Unit 6: OWL, OWL 2, SPARQL+OWL
Some facts about OWL…

- OWL stands for Web Ontology Language
- Strongly Simplified: OWL is an Ontology language with an RDF syntax
  - There are different syntaxes for OWL, we will focus on RDF syntax here, but occasionally use DL syntax or First-order logics notation for explanation.
- OWL extends RDF Schema by more expressive constructs.
- A fragment of OWL is expressible in Description Logics (sometimes referred to as OWL DL)
- The original OWL standards date back to 2004 (also sometimes referred to as OWL 1)
- There was a significant revision in 2008 (also often referred to as OWL 2)
In today’s lecture:

- OWL 1 Overview
- OWL 2 new features
- OWL 2 tractable fragments: EL, QL, RL
- OWL + SPARQL

Disclaimer: We will only be able to scratch the surface (e.g. not be able to give an in-depth Description Logics introduction)
OWL 1 Overview:

See Lecture 3 slides 31ff.
Why OWL1 is Not Enough

Too expensive to reason with
- High complexity: Satisfiability checking is NEXPTIME-complete
- Some ontologies only use some limited expressive power; e.g. The SNOMED (Systematised Nomenclature of Medicine) ontology

Not expressive enough; e.g.
- No user defined datatypes
  [Pan 2004; Pan and Horrocks 2005; Motik and Horrocks 2008]
- No metamodeling support
  [Pan 2004; Pan, Horrocks, Schreiber, 2005; Motik 2007]
- Limited support for modeling relations between properties
  [Horrocks et al., 2006]
Since 2009: OWL 2: A new version of OWL

Two Main goals:

1. **To define “profiles” of OWL that are:**
   - smaller, easier to implement and deploy
   - cover important application areas and are easily understandable to non-expert users

2. **To add a few extensions to current OWL that are useful, and are known to be implementable**
   - many things happened in research since 2004 in research

---

**OWL 2 Web Ontology Language Document Overview**

**W3C Recommendation 27 October 2009**
New Expressiveness in OWL 2

New expressive power

- DatatypeDefinitions: user defined datatypes using XSD restrictions, e.g.

```turtle
:personAge owl:equivalentClass
[ a rdfs:Datatype ;
  owl:onDatatype xsd:integer ;
  owl:withRestrictions
    ( xsd:minInclusive "0"^^xsd:integer
    xsd:maxInclusive "150"^^xsd:integer ) ] .
```

```turtle
dbpedia:Elizabeth_II :age "86"^^:personAge
```

- punning (metamodeling), e.g.:

```turtle
:John a :Father .
:Father a :SocialRole .
```
New Expressiveness in OWL 2

New expressive power on properties

- Qualified cardinality restrictions
- Property chain axioms
- Local reflexivity restrictions
- reflexive, irreflexive, symmetric, and antisymmetric properties
- Disjoint properties
- keys
Qualified cardinality restrictions

- In OWL 1 you could only make general cardinality restrictions, e.g. we were cheating here:

  A Senior researcher is a foaf:Person who isAuthorOf 10+ Publications

  $$ex:Senior \equiv foaf:Person \sqcap \geq 10 \; ex:isAuthorOf \sqcap \exists ex:isAuthorOf.ex:Publication$$

What we really wanted to say (but which wasn’t expressible in OWL1)

$$ex:Senior \equiv foaf:Person \sqcap \geq 10 \; ex:isAuthorOf.ex:Publication$$
Qualified cardinality restrictions

- In OWL 1 you could only make general cardinality restrictions, e.g. we were cheating here:

A Senior researcher is a foaf:Person who isAuthorOf 10+ Publications

```owl
ex:Senior owl:intersectionOf (  
  foaf:Person  
  [ a owl:Restriction; owl:onProperty ex:isAuthorOf ; owl:minCardinality 10 ]  
  [ a owl:Restriction; owl:onProperty ex:isAuthorOf ; owl:someValuesFrom ex:Publication ] ).
```

What we really wanted to say (but which wasn’t expressible in OWL1)

```owl
ex:Senior owl:intersectionOf (  
  foaf:Person  
  [ a owl:Restriction; owl:onProperty ex:isAuthorOf ; owl:minQualifiedCardinality 10  
```
Property Chain axioms:

- E.g. could be useful to tie sioc:name and foaf:nick via foaf:holdsAccount:

\[(\text{foaf:holdsAccount} \circ \text{sioc:name}) \sqsubseteq \text{foaf:nick}\]

\[
\text{foaf:nick} \ \text{owl:propertyChainAxiom} \ (\text{foaf:holdsAccount} \ \text{sioc:name}) \ .
\]
local reflexivity restrictions

\[\text{Narcissist} \equiv \exists \text{loves.self}()\]

\text{Narcissist} \equiv \exists \text{loves.self}()

\text{Narcissist} \equiv \exists \text{loves.self}()

\text{:Narcissist owl:equivalentClass}

[ a owl:Restriction ;
owl:onProperty :loves ;
owl:hasSelf "true"^^xsd:boolean ] .

