model theory

Natural question: are the semantics equivalent?

- by means of entailment
- well, mostly yes: the OWL 2 correspondence theorem

let G1 and G2 be RDF graphs s.t.

F(G1) and F(G2) are corresponding ontologies in FS*

F(G1) entails F(G2) under the DL semantics, then

G1 entails G2 under the RDF semantics

* which meet the OWL 2 DL global restrictions
So every OWL ontology maps to an RDF graph.

What about the other way?

- is every RDF graph an OWL ontology?
  w.r.t. some canonical parsing?
OWL 2 DL and OWL 2 Full

So **every** OWL ontology maps to an RDF graph.

What about the **other** way?

- is every RDF graph an OWL ontology?
  w.r.t. some canonical parsing?

- **not** in the DL sense
  - can make statements about the standard vocabulary
    \(<\text{rdf:}\text{type} \text{rdf:}\text{type} \text{rdf:}\text{type}>\) is a **valid** RDF triple!
  - too expressive (not in a **decidable** fragment of FOL)
OWL 2 DL and OWL 2 Full

So every OWL ontology maps to an RDF graph.

What about the other way?

- is every RDF graph an OWL ontology?
  - w.r.t. some canonical parsing?

- not in the DL sense
  - can make statements about the standard vocabulary
    <rdf:type rdf:type rdf:type> is a valid RDF triple!
  - too expressive (not in a decidable fragment of FOL)

The decidable fragment is called OWL 2 DL.

What's beyond is OWL 2 Full - the dark side, an undecidable, very expressive ontology language.
OWL 2 DL syntactic restrictions

Can't use terms from `owl:`, `rdf:` etc. as entities

`SubObjectPropertyOf (rdf:type :typeOf)`
OWL 2 DL syntactic restrictions

Can't use terms from `owl:`, `rdf:` etc. as entities

```
SubObjectPropertyOf (rdf:type :typeOf)
```

Restrictions on datatypes

- no datatype occurs on LHS of two or more definitions
- datatype definitions are acyclic
  (all value spaces are exact, datatypes are unfoldable)

```
DatatypeDefinition( :TaxNumber
    DatatypeRestriction( xsd:string xsd:pattern "[0-9]{11}"")
DatatypeDefinition( :AlternativeTaxNumber
    DatatypeRestriction( xsd:string xsd:pattern "[0-9]{3}-[0-9]{7}"")
DatatypeDefinition( :ID
    DataUnionOf( :TaxNumber :AlternativeTaxNumber ) )
```
Complex object properties

Property is complex if its assertions can be derived from other property assertions

- this includes owl:topObjectProperty
- properties on the RHS of a chain

Otherwise it is simple

- Examples
  
  SubObjectPropertyOf(ObjectPropertyChain(
    :hasParent :hasBrother) :hasUncle)

  SubObjectPropertyOf(:hasUncle :hasRelative)
Restrictions on complex properties

**Complex** properties can't occur in cardinality restrictions

- ObjectMinCardinality
- ObjectMaxCardinality
- ObjectExactCardinality
- ObjectHasSelf

What we *can't* say:

- ObjectMinCardinality(2 :hasRelative owl:Thing)
- TransitiveObjectProperty(:loves)
- ObjectHasSelf(:loves)
Restrictions on property hierarchies

Object property hierarchies must be regular

- let \( \rightarrow^* \) be the reflexive-transitive closure on properties
- must exist strict linear order \(<\) on properties
  s.t. \( :p_1 < :p_2 \) means \( :p_2 \rightarrow^* :p_1 \) doesn't hold
Restrictions on property hierarchies

Object property hierarchies must be **regular**

- Let $\rightarrow$ * be the reflexive-transitive closure on properties
- Must exist **strict linear order** $<$ on properties
  
  s.t. $:p_1 < :p_2$ means $:p_2 \rightarrow * :p_1$ doesn't hold
- Each $\text{SubObjectPropertyOf} (\text{ObjectPropertyChain}( :p_1 \ldots :p_n ) :p)$ axiom meets the following
  - $\text{SubObjectPropertyOf} (\text{ObjectPropertyChain}( :p :p ) :p )$, or
  - $:p$ is owl:topObjectProperty, or
  - $:p_i < :p$ for all $i = 1 \ldots n$, or
  - $\text{SubObjectPropertyOf} (\text{ObjectPropertyChain}( :p :p_1 \ldots :p_n ) :p )$, or
  - $\text{SubObjectPropertyOf} (\text{ObjectPropertyChain}( :p_1 \ldots :p_n :p ) :p)$
Regular and irregular hierarchies

