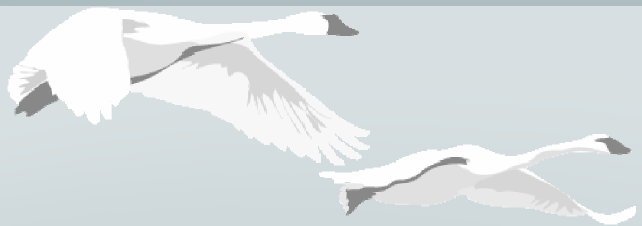


Semantic Web Ontologies

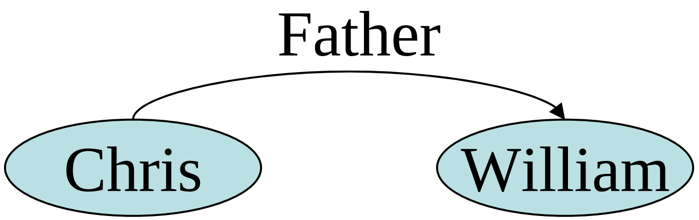
Chris Welty

IBM Research

(also W3C Semantic Web Best Practices WG)

RDF to RDFS

- RDF
  - triples as relations
    - <Chris Father William>
  - Interpreted as a graph
  - What do they mean?
    - <Person Father Person>
- RDFS
  - Data types: class
  - Specialization: subclass



The groundwork

- RDF + RDFS driven by multiple influences
  - Database
  - Knowledge Representation
  - Un/Semi Structured data
  - ERP
  - ...
- Provides a *foundation* to expand into these areas

OWL

- Adds *entailments* to RDFS
  - IF Author-of(*x*,*y*) AND Author-of(*z*,*y*)
    THEN Collaborator(*x*,*z*)
- Three levels
  - ~~Lite: easier to implement Description Logic~~
  - DL: based on latest Description Logic
  - Full: First order with reified predicates

Entailment

- $A \rightarrow B$ ("A implies B")
  - $\forall x\ P(x) \rightarrow Q(x) \wedge R(x)$
  - $P(A) \rightarrow Q(A) \wedge R(A)$
- $A \vDash B$ ("A entails B")
  - More general than implication
  - Implication generally requires unification
  - $(\forall x\ P(x) \vee Q(x) \rightarrow R(x)) \vDash (\forall x\ P(x) \rightarrow R(x))$
  - Computationally, a form of "caching"

The Two Semantics

- Language (denotational) semantics
  - Lambda calculus, model theory, set theory
  - Critical for understanding inference
  - What does it *mean* to say A $\rightarrow$ B?
- Domain (interpretive) semantics
  - What terms refer to
  - Constrain “unintended models”
  - What does it *mean* to say “Horse”

A Brief Skewed History of KR

- KR in the 70s
  - focused on graphical notations (i.e. semantic nets)
  - Much attention paid to taxonomies
  - No semantics
  - McDermott: “No representation w/o denotation”
- KR in the 80s
  - Split into several areas
  - Lots of semantics
  - Brachman and Levesque: “Tradeoff between expressiveness and tractability”

The Tradeoff

The tractability of computing the deductive closure of a knowledge base decreases as the expressiveness of the language increases

- Computational properties
  - Sound
    - Everything provable is true
  - Complete
    - Everything true is provable
  - Decidable
    - Everything true is provable in finite time

Description Logics Philosophy

- Map the territory around the E/T boundary
- Maintain computational properties
  - Sound, complete
  - Decidable in Pspace, often NP
  - A “decidable fragment” of FOL
- Start in known territory
- Increase expressiveness one operation at a time
- Set and Model Theoretic Semantics
- Focus on *subsumption* reasoning

Subsumption

- Exploit the power of taxonomies
- For *efficiency*, compute when one expression entails another

$\forall x,y \; \mathrm{Book}(x) \wedge \mathrm{Author\text{-}of}(x,y) \rightarrow \mathrm{Person}(y)$

$\forall x,y \; \mathrm{Book}(x) \wedge \mathrm{Person}(y) \wedge \mathrm{About}(x,y) \rightarrow \mathrm{Biography}(x)$

$\forall x,y \; \mathrm{Book}(x) \wedge \mathrm{About}(x,y) \wedge \mathrm{Author\text{-}of}(x,y) \rightarrow \mathrm{Autobiography}(x)$

