OWL

Ontology

- A rich and formal language to define both concrete and abstract knowledge
- Terminology+assertive knowledge, similar to schema+database

- **Class**: Person **Annotations**: ...
  SubClassOf: owl:Thing **that** hasFirstName **exactly** 1
  and hasFirstName **only string**[**minLength** 1] ,...
  SubClassOf: hasGender **exactly** 1
  and hasGender **only** {female , male}
  EquivalentTo: g:People ,... **DisjointWith**: g:Rock , g:Mineral ,...
  DisjointUnionOf: **Annotations**: ... Child, Adult
  HasKey: hasSSN
- **Individual**: John
  Types: Person ,
  hasFirstName value "John"
  or **hasFirstName value** "Jack"^^xsd:string
  Facts: hasWife Mary,
  not hasChild Susan, hasAge 33,
  SameAs: Jack ,...
  DifferentFrom: Susan ,...
- **EquivalentProperties**: hates, loathes, despises
- **DisjointProperties**: hates, loves, indifferent

Ontology

- A rich and formal language to define both concrete and abstract knowledge
- Terminology+assertive knowledge, similar to schema+database
- Essential difference: open world (true / do not know) vs. closed world (true / false)
- Terminology: a system of relations among terms

OWL2

- To express ontologies
- Two equivalent semantics
- Five syntactic presentations
  1. Functional syntax (human readable)
  2. RDF/XML (RDF with XML syntax)
  3. Turtle (triples)
  4. OWL 2 XML (syntax 1 in XML)
  5. **Manchester syntax**
- http://www.w3.org/TR/owl2-primer/

- **Class**: Person **Annotations**: ...
    SubClassOf: owl:Thing **that** hasFirstName **exactly** 1
        and hasFirstName **only string**[**minLength** 1] ,...
    SubClassOf: hasGender **exactly** 1
        and hasGender **only** {female , male}
    EquivalentTo: g:People ,... **DisjointWith**: g:Rock , g:Mineral ,...
    DisjointUnionOf: **Annotations**: ... Child, Adult
    HasKey: hasSSN
- **Daclaration** (**Class** (Person))
- **SubClassOf** (Person
    ObjectIntersectionOf
        (owl:Thing
        ObjectExactCardinality(1 hasFirstName owl:thing)))
- **EquivalentClasses** (Person g:People)

Parts of an ontology

- Entities: individual, classes or properties (relations)
  - Datatype properties and object properties
- Expressions: entity combinations:
  - Professor and Female
- Propositions (statements, axioms): boolean statements

Mary instance of class Person

- **ClassAssertion**(:Person :Mary)
- <Person rdf:about="Mary"/>
- :mary rdf:type :Person .
- **Individual:** Mary
  Types: Woman
- <ClassAssertion>
  <Class IRI="Person"/>
  <NamedIndividual IRI="Mary"/>
  </ClassAssertion>

Classification in OWL

- Mary may be declared to belong to many classes. Or no ClassAssertion at all
- More ClassAssertion's may be deduced from
  - Subclasses
  - Domain/range of properties
  - Other ways
- (Even more Class Assertions may hold in a model for this theory, this is Open World)

Class hierarchy

- **SubClassOf**(:Woman :Person)
- **Class**: Woman
- **SubClassOf**: Person
- Corresponds to set inclusion
- Double inclusion:
  - **EquivalentClasses**(:Person :Human)
  - **Class**: Person
    EquivalentTo: Human
- **EquivalentClasses** is n-ary
- (OWL subclasses are usually far more than those in a conceptual model)

Class disjointness

- **DisjointClasses**(:Woman :Man)
- **DisjointClasses**: Woman, Man
- Essential information that is often left unexpressed
- **DisjointClasses**(:Car :Person :Accident …)
- **DisjointUnion**(:Person :Young :Adult :Old)
- **DisjointUnionOf**: Person Young Adult Old

Association instances

- **ObjectPropertyAssertion**(:hasWife :John :Mary)
- **Individual**: John
- **Facts**: hasWife Mary