SubObjectPropertyOf (  
    ObjectPropertyChain( :hasFather :hasBrother ) :hasUncle )
SubObjectPropertyOf (  
    ObjectPropertyChain( :hasUncle :hasWife ) :hasAuntInLaw )
:hasFather < :hasBrother < :hasUncle < :hasWife < :hasAuntInLaw
Regular and irregular hierarchies

SubObjectPropertyOf ( ObjectPropertyChain

  (:hasParent :hasSpouse :hasParent ) :hasGrandparent )

:hasParent < :hasSpouse < :hasGrandparent
Regular and irregular hierarchies

SubObjectPropertyOf ( ObjectPropertyChain

   (:hasFather :hasBrother ) :hasUncle )

SubObjectPropertyOf ( ObjectPropertyChain

   ( :hasChild :hasUncle ) :hasBrother )

no linear order between :hasUncle and :hasBrother

SubObjectPropertyOf ( ObjectPropertyChain (:p :s :r) :s)

(:s, :s) can't be in <
How're you feeling?

The OWL 2 DL vs. OWL 2 Full is tricky
- the OWL API will check the profile!
- and point to where you violate it

The rest is easier 😊
- modeling issues
- where the Full stuff matters
OWL can't represent everything

It can't represent what FOL can't (naturally) represent

- temporal knowledge
- various sorts of uncertainty
- higher-order knowledge

It has troubles with knowledge beyond the 2-var FOL

- n-ary relationships of sorts
N-ary stuff is problematic

OWL (and RDF) are 2-variable logics

- schema restrictions and properties are binary
  
  ObjectExactCardinalityFrom(2 :hasParent :Person)
  ObjectAllValuesFrom(:hasChild :Person)

- assertions are binary

How do we say:

- Stewie has a high (but falling) temperature?
- Megan bought a book A from store B?
- Lois visited LAX, JFK, and BOS on a single trip?
Workarounds

Via classes that work like *reified* properties

In OWL 2 property chains help

\[ :\text{has-temperature} \circ :\text{has-trend} \subseteq :\text{has-temperature-trend} \]
Meta-modeling

Bird
 subclasses of
Eagle

BaldEagle
 instance of
Harry
Meta-modeling

:BaldEagle subclass of :Endangered would imply :Harry is a :Species

:Species and :Endangered are meta-classes
Meta-modeling in DL and Full

OWL 2 Full

- supports meta-modeling!
  
  ClassAssertion(:BaldEagle :EndangeredSpecies)

OWL 2 DL

- limited support of meta-modeling

In contrast to DL, OWL 2 Full:

i. can use the built-in vocabulary

ii. don't separate out classes, properties, and individuals

iii. has no decidability restrictions
Can it work in OWL 2 DL?

OWL Full is **trivially** undecidable due to iii. which isn't **very** useful for meta-modeling

Is OWL 2 DL with i. and ii. decidable?
Can it work in OWL 2 DL?

OWL Full is *trivially* undecidable due to iii. which isn't *very* useful for meta-modeling

Is OWL 2 DL with i. and ii. decidable?

- **bad** news: no
- **good** news: it's due to i. while we really want ii.
  who wants `ClassAssertion(owl:allValuesFrom :X)`?!

Main question: how to allow **ii.** and still be first-order?

- semantic extensions (B. Motik, 2007)
- axiomatization (S. Rudolph and B. Glimm, 2010)
Semantic extensions to OWL 2 DL

Contextual semantics (or punning with entities)

- each name :n treated as :n_class, :n_ind, :n_obj, :n_data depending on syntactic occurrence
- **simple**: ClassAssertion(:BaldEagle :EndangeredSpecies)
- no interaction between :BaldEagle_{class}, :BaldEagle_{ind}

→ no **useful** entailments

How about something more in the OWL 2 Full spirit?
Semantic extensions to OWL 2 DL

HiLog semantics

- $\Delta$ - the domain
- $I$ - maps all entities to elements of $\Delta$
- $C^I: \Delta \rightarrow 2^\Delta$ atomic class extension
- $R^I: \Delta \rightarrow 2^{\Delta \times \Delta}$

Each entity has its identity, a dedicated domain element which is then extended to interpret the entity.