$\models$

$\forall x \; \mathrm{Autobiography}(x) \rightarrow \mathrm{Biography}(x)$

Necessary and Sufficient

- C necessary for P

  $P(x) \rightarrow C(x)$

- C sufficient for P

  $C(x) \rightarrow P(x)$

- Description logics “concepts” (classes) include necessary *and* sufficient conditions
  - This is peculiar in KR

Simple example

- Book::

  (**all** author Person)
- Biography::

  (**and** Book (**all** about Person))
- Auto-Biography::

  (**and** Book (**same-as** about author))

$\models$ Auto-Biography $\subseteq$ Biography

Early DLs

- KL-ONE, NIKL
- Krypton
  - No longer *frame based*
- Structural DLs
  - CLASSIC (AT&T), BACK (Hamburg), Loom(ISI)
  - Eventually shown to be *incomplete*
- Later Propositional DLs
  - RACER, DLQ
  - FaCT

Simple Structural Language
(*FL-*)

- Concept Operators: **And**
  - **(and person male)**
- Relation operators: **all, exists**
  - (all author Person)
  - (exists author Person)
- Subsumption decidable in P
- “Compilation” reduces to linear

Experiment

- Add Negation to FL-
- Early result
  - semi-decidable in EXPTIME
- Later results
  - decidable in P-SPACE
  - Compilation yields average linear time complexity
  - Expensive cases documented

Simple Propositional Language (*ALC*)

- Concept Operators
  - AND, OR, NOT
- Relation Operators
  - ALL, Exists
- Decidable in P-SPACE
- Usually extremely fast

FaCT

- The most expressive DL to date
- Concept Operators
  - AND, OR, NOT
  - Concrete domains (integers, strings)
- Relation operators
  - AND, OR, NOT, inverse, exists, all, at-least, at-most
  - Same-as, one-of
  - Qualified number restrictions (at-most n C)
  - Transitivity
- Rules

OWL-DL

The good

- A tagged and updated version of FaCT
- Supported by tools
  - Prototype Reasoning engine (U Manchester)
  - Commercial Reasoning enginer (NI.com)
  - Ontology Editor
- Expressed in RDF, RDF-S
- Decidable in P-SPACE
- Average cases extremely fast

OWL-DL
The bad

- No Higher order, Modality
- No N-ary relations
- No Only-sufficient conditions
- No Non-monotonicity
- No general existential quantification

  $\exists x$ Elephant(x)

OWL-DL
The ugly

```
<owl:Class rdf:ID="Wine">
 <rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#madeFromGrape"/>
   <owl:minCardinality rdf:datatype="&xsd;NonNegativeInteger">
       1
   </owl:minCardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#locatedIn"/>
   <owl:minCardinality rdf:datatype="&xsd;NonNegativeInteger">
      1
   </owl:minCardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

What can you say?

- All Presidents are People

```
<owl:Class rdf:ID=“President">
  <rdfs:subClassOf rdf:resource=“Person"/>
</owl:Class>
```

- George Washington is a President

```
<President rdf:ID=“George Washington”/>
```

- A NonFrenchWine is not from France

```
<owl:Class rdf:ID="NonFrenchWine">
 <owl:intersectionOf rdf:parseType="Collection">
   <owl:Class rdf:about="#Wine"/>
   <owl:complementOf>
    <owl:Restriction>
     <owl:onProperty rdf:resource="#locatedIn" />
     <owl:hasValue rdf:resource="#FrenchRegion" />
    </owl:Restriction>
   </owl:complementOf>
  </owl:Class>
 </owl:intersectionOf>
</owl:Class>
```

← Located-in-France

NOT Located-in-France

- OWL Lite – easier to implement
  - Cardinality restricted to 0,1
  - All classes must be named
  - No One-of or disjoint-with
- OWL DL – decidable, complete
- OWL Full – undecidable, incomplete
  - Relations, properties, in UoD
  - Full RDF compatibility
  - Possible to implement other useful subsets (e.g. LP)

OWL is not just OWL-DL!!!

Generally, Lite $\subset$ DL $\subset$ Full

OWL Walkthrough

OWL Ontology Elements

- Classes

  <owl:Class rdf:ID=“**Winery**” />

- Individuals

  <Winery rdf:ID=“**WeltyVinyards**” />

- Properties

  <owl:ObjectProperty rdf:ID=“**hasColor**” />

OWL Individuals

- Populate the universe of discourse
  - Aka “instances”, “objects”
- Have structure
  - Relations to other individuals
  - Attributes with values
- Have class membership
  - Explicit and implicit
- Can be equated, differentiated