Manchester syntax

- Class: Person Annotations: ...
  SubClassOf: owl:Thing that hasFirstName exactly 1
  and hasFirstName only string[minLength 1] ,...
  SubClassOf: hasGender exactly 1
  and hasGender only {female , male}
  EquivalentTo: g:People ,... DisjointWith: g:Rock , g:Mineral ,...
  DisjointUnionOf: Annotations: ... Child, Adult
  HasKey: Annotations: ... hasSSN
- Individual: John
  Types: Person ,
  hasFirstName value "John«
  or hasFirstName value "Jack"^^xsd:string
  Facts: hasWife Mary,
  not hasChild Susan, hasAge 33,
  SameAs: Jack ,...
  DifferentFrom: Susan ,...
- EquivalentProperties: hates, loathes, despises
- DisjointProperties: hates, loves, indifferent

Associations

- Negative instances
  - **NegativeObjectPropertyAssertion**(:hasWife :Bill :Mary)
  - Very important, since OWL assumes an open world
  - NegativeClassAssertion does not exists, but we have a complement operation on classes
- **Individual**: Bill

  Facts: **not** hasWife Mary

Association inclusion

- Association inclusion:
  - **SubObjectPropertyOf**(:hasWife :hasSpouse)

- **ObjectProperty**: hasWife
  SubPropertyOf: hasSpouse

Domain restriction

- **ObjectPropertyDomain**(:hasWife :Man)
  ObjectPropertyRange(:hasWife :Woman)

- **ObjectProperty**: hasWife
  Domain: Man
  Range: Woman

Equality and disequality

- Are John and Bill the same person?
- **DifferentIndividuals**(:John :Bill :Matt :Dave)
  – Could I deduce that in some other way?
  – What can I deduce from this?
- **SameIndividual**(:John :Bill)
  – Could I deduce that in some other way?
  – What can I deduce from this?
- **DI** and **SI** are both n-ary

Manchester syntax

- **Individual**: John
  SameAs: Jack
  DifferentFrom: Bill, Matt, Dave

Attributes (datatype properties)

- **DataPropertyAssertion**(:hasAge :John "51"^^xsd:integer)
- **NegativeDataPropertyAssertion**(:hasAge :Jack "53"^^xsd:integer)
- Manchester syntax: same as for ObjectProperties

Attribute domain

- **DataPropertyDomain**(:hasAge :Person) **DataPropertyRange**(:hasAge xsd:nonNegativeInteger)
- Beware: it implies that only persons may have an age
- DataProperty: hasAge
  Domain: Person ,...
  Range: xsd:nonNegativeInteger,...

Class operators

- **EquivalentClasses**(
  :Mother
  ObjectIntersectionOf(:Woman :Parent))

  Mother ⇔ (Woman and Parent)
- **EquivalentClasses**(:Parent
  ObjectUnionOf(:Mother :Father))

  Parent ⇔ (Mother or Father)
- Union and Intersection are n-ary

Manchester syntax

- **Class**: Mother
  EquivalentTo: Woman **and** Parent
- **Class**: Parent
  EquivalentTo: Mother **or** Father

Class operators

- **EquivalentClasses**(:ChildlessPerson
  ObjectIntersectionOf(
  :Person
  ObjectComplementOf(:Parent)))
- **Class**: ChildlessPerson
  EquivalentTo: Person **and not** Parent

Operators and implication

- **SubClassOf**(
  :Grandfather
  ObjectIntersectionOf(:Man :Parent)
)
- GrandFather $\Rightarrow$ (Man and Parent)
- **Class**: Grandfather
  SubClassOf: Man **and** Parent

Class operators

- **ClassAssertion**(
    ObjectIntersectionOf(
        :Person
        ObjectComplementOf(:Parent)
)
    :Jack
)
- **Individual**: Jack
    Types: Person **and not** Parent
- Person(Jack) and not Parent(Jack)
- Jack $\in$ Person $\cap$ Compl(Parent)

Class operators

- ⟦**Intersection**(A,B)⟧ = { x | x∈⟦A⟧ and x∈⟦B⟧ }
- ⟦**Union(**A,B)⟧ = { x | x∈⟦A⟧ or x∈⟦B⟧ }
- ⟦**Complement**(A)⟧ = { x | not x∈⟦A⟧ }

Projection (Existential Quantification)