\text{:Narcissist owl:equivalentClass}

[ a owl:Restriction ;
owl:onProperty :loves ;
owl:hasSelf "true"^^xsd:boolean ] .
In OWL 1, you can define that a property is functional, transitive, symmetric, inverse functional... 

```
owl:SymmetricProperty
owl:FunctionalProperty
owl:InverseFunctionalProperty
owl:TransitiveProperty
```

... additional property features in OWL2: reflexive, irreflexive, and asymmetric, properties.

```
owl:ReflexiveProperty
owl:IrreflexiveProperty
owl:AsymmetricProperty
```
Disjoint properties:

In OWL 1 disjointness can only be asserted for classes

```
```

In OWL2 also allowed to assert disjointness of Properties

```
```
Multi-attribute Keys now possible in OWL 2, e.g. foaf:OnlineAccount/members are uniquely identified by a combination of foaf:accountName and foaf:accountServiceHomepage:

foaf:OnlineAccount owl:hasKey

(foaf:accountName foaf:accountServiceHomepage) .
Syntactic sugar (make things easier to say)

- Disjoint unions, e.g.:
  
  Element owl:DisjointUnionOf (Metal Wood Water Fire Earth)

- Disjoint classes, and properties e.g.:

  [ a owl:AllDisjointClasses ;
    owl:members ( University Department Professor Student ) ] .

  [ a owl:AllDisjointProperties ;
    owl:members ( spouseOf childOf grandChildOf ) ] .

- More Syntactic sugar for Negative assertions, e.g.:

  
  owl:NegativePropertyAssertion

  allows to state negated facts, such as (but the RDF syntax for it looks quite ugly ;-)):

  \[ \neg \text{childOf} (adam, eve) \]

  * Note: this is already expressible in OWL1: \( \{adam\} \subseteq \neg \exists \text{childOf}.\{eve\} \)
\( S \) used for ALC with role transitivity (also reflexivity, symmetry)

\( H \) used for role hierarchy

\( R \) (subsumes \( H \)) often used for with role (property chain) inclusion axioms.

**Additional letters** indicate other extensions, e.g.:

- \( S \) for property characteristics (e.g., reflexive and symmetric)
- \( O \) for **nominals**/singleton classes
- \( I \) for inverse roles
- \( N \) for unqualified number restrictions
- \( Q \) for qualified number restrictions

property characteristics (\( S \)) + \( R \) + nominals (\( O \)) + inverse (\( I \)) + qualified number restrictions(\( Q \)) = **SROIQ**

**SROIQ** [Horrocks et al., 2006] is the basis for **OWL 2 DL**
OWL 2 Profiles

Rationale:
- Tractable
- Tailored to specific reasoning services

Popular reasoning services
- Instance reasoning: OWL 2 RL
- Query answering: OWL 2 QL
- Terminological reasoning (reasoning about classes and Properties): OWL 2 EL

Specification: [http://www.w3.org/TR/owl2-profiles/](http://www.w3.org/TR/owl2-profiles/)
The family tree

OWL 2 Full

OWL 2 DL

OWL 2 SROIQ

OWL 2 SHOIN

OWL 2 QL

OWL 2 RL

OWL 2 EL

OWL 2 DL-Lite

Undecidable

2NExpTime-Complete

NExpTime-Complete

PTime-Complete

In AC^0
OWL 2 RL: OWL reasoning via rules
Ontologies: Example FOAF

```
foaf:knows rdfs:domain foaf:Person
\exists knows. T \subseteq Person
foaf:knows rdfs:range foaf:Person
\exists knows^- . T \subseteq Person

foaf:Person rdfs:subClassOf foaf:Agent
Person \subseteq Agent

foaf:homepage rdf:type owl:inverseFunctionalProperty .
T \subseteq 1homepage^- 
```

...
Recall that the semantics of RDFS can be expressed as (Datalog like) rules:

\[
\]

\[
\]

\[
\]

\[
\text{rdfs4: } \{ ?S \ ?P2 \ ?O \} :- \{ ?S \ ?P1 \ ?O . \ ?P1 \text{ rdfs:subPropertyOf } ?P2 . \}
\]