```
<Wine rdf:ID="ChateauChevalBlancStEmilion">
  <hasMaker rdf:resource="#ChateauChevalBlanc" />
  </Wine>
```

Could entail $\in$ FrenchWine

OWL Properties

- Relate individuals to other individuals
  - ObjectProperty
  - Aka relation
- Relate individuals to primitive values
  - DataTypeProperty
  - Aka attribute
- Can have global domain/range restrictions
- Can be related by special relations
  - SubProperty, Inverse, equivalence
- Special classes of properties
  - Transitive, symmetric, functional, inverseFunctional

OWL Full: properties and classes can be individuals

OWL Classes

- Describe *sets* of individuals
  - Constraints on membership, and on properties of members
- Can be related by special relations
  - Subclass, equivalence, union, intersection, complement, disjointness
- Can enumerate members
- Two special classes
  - OWL:Thing (subsumes all classes)
  - OWL:Nothing (subsumed by all classes)

OWL Full: classes can be individuals, and can have properties of their *own*

Local Property Restrictions

- Constrain the properties of class members
  - Cardinality, default value, range (universal, existential, qualified)
- Specified as part of a class

```
<owl:Class rdf:ID="Vintage">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#hasVintageYear"/>
   <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
     1
   </owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

All vintages must have 1 vintage year

Local vs. Global restrictions

```
<owl:ObjectProperty rdf:ID="madeFrom">
  <rdfs:domain rdf:resource="#ManufacturedThing" />
  <rdfs:range rdf:resource="#NaturalThing" />
</owl:ObjectProperty>

<owl:Class rdf:ID="Wine">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#madeFrom"/>
   <owl:allValuesFrom rdf:resource="#Grape" />
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

The madeFrom relation holds between instances of ManufacturedThing and NaturalThing (*sufficient*)

When the domain of a madeFrom relation is an instance of Wine, the range is an instance of Grape (*sufficient*)

Global Implications

```
<owl:Thing rdf:ID=“fruit1” />
<owl:Thing rdf:ID=“drink1“>
  <madeFrom rdf:resource= =“Fruit1”>
</owl:Thing>
```

drink1 madeFrom fruit1

$\models$

```
<owl:ObjectProperty rdf:ID="madeFrom">
  <rdfs:domain rdf:resource="#ManufacturedThing" />
  <rdfs:range rdf:resource="#NaturalThing" />
</owl:ObjectProperty>
```

```
<ManufacturedThing rdf:resource=“drink1” />
<NaturalThing rdf:resource=“fruit1” />
```

drink1 type ManufacturedThing

fruit1 type NaturalThing

Local Implications

```
<owl:Thing rdf:ID="fruit1" />
<Wine rdf:ID="drink1">
  <madeFrom rdf:resource="fruit1">
</owl:Thing>
```

drink1 type Wine

drink1 madeFrom fruit1

$\models$

```
<owl:Class rdf:ID="Wine"> <rdfs:subClassOf>  <owl:Restriction>
   <owl:onProperty rdf:resource="#madeFrom"/>
   <owl:allValuesFrom rdf:resource="#Grape" />
  </owl:Restriction> </rdfs:subClassOf> </owl:Class>
```

```
<Grape rdf:resource="fruit1" />
```

fruit1 type Grape

Sub-Properties

```
<owl:ObjectProperty rdf:ID="containedIn">
  <rdfs:domain rdf:resource="#spatialThing" />
  <rdfs:range rdf:resource="#spatialThing" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="locatedIn">
  <rdfs:subPropertyOf rdf:resource="#containedIn" />
</owl:ObjectProperty>
```

If *x* is locatedIn *y* then *x* is containedIn *y*

Transitive and Inverse Properties

```
<owl:ObjectProperty rdf:ID="containedIn">
  <rdfs:type rdf:resource="&owl;transitiveProperty" />
</owl:ObjectProperty>
```

If *x* is containedIn *y* and *y* is containedIn *z* then *x* is containedIn *z*