Can be encoded in FOL.
Contextual vs. HiLog

contextual model

syntax

HiLog model

:BaldEagle
Direct axiomatization in OWL 2 DL

Extend the vocabulary

- classes: :Inst, :Class
- properties: :type, :subClassOf, :Rinst
- individuals: :oC for each normal class C
Direct axiomatization in OWL 2 DL

Extend the vocabulary

- classes: :Inst, :Class
- properties: :type, :subClassOf, :Rinst
- individuals: :oc for each normal class C

And restrict it

- DisjointClasses(:Inst :Class)
- ClassAssertion(oc :C), ClassAssertion(:i :Inst) for each :i, :C
- EquivalentClasses(:C ObjectSomeValuesFrom(:type :C))
- ObjectPropertyDomain(:R :Inst) for each :R
  ObjectPropertyDomain(:R :Inst)
- etc.-etc.
Direct encoding in OWL 2 DL

So we

- conceptualize the meta-layer
- make sure it doesn't interfere with ontology layer
- no unwanted entailments due to the extra stuff
  (could be hidden behind a reasonable API/GUI)
Direct encoding in OWL 2 DL

So we

- conceptualize the **meta-layer**
- make sure it doesn't interfere with **ontology** layer
- no **unwanted** entailments due to the extra stuff
  (could be hidden behind a reasonable API/GUI)
Meta-reasoning

Endangered $\rightarrow$ cannot be hunted, don't hunt Harry!
Meta-reasoning

Endangered → cannot be hunted, don't hunt Harry!
Contextual semantics: can't do within logic
Meta-reasoning

Endangered $\rightarrow$ cannot be hunted, don't hunt Harry!

Contextual semantics: can't do within logic

HiLog semantics

- need language extensions:
  Endangered(P) $\land$ P(i) $\rightarrow$ CantHunt(i)
- entails cantHunt(:Harry)
Meta-reasoning

Endangered → cannot be hunted, don't hunt Harry!

Contextual semantics: can't do logically

HiLog semantics

• need language extensions:

  Endangered(P) ∧ P(i) → CantHunt(i)

• entails cantHunt(:Harry)

Axiomatization

  ObjectPropertyAssertion(:subClassOf :oBaldEagle :oEndangered)

  SubClassOf(ObjectSomeValue(:type :oEndangered) :CantHunt)

  entails ClassAssertion(:Harry :CantHunt)
Meta-modeling in OWL 2 DL

Some **limited** support is available:

- annotations (:isEndangered could be semantic-free)
- punning (BaldEagle-as-class vs. BaldEagle-as-instance)
  but not for properties

What's often done:

- parallel hierarchy of meta-classes and **extra-logical** linking
- OWL Full
Classes as property values

Another example of meta-modeling

:Animal

subclass of

:Lion

subclass of

:AfricanLion
Classes as property values

Another example of meta-modeling

The books are not about some specific lions but about (African)Lion as a class.
Workaround: parallel hierarchy

Obvious maintenance overhead for keeping the hierarchies in sync

Or (you guessed it!) OWL Full
Integrity constraints

Popular idea: OWL as a constraint language for RDF

• take a Linked Data dataset
• describe ICs as OWL axioms
• validate!
Integrity constraints

Popular idea: OWL as a constraint language for RDF

- take a Linked Data dataset
- describe ICs as OWL axioms
- validate!

Does NOT WORK
IC failure example 1

Schema

SubClassOf(:Person
ObjectSomeValuesFrom(:hasParent :Person))

Data

ClassAssertion(:Stewie :Person)

Valid?
IC failure example 1

Schema

SubClassOf(:Person
  ObjectSomeValuesFrom(:hasParent :Person))

Data

ClassAssertion(:Stewie :Person)

Valid?

• yes!
• but Stewie is inferred to have a parent
IC failure example 1

Schema

- SubClassOf(:Person
  ObjectSomeValuesFrom(:hasParent :Person))

Data

- ClassAssertion(:Stewie :Person)

Problem

- **Open World Assumption**
  - Stewie is not *known* to have a parent
  - but he must, otherwise it's inconsistent
IC failure example 2

Schema

SubClassOf(:Person
ObjectMaxCardinality(1 :hasMother :Woman)

Data

ClassAssertion(:Stewie :Person)
ObjectPropertyAssertion(:hasMother :Stewie :Lois)
ObjectPropertyAssertion(:hasMother :Stewie :Peter)

Valid?
IC failure example 2

Schema

SubClassOf(:Person
ObjectMaxCardinality(1 :hasMother :Woman)

Data

ClassAssertion(:Stewie :Person)
ObjectPropertyAssertion(:hasMother :Stewie :Lois)
ObjectPropertyAssertion(:hasMother :Stewie :Peter)

Valid?