- ⟦**SomeValuesFrom**(R,A)⟧
  = { x | ∃y (x,y)∈⟦R⟧ and y∈⟦A⟧ }
- **EquivalentClasses**(:Parent
  ObjectSomeValuesFrom(:hasChild :Person)
)
- **Class**: Parent
  EquivalentTo: hasChild **some** Person
- ⟦SomeValuesFrom(R,A)⟧ $= \pi_1(R \bowtie_{l.2=r.1} A)$

Existential restriction

- **EquivalentClasses**(
  :Parent
  ObjectSomeValuesFrom(:hasChild :Person)
)
- Parent(x) ⇔ ∃y. hasChild(x,y) and Person(y)

Universal restriction

- $[\![$**AllValuesFrom**(R,A)$]\!]$
  $= \{ x \mid \forall y\ (x,y) \in [\![R]\!] \Rightarrow y \in [\![A]\!] \}$
- HappyPerson
  ⇔AllValuesFrom(hasChild, HappyPerson)
- **EquivalentClasses**(
  :HappyPerson
  ObjectAllValuesFrom(
  :hasChild :HappyPerson))
- **Class**: HappyPerson
  EquivalentTo: hasChild **only** HappyPerson

For all - exists

- **EquivalentClasses**(
  :HappyPerson
  ObjectIntersectionOf(
  ObjectAllValuesFrom(
  :hasChild :HappyPerson
)
  ObjectSomeValuesFrom(
  :hasChild :HappyPerson
)))
- **Class**: HappyPerson
  EquivalentTo: hasChild **only** HappyPerson
  and
  hasChild **some** HappyPerson

Relating individuals and classes

- $[\![$**HasValue**(R,a)$]\!]$
  $= \{ x \mid (x,a) \in [\![R]\!] \}$
- HasValue(R,a) = SomeValuesFrom(R,{a})
- **EquivalentClasses**(
      :JohnsChildren
      ObjectHasValue(:hasParent :John)
)
- **Class**: JohnsChildren
      EquivalentTo: hasParent **value** John

Diagonal projection

- ⟦**ObjectHasSelf(R**)⟧
  $= \{ x \mid (x,x) \in [\![R]\!] \}$
- **EquivalentClasses**(
  :NarcisticPerson
  ObjectHasSelf(:loves)
)
- **Class**: NarcisticPerson
  EquivalentTo: loves **Self**

Cardinality

- **ClassAssertion**(
  ObjectMaxCardinality(4 :hasChild :Parent)
  :John)
- ⟦**MaxCardinality**(n,R,A)⟧
  $= \{ x \mid |\{y \mid (x,y) \in [\![R]\!] \text{ and } y \in [\![A]\!]\}| \leq n \}$
- ⟦**MinCardinality**(n,R,A)⟧
  $= \{ x \mid |\{y \mid (x,y) \in [\![R]\!] \text{ and } y \in [\![A]\!]\}| \geq n \}$
- **ExactCardinality**(n,R,A)
- MaxCardinality(n,R) = MaxCardinality(n,R,owl:Thing)

Manchester syntax

- **Individual**: John

  Types: hasChild **max** 4 Parent
- **Individual**: John

  Types: hasChild **min/exactly** 4 Parent
- **Individual**: John

  Types: hasChild **max/min/exactly** 4

Enumeration

- ⟦**ObjectOneOf**(a … z))⟧
  = { a,…,z }
- **EquivalentClasses**(
  :MyBirthdayGuests
  ObjectOneOf(:Bill :John :Mary)
)
- **Class**: MyBirthdayGuests
  EquivalentTo: { Bill, John, Mary }
- What can I now deduce from:
  - **ClassAssertion**(:MyBirthdayGuests :Jim)

Association inversion

- Declaring inversion among two properties:
  - **InverseObjectProperties**(:hasParent :hasChild)
  - **EquivalentObjectProperties**
    (:hasParent **ObjectInverseOf**(:hasChild))
- **ObjectProperty**: hasParent
  InverseOf: hasChild

Inverting one property:

- **EquivalentClasses**(
  :Orphan
  ObjectAllValuesFrom(
  ObjectInverseOf(:hasChild)
  :Dead
)
)
- **Class**: Orphan
  EquivalentTo: **inverse** hasChild **only** Dead