```
<owl:ObjectProperty rdf:ID="contains">
  <owl:inverseOf rdf:resource="#containedIn" />
</owl:ObjectProperty>
```

If *x* is containedIn *y* then *y* contains *x*

Implications

```
<place1 locatedIn place2>
<place2 locatedIn place3>
```

⊨

```
<place1 type SpatialThing>
<place2 type SpatialThing>
<place3 type SpatialThing>
<place1 containedIn place2>
<place2 containedIn place3>
<place1 containedIn place3>
<place3 contains place2>
<place3 contains place1>
<place2 contains place1>
```

```
<owl:Thing rdf:ID="place2" >
  <locatedIn rdf:resource="place1" />
</owl:Thing>
<owl:Thing rdf:ID="place1" >
```

```
<SpatialThing rdf:resource="place1" />
```

locatedIn subPropertyOf containedIn

containedIn is transitive

contains is the inverse of containedIn

Importance of Disjointness

- Many OWL constructs (e.g. range and domain) are necessary and *sufficient*
  - Many frame and OO systems use only necessary
- Result is that types are inferred
- To generate “type mismatch” errors requires an inconsistency

```
<owl:Class rdf:ID="Wine">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#madeFrom"/>
   <owl:allValuesFrom rdf:resource="#Grape" />
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Grape" />
<owl:Class rdf:ID="Color">
 <rdfs:subClassOf rdf:resource="#WineDescriptor"/>
 <owl:oneOf rdf:parseType="Collection">
  <owl:Thing rdf:about="#White"/>
  <owl:Thing rdf:about="#Rose"/>
  <owl:Thing rdf:about="#Red"/>
 </owl:oneOf>
</owl:class>

<wine1 type Wine>
<wine1 madeFrom White>
```

OK!! ⊨ <White type Grape>

```
<owl:Class rdf:resource="#Color">
  <owl:disjointWith rdf:resource="#Grape"/>
</owl:Class>
```

Leads to inconsistency

General Lessons

- Don’t argue about what an ontology *is*
- Building ontologies is *engineering*
- Knowledge-*based* systems rely on knowledge
- Using reasoning vs. specifying semantics
- Neither semantics nor web are new
- Difference between semantic web and RDF, OWL
- Difference between “Abox” and “Tbox”

For more information

- SW Best Practices WG:

  http://www.w3.org/2001/sw/BestPractices/
- Webont WG:

  http://www.w3.org/2001/sw/WebOnt/
- OWL Guide:

  http://www.w3.org/TR/owl-guide/
- OWL Overview:

  http://www.w3.org/TR/owl-guide/

This slide intentionally left blank.

Example

- <owl:Class rdf:ID="**Wine**">
- <rdfs:subClassOf rdf:resource="#**PotableLiquid**" />
- <rdfs:subClassOf>
- <owl:Restriction>
- <owl:onProperty rdf:resource="#**hasMaker**" />
- <owl:cardinality rdf:datatype="**http://www.w3.org/2001/XMLSchema#nonNegativeInteger**">**1**</owl:cardinality>
- </owl:Restriction>
- </rdfs:subClassOf>
- \- <rdfs:subClassOf>
- \- <owl:Restriction>
- <owl:onProperty rdf:resource="#**hasMaker**" />
- <owl:allValuesFrom rdf:resource="#**Winery**" />
- </owl:Restriction>
- </rdfs:subClassOf>
- \- <rdfs:subClassOf>
- \- <owl:Restriction>
- <owl:onProperty rdf:resource="#**madeFromGrape**" />
- <owl:minCardinality rdf:datatype="**http://www.w3.org/2001/XMLSchema#nonNegativeInteger**">**1**</owl:minCardinality>
- </owl:Restriction>
- </rdfs:subClassOf>
- \- <rdfs:subClassOf>
- \- <owl:Restriction>

Finite Universes

- OWL Full includes the *language elements* in the Universe of Discourse, OWL DL does not
- `OWL:THING` in OWL-DL is an `OWL:CLASS`
- Thus it is legal to say:
  - OWL:THING one-of (A)
  - In OWL-Full, this entails all language elements are the same
  - In OWL-DL, it does not