• yes!
• but :Lois and :Peter are inferred to be identical
IC failure example 2

Schema

SubClassOf (:Person
ObjectMaxCardinality(1 :hasMother :Woman)

Data

ClassAssertion(:Stewie :Person)
ObjectPropertyAssertion(:hasMother :Stewie :Lois)
ObjectPropertyAssertion(:hasMother :Stewie :Peter)

Problem

Lack of the **Unique Name Assumption**
Lois and Peter aren't *known* to be different
OWL and ICs: proposals

Rules with **DL-queries** and **NAF**

- $\text{DL[Person(x)]} \land \text{DL[hasParent(x,y)]} \rightarrow P(x,y)$
- $\text{DL[Person(x)]} \land \text{NAF}[P(x,y)] \rightarrow \bot$

**Minimal model interpretation**

- constraints checked in all **minimal** models

  ```
  ClassAssertion(:Stewie :Person)
  ClassAssertion(:Stewie ObjectSomeValuesFrom(:hasParent :Person))
  ```

  ```still valid!```
Integrity constraints as queries

Instead of (non-recursive) rules we can use SPARQL, a query language for RDF, which can express NAF as OPTIONAL/FILTER/!BOUND (NOT EXISTS in SPARQL 1.1)
Integrity constraints as queries

Instead of (non-recursive) rules we can use SPARQL, a query language for RDF which can express NAF as OPTIONAL/FILTER//!BOUND (NOT EXISTS in SPARQL 1.1).

Check that every named person has a named parent

ASK WHERE { ?x rdf:type :Person .
OPTIONAL { ?x :hasParent ?y .
?y rdf:type :Person . }
FILTER(!BOUND(?y))}

“yes” means a violation
Integrity constraints as queries

Can be implemented by RDF databases

- keep axioms *separately* from data
- run queries as data *changes*

Syntax does *not* matter

- OWL axioms $\rightarrow$ queries (Stardog)
- SPIN, queries as RDF triples (AllegroGraph)
  spinrdf.org
Time

OWL doesn't support temporal concepts:

- class of people who were employed before the crisis
- everyone will be eventually dead
- A was true, will be true, will be true after B... etc.

Available out-of-the-box? XSD datatypes

- xsd:dateTime, xsd:dateTimeStamp
- Facts are expressible:

  DataPropertyAssertion ( :startTime 
  :MeetingA “2002-09-24-06:00”^^xsd:dateTime )
Time: “solutions”

OWL Time (formerly DAML Time)

- ontology on top of the existing logical model
  
  SubClassOf(:Process

  ObjectSomeValuesFrom(:hasDuration time:Interval)

- may help standardize temporal vocabulary
- very limited temporal reasoning
Time: “solutions”

OWL Time (formerly DAML Time)

- ontology on top of the existing logical model

\[
\text{SubClassOf(:Process}
\]
\[
\text{ObjectSomeValuesFrom(:hasDuration time:Interval)}
\]

- may help standardize temporal vocabulary
- very limited temporal reasoning

Various extensions based on temporal logics

Rule built-ins

\[
\text{Patient(?p) } \land \text{ hasTreatment(?p, ?t) } \land \text{ hasDrug(?t, DDI) } \land \\
\text{ temporal:hasValidTime(?t, ?tVT) } \land \text{ temporal:before(?tVT, “1999”) } \\
\rightarrow \text{ TrialEligible(?p)}
\]
Uncertainty

Similar to Time: **first-order** logical model provides **very** limited means to capture uncertainty:

- disjunction
- Open World Assumption
  
  information may be *legitimately* missing
- no Unique Name Assumption
  
  captures canonical name uncertainty
  
  New York and The Big Apple
different from name *ambiguity*!
  
    New York as a city vs. New York as a state
The sad state of affairs

At least 30 yrs of the “uncertainty in AI” research

- combinations of logic and probability
  - very-very-VERY hard (computationally and cognitively)
  - ClassAssertion(:Stewie :Infant 0.7)

- Bayesian and Markov models
  - computationally tractable
  - but propositional!
  - ... or, again, computationally impractical

- statistical black-box models (Breast Cancer Risk Calc)

No reusable modeling of uncertainty in SemWeb
To summarize

OWL 2 isn't a silver bullet

But

- it's helpful in certain, reasonably understood scenarios
  - terminology management
  - data integration
- it matures fast
  - tool support is getting better
  - people accumulate experience

So

- you may try it for your next project
- and tell us about your experience! e.g. at OWLED!
Questions!