Association disjointness

- **DisjointObjectProperties**(:hasFather :hasMother :hasSpouse)
  - **Disjoint**(R,S): $\forall$x,y. R(x,y) => not S(x,y)
- **DisjointProperties**: hasFather, hasMother, hasSpouse

Symmetric and Asymmetric

- **SymmetricObjectProperty**(:hasSpouse)
  - Symmetric(R): ∀x,y. R(x,y) => R(y,x)
- **ObjectProperty**: hasSpouse
  Characteristics: **Symmetric**
- **AsymmetricObjectProperty**(:hasChild)
  - Asymmetric(R): ∀x,y. R(x,y) => not R(y,x)
- **ObjectProperty**: hasChild
  Characteristics: **Asymmetric**

Reflexive and Irreflexive

- **ReflexiveObjectProperty**(:hasRelative)
  - Reflexive(R): $\forall$x. R(x,x)
    ($\forall$x,y. R(x,y) or R(y,x) => R(x,x))
- **ObjectProperty**: hasRelative
  Characteristics: **Reflexive**
- **IrreflexiveObjectProperty**(:parentOf)
  - Irreflexive(R): $\forall$x. not R(x,x)
- **ObjectProperty**: parentOf
  Characteristics: **Irreflexive**

Functionality

- **FunctionalObjectProperty**(:hasHusband)
  - **Functional**(R): $\forall$x,y,z. R(x,y) and R(x,z) => y=z
- **InverseFunctionalObjectProperty**(R): Functional(Inverse(R))
- **ObjectProperty**: hasHusband

  Characteristics: **Functional**,

  InverseFunctional

Association transitivity

- **TransitiveObjectProperty**(:hasAncestor)
  - **Transitive**(R): $\forall$x,y,z. R(x,y) and R(y,z) => R(x,z)

- **SubObjectPropertyOf**(:hasParent :hasAncestor)

- **ObjectProperty**: hasAncestor
  Characteristics: **Transitive**
  SubClassOf: hasParent

Chain generalization

- **SubObjectPropertyOf**(
  ObjectPropertyChain(:hasParent :hasParent)
  :hasGrandparent)
  - $[\![\text{Chain}(R,S)]\!]$
    $= \{ (x,z) \mid \exists y.\ (x,y) \in [\![R]\!] \text{ and } (y,z) \in [\![S]\!] \}$
- **ObjectPropertyChain** accepts *n* arguments
- **ObjectProperty**: hasGrandparent
  SubPropertyChain: hasParent **o** hasParent

ObjectPropertyChain

- **ObjectPropertyChain**(P1 P2 P3) is not a general purpose association operator: it can only be used at the left hand side of an inclusion
- The only association constructor is **ObjectInverseOf**(P)
- Moreover, chain inclusion cannot be cyclic, as in:
  - P1.P2 $\subseteq$ P3
  - P3.P4 $\subseteq$ P5
  - P5 $\subseteq$ P2

Keys

- **HasKey** (:Person () (:hasSSN))
  - **HasKey**(C (P1 … Pn) (D1 … Dm)):
    ∀x,y named and such that C(x) and C(y)
    (∃z1...zn+m.
    and P1(x,z1) and P2(y,z1) and ...
    and Dm(x,zn+m) and Dm(y,zn+m) => x=y
- **Class**: Person
  HasKey: hasSSN

OWL and FOL

- Every student is a person
  - FOL: $\forall x.\ x \in Student \Rightarrow x \in Person$
  - OWL: $Student \subseteq Person$
- If somebody is adult and is not working, is unemployed
  - FOL: $\forall x.\ x \in Adult$ and $not(x \in Working) \Rightarrow x \in Unemployed$
  - OWL: $(Adult \cap \neg(Working)) \subseteq Unemployed$

OWL and FOL

- If somebody is unemployed , then is adult and is not working
  - FOL: $\forall x.\ x \in Unemployed \Rightarrow$
    $x \in Adult$ and not($x \in Working$)
  - OWL: $Unemployed \subseteq (Adult \cap \neg(Working))$
- An unemployed is (defined as) an adult who is not working
  - FOL: $\forall x.\ x \in Unemployed \Leftrightarrow$
    $x \in Adult$ and not($x \in Working$)
  - OWL: $Unemployed = (Adult \cap \neg(Working))$