Some OWL Reasoning e.g. inverseFunctionalProperty can also be expressed by Rules:

owl1: { ?S1 owl:SameAs ?S2 } :- 


→ OWL 2 RL is the maximal fragment of OWL DL such that reasoning can be expressed in Rules!
Example OWL 2 RL inference:

Rules of the previous slides are sufficient e.g. for the example I showed you last time:

```
<http://dbpedia.org/resource/Tim_Berners-Lee>
  foaf:homepage 
    <http://www.w3.org/People/Berners-Lee/> .

foaf:name rdfs:subPropertyOf rdfs:label .
foaf:homepage a owl:InverseFunctionalProperty .

<http://dblp.l3s.de/d2r/page/authors/Tim_Berners-Lee>
  foaf:homepage 
    <http://www.w3.org/People/Berners-Lee/> ;
  foaf:name "Tim Berners-Lee".
```

by owl1 \[\rightarrow\] \(<.../dblp.../Tim_Berners-Lee> owl:sameAs <.../dbpedia.../Tim_Berners-Lee>\).
by owl2 \[\rightarrow\] \(<.../dbpedia.../Tim_Berners-Lee> foaf:name "Tim Berners-Lee"\).
by rdfs4 \[\rightarrow\] \(<.../dbpedia.../Tim_Berners-Lee> rdfs:label "Tim Berners-Lee"\).

```
SELECT ?P ?O
WHERE { <http://dbpedia.org/resource/Tim_Berners-Lee> rdfs:label ?O } 

?O

"Tim Berners-Lee"
```
RDFS+OWL inference in OWL 2 RL, what’s missing?

Note: Not all of OWL Reasoning can be expressed in Datalog, e.g.:

```
foaf:Person owl:disjointWith foaf:Organisation
```

Can be written/and reasoned about with FOL/DL reasoners:

- **FOL Syntax:** $\forall X. Person(X) \supset \neg Organisation(X)$
- **DL Syntax:** $\text{Person} \sqcap \text{Organisation} \sqsubseteq \bot$

But can be “approximated” by Rules (this is what is done in OWL 2 RL):

```
```
Some expressions are only allowed on one side of a subclassOf axiom, e.g.

$$\exists isAuthorOf.PUBLICATION \sqsubseteq \text{Scientist}$$

is ok, can be covered by a simple Datalog-style rule:

$$
\{ \ ?S \ a \ ?D \ \} \ :- \ \{ \ \text{[owl:onProperty \ ?P \ ; \ owl:someValuesFrom \ ?C]} \\
\text{rdfs:subClassOf \ ?D}. \\
\ ?S \ ?P \ ?O . \ ?O \ a \ ?C . \ \}
$$

But not the other way around (would need a rule with “existential” in the head):

$$
\text{Scientist} \sqsubseteq \ \exists \ isAuthorOf.PUBLICATION
$$

This is why OWL 2 RL forbids e.g. certain constructs on the right/left-hand-side of rdfs:subClassOf.
A (near maximal) fragment of OWL 2 such that
- Data complexity of conjunctive query answering in $\text{AC}^0$

Based on DL-Lite family of description logics [Calvanese et al. 2005; 2006; 2008]

Can exploit query rewriting based reasoning technique
- Computationally optimal
- Data storage and query evaluation can be delegated to standard RDBMS or RDF Store/SPARQL engine.
- Novel technique to prevent exponential blowup produced by rewritings [Kontchakov et al. 2010, Rosati and Almatelli 2010]
- Can be extended to more expressive languages (beyond $\text{AC}^0$) by delegating query answering to a Datalog engine [Perez-Urbina et al. 2009]
Given ontology $\mathcal{O}$ and query $Q$, use $\mathcal{O}$ to rewrite $Q$ as $Q'$ s.t., for any set of ground facts $\mathcal{A}$:
- $\text{ans}(Q, \mathcal{O}, \mathcal{A}) = \text{ans}(Q', \emptyset, \mathcal{A})$

Use (GAV) mapping $\mathcal{M}$ to map $Q'$ to SQL query
Query Rewriting Technique (basics)

Given ontology $\mathcal{O}$ and query $Q$, use $\mathcal{O}$ to rewrite $Q$ as $Q'$ s.t., for any set of ground facts $\mathcal{A}$:

- $\text{ans}(Q, \mathcal{O}, \mathcal{A}) = \text{ans}(Q', \emptyset, \mathcal{A})$

Resolution based query rewriting
- **Clausify** ontology axioms (using Skolemization)
- **Saturate** (clausified) ontology and query using resolution
- **Prune** redundant query clauses
Query Rewriting Technique (basics)

Example:

\[
\text{Doctor} \sqsubseteq \exists \text{treats.Patient} \\
\text{Consultant} \sqsubseteq \text{Doctor}
\]

\[
Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y)
\]
Query Rewriting Technique (basics) - Clausify

Example:

\[
\text{Doctor} \sqsubseteq \exists \text{treats} . \text{Patient} \\
\text{Consultant} \sqsubseteq \text{Doctor}
\]

\[
\begin{align*}
\text{treats}(x, f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Patient}(f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Doctor}(x) & \leftarrow \text{Consultant}(x)
\end{align*}
\]

\[
Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y)
\]

*Clausified ontology*
Example:

\[
\begin{align*}
\text{Doctor} & \subseteq \exists \text{treats}\cdot \text{Patient} \\
\text{Consultant} & \subseteq \text{Doctor} \\
\text{treats}(x, f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Patient}(f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Doctor}(x) & \leftarrow \text{Consultant}(x) \\
Q(x) & \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \\
Q(x) & \leftarrow \text{Doctor}(x) \land \text{Patient}(f(x))
\end{align*}
\]
Example:

\[
\begin{align*}
\text{Doctor} & \subseteq \exists \text{treats.Patient} \\
\text{Consultant} & \subseteq \text{Doctor} \\
\text{treats}(x, f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Patient}(f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Doctor}(x) & \leftarrow \text{Consultant}(x) \\
Q(x) & \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \\
Q(x) & \leftarrow \text{Doctor}(x) \land \text{Patient}(f(x)) \\
Q(x) & \leftarrow \text{treats}(x, f(x)) \land \text{Doctor}(x)
\end{align*}
\]
Example:

\[
\begin{align*}
\text{Doctor} & \subseteq \exists \text{treats.Patient} \\
\text{Consultant} & \subseteq \text{Doctor} \\
\text{treats}(x, f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Patient}(f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Doctor}(x) & \leftarrow \text{Consultant}(x) \\
Q(x) & \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \\
Q(x) & \leftarrow \text{Doctor}(x) \land \text{Patient}(f(x)) \\
Q(x) & \leftarrow \text{treats}(x, f(x)) \land \text{Doctor}(x) \\
Q(x) & \leftarrow \text{Doctor}(x)
\end{align*}
\]
Example:

\[
\begin{align*}
\text{Doctor} & \sqsubseteq \exists \text{treats. Patient} \\
\text{Consultant} & \sqsubseteq \text{Doctor} \\
\text{treats}(x, f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Patient}(f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Doctor}(x) & \leftarrow \text{Consultant}(x) \\
\end{align*}
\]

\[
\begin{align*}
Q(x) & \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \\
Q(x) & \leftarrow \text{Doctor}(x) \land \text{Patient}(f(x)) \\
Q(x) & \leftarrow \text{treats}(x, f(x)) \land \text{Doctor}(x) \\
Q(x) & \leftarrow \text{Doctor}(x) \\
Q(x) & \leftarrow \text{Consultant}(x)
\end{align*}
\]
Example:

\[
\text{Doctor} \sqsubseteq \exists \text{treats.Patient} \\
\text{Consultant} \sqsubseteq \text{Doctor}
\]

\[
\begin{align*}
\text{treats}(x, f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Patient}(f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Doctor}(x) & \leftarrow \text{Consultant}(x)
\end{align*}
\]

\[
\begin{align*}
Q(x) & \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \\
Q(x) & \leftarrow \text{Doctor}(x) \land \text{Patient}(f(x)) \\
Q(x) & \leftarrow \text{treats}(x, f(x)) \land \text{Doctor}(x) \\
Q(x) & \leftarrow \text{Doctor}(x) \\
Q(x) & \leftarrow \text{Consultant}(x)
\end{align*}
\]

The result is a union of conjunctive queries

\[
Q(x) \leftarrow (\text{treats}(x, y) \land \text{Patient}(y)) \lor \text{Doctor}(x) \lor \text{Consultant}(x)
\]
*Could* be used to answer some SPARQL queries over ontologies:

Example

\begin{align*}
\text{Doctor} & \sqsubseteq \exists \text{treats. Patient} \\
\text{Consultant} & \sqsubseteq \text{Doctor}
\end{align*}

Original Query:

\[ Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \]

The resulting union of conjunctive queries:

\[ Q(x) \leftarrow (\text{treats}(x, y) \land \text{Patient}(y)) \lor \text{Doctor}(x) \lor \text{Consultant}(x) \]

\begin{verbatim}
UNION { ?X a :Doctor .}
UNION { ?X a :Consultant.} }
\end{verbatim}
→ OWL 2 QL is the maximal fragment of OWL DL such that Query Answering can be expressed by (polynomial) Query rewriting techniques!