OWL and FOL

- If somebody has a young child then is a recent parent
  - FOL: $\forall x. (\exists y. ParentOf(x,y) \text{ and } Young(y) \Rightarrow RecentParent(x))$
  - OWL: $ObjSomeValues(ParentOf, Young) \subseteq RecentParent$
- If a male has a young child then is a recent father
  - $\forall x. (Male(x) \text{ and } \exists y. ParentOf(x,y) \text{ and } Young(y) \Rightarrow RecentFather(x))$
  - $(Male \cap ObjSomeValues(ParentOf, Young)) \subseteq RecentFather$

OWL and FOL: binary inclusions

- Binary inclusions
  - FOL: $\forall$x,y. (P(x,y) $\Rightarrow$ Q(x,y))
  - OWL: ObjSubProperty(P,Q)
  - OWL: P $\subseteq$ Q , or P $\subseteq^2$ Q

Unary vs binary inclusion

- Unary inclusions: in OWL we have many operations that yield sets:
  - Property projections (ObjSome/All/HasValues, ObjHasSelf, ObjectMax/Min/ExactCardinality)
  - Listing: ObjOneOf
  - Boolean ops: $\cap$, $\cup$, $\neg$
- Binary: only one operation yield a property (ObjInverseOf), plus we can use ObjPropertyChain at the left hand side:
  - $P \cdot Q^{-1} \cdot R \subseteq^2 S$

Binary inclusion in OWL

- $P \subseteq^2 R$
- $P \cdot Q^{-1} \cdot R^{-1} \subseteq^2 S$
- In FOL:
  - $\forall x,y.\ P(x,y) \Rightarrow R(x,y)$
  - $\forall x,y.\ (\exists w,w'.\ P(x,w) \text{ and } Q^{-1}(w,w') \text{ and } R^{-1}(w',y)$
    $\Rightarrow S(x,y)\ )$
  - $\forall x,y.\ (\exists w,w'.\ P(x,w) \text{ and } Q(w',w) \text{ and } R(y,w')$
    $\Rightarrow S(x,y)\ )$

Binary inclusion in OWL

- If x and y have a common parent, they are brothers
  - $\forall x,y.\ (\exists w.\ \text{ParentOf}(w,x) \text{ and } \text{ParentOf}(w,y)$
    $\Rightarrow \text{Sibling}(x,y)\ )$
  - OWL: $\text{ParentOf}^{-1} \cdot \text{ParentOf} \subseteq^{2} \text{Sibling}$
- What happens if I add:
  - IrreflexiveProperty(Sibling) ?
- If x and y are bothers, they have a common parent:
  - $\forall x,y.\ (\text{Sibling}(x,y) \Rightarrow \exists w.\ \text{ParentOf}(w,x) \text{ and } \text{ParentOf}(w,y)\ )$
  - OWL?

Atomic types restrictions

- **DatatypeDefinition**(
    :personAge
    DatatypeRestriction(
        xsd:integer
        xsd:minInclusive
        "0"^^xsd:integer
        xsd:maxInclusive
        "150"^^xsd:integer
)
)

Atomic types restrictions

- **SubClassOf**(
  :Teenager
  DataSomeValuesFrom(
  :hasAge
  DatatypeRestriction(
  xsd:integer
  xsd:minExclusive
  "12"^^xsd:integer
  xsd:maxInclusive
  "19"^^xsd:integer)
))

Atomic types properties

- **FunctionalDataProperty**(:hasAge)

Atomic types enumeration

- **DatatypeDefinition**(

```
    :toddlerAge
    DataOneOf(
        "1"^^xsd:integer
        "2"^^xsd:integer
    )
)
```

Annotations

- No semantic effect
- AnnotationAssertion(
      rdfs:comment
      :Person
      "Represents the set of all people.“
)
- Etc…

Prefixes

- Prefix(:   =<http://example.com/owl/families/>)
  Prefix(otherOnt:
    =<http://example.org/otherOntologies/families/
)
  Prefix(xsd:
    =<http://www.w3.org/2001/XMLSchema#>
)
  Prefix(owl: =<http://www.w3.org/2002/07/owl#>)
  Ontology(<http://example.com/owl/families> ...
)

Declarations

- **Declaration**(**NamedIndividual**(:John))
  Declaration(**Class**(:Person))
  Declaration(**ObjectProperty**(:hasWife))
  Declaration(**DataProperty**(:hasAge))
- Every class and property that is used must be declared

Type constraints

- Properties
  - Each used property must be declared, with only one kind (object/data/annotation)
- Class/datatype:
  - Each used class or datatype must be declared, with only one kind (class or datatype)

Manchester syntax

```
ObjectProperty: hasWife
   Characteristics: Functional, InverseFunctional, Reflexive,...
   Domain: Annotations: rdfs:comment "General domain",
             Person,
           Annotations: rdfs:comment "More specific domain"
             Man
   Range: Person, Woman
   SubPropertyOf: hasSpouse, loves
   EquivalentTo: isMarriedTo
   DisjointWith: hates ,...
   InverseOf: hasSpouse, inverse hasSpouse
   SubPropertyChain: hasChild o hasParent o...
```

Class

```
Class: Person
   Annotations: ...
   SubClassOf: owl:Thing that hasFirstName exactly 1
            and hasFirstName only string[minLength 1]  ,...
   SubClassOf: hasAge exactly 1 and hasAge only not NegInt,...
   SubClassOf: hasGender exactly 1 and hasGender only {female , male}
,...
   SubClassOf: hasSSN max 1, hasSSN min 1
   SubClassOf: not hates Self, ...
   EquivalentTo: g:People ,...
   DisjointWith: g:Rock , g:Mineral ,...
   DisjointUnionOf: Annotations: ... Child, Adult
   HasKey: Annotations: ... hasSSN
```

Individual

```
Individual: John
  Annotations: ...
  Types: Person ,
    hasFirstName value "John“
      or hasFirstName value "Jack"^^xsd:string
  Facts: hasWife Mary, not hasChild Susan,
      hasAge 33, hasChild _:child1
  SameAs: Jack ,...
  DifferentFrom: Susan ,...
```

Other

DisjointClasses: Rock, Scissor, Paper

EquivalentProperties: hates, loathes, despises

DisjointProperties: hates, loves, indifferent

EquivalentProperties: favoriteNumber, favoriteInteger

DisjointProperties: favoriteInteger, favoriteReal

SameIndividual: John, Jack, Joe, Jim

DifferentIndividuals: John, Susan, Mary, Jill

Semantics

- Direct semantics: logical interpretation
- RDF based semantics: the OWL ontology is mapped to an RDF graph, that is interpreted
- In practice, the two interpretations yield the same notion of deduction

Restrictions on OWL DL

- No cyclic chain inclusions:
  - SubObjectPropertyOf(ObjectPropertyChain(*a:hasFather a:hasBrother*) *a:hasUncle*)
  - SubObjectPropertyOf(ObjectPropertyChain(*a:hasChild a:hasUncle*) *a:hasBrother*)
- Simple recursion is allowed:
  - SubObjectPropertyOf(ObjectPropertyChain(*a:hasChild a:hasSibling*) *a:hasChild*)

OWL DL vs OWL Full

- OWL DL is a wide subset of OWL Full, but OWL DL has decidable inference
- OWL DL has a logical semantics (direct semantics)
- OWL Full has an RDF based semantics
- On OWL DL the two semantics coincide

Meta-reasoning (punning)

- In OWL 2 DL the same IRI can be used for an object and for a class:
  - ClassAssertion(:Father :John)
    ClassAssertion(:SocialRole :Father)
- In RDF:
  - <Father rdf:about="John"/>
    <SocialRole rdf:about="Father"/>
- In OWL 2 DL the two uses of the IRI are semantically distinct

Profiles

- Subsets of OWL with a good expressive power and a reasonable complexity

Profiles

- OWL 2 EL: efficient (PTime) on big ontologies
- OWL 2 QL: to manage many individuals and simple ontologies with relational techniques
- OWL 2 RL: similar to QL, oriented towards huge sets of RDF triples with rule-based implementations

OWL 2 EL (Existential Logic)

- Designed for big and complex ontologies
- Class expressions are quite expressive
- No disjunction, inversion, negation and universal quantification on associations

Tool classes

- Editors
- Reasoners