Again: several restrictions on what can and can’t be used, e.g. owl:sameAs is not allowed in OWL 2 QL … unfortunately, in the general case, non-distinguished variables can make trouble…
A (near maximal) fragment of OWL 2 such that
- Satisfiability checking is in PTime (PTime-Complete)
- Data complexity of query answering also PTime-Complete
Based on $\mathcal{EL}$ family of description logics [Baader et al. 2005]
Can exploit saturation based reasoning techniques
- Computes complete classification in “one pass”
- Computationally optimal (PTime for EL)
- Can be extended to Horn fragment of OWL DL [Kazakov 2009]

Will skip over this since it’s mainly useful for terminological reasoning, less for query answering...
Normalise ontology axioms to standard form:

\[ A \subseteq B \quad A \cap B \subseteq C \quad A \subseteq \exists R.B \quad \exists R.B \subseteq C \]

Saturate using inference rules:

\[ A \subseteq B \quad A \subseteq C \quad \overline{A \subseteq D} \]

\[ A \subseteq \exists R.B \quad B \subseteq C \quad \overline{\exists R.C \subseteq D} \]

(This is a simplification, the whole EL requires (many) more rules)
Saturation-based Technique (basics)

Example:

\[
\text{OrganTransplant} \equiv \text{Transplant} \sqcap \exists \text{site.Organ} \\
\text{HeartTransplant} \equiv \text{Transplant} \sqcap \exists \text{site.Heart} \\
\text{Heart} \sqsubseteq \text{Organ}
\]

\[
\text{OrganTransplant} \sqsubseteq \text{Transplant} \\
\text{OrganTransplant} \sqsubseteq \exists \text{site.Organ} \\
\exists \text{site.Organ} \sqsubseteq \text{SO} \\
\text{Transplant} \sqcap \text{SO} \sqsubseteq \text{OrganTransplant} \\
\text{HeartTransplant} \sqsubseteq \text{Transplant} \\
\text{HeartTransplant} \sqsubseteq \exists \text{site.Heart} \\
\exists \text{site.Heart} \sqsubseteq \text{SH} \\
\text{Transplant} \sqcap \text{SH} \sqsubseteq \text{HeartTransplant} \\
\text{Heart} \sqsubseteq \text{Organ}
\]
Saturation-based Technique (basics)

Example:

\[
\begin{align*}
\text{OrganTransplant} & \equiv \text{Transplant} \sqcap \exists \text{site.Organ} \\
\text{HeartTransplant} & \equiv \text{Transplant} \sqcap \exists \text{site.Heart} \\
\text{Heart} & \sqsubseteq \text{Organ} \\
\end{align*}
\]

\[
A \sqsubseteq \exists R.B \quad B \sqsubseteq C \quad \exists R.C \sqsubseteq D
\]

\[
A \sqsubseteq D
\]
Saturation-based Technique (basics)

Example:

\[\text{OrganTransplant} \equiv \text{Transplant} \cap \exists \text{site.\,Organ}\]
\[\text{HeartTransplant} \equiv \text{Transplant} \cap \exists \text{site.\,Heart}\]
\[\text{Heart} \subseteq \text{Organ}\]

\[\begin{align*}
A \subseteq & \exists R.B \\
B \subseteq & C \\
\exists R.C \subseteq & D \\
\hline
A \subseteq & D \\
\end{align*}\]

\[\text{OrganTransplant} \subseteq \text{Transplant}\]
\[\text{OrganTransplant} \subseteq \exists \text{site.\,Organ}\]
\[\exists \text{site.\,Organ} \subseteq \text{SO}\]
\[\text{Transplant} \cap \text{SO} \subseteq \text{OrganTransplant}\]
\[\text{HeartTransplant} \subseteq \text{Transplant}\]
\[\text{HeartTransplant} \subseteq \exists \text{site.\,Heart}\]
\[\exists \text{site.\,Heart} \subseteq \text{SH}\]
\[\text{Transplant} \cap \text{SH} \subseteq \text{HeartTransplant}\]
\[\text{Heart} \subseteq \text{Organ}\]

HeartTransplant \subseteq \text{SO}
SPARQL and OWL

… Now what about SPARQL1.1 and OWL?
SPARQL1.1 Entailment Regimes

SPARQL1.1 defines SPARQL query answering over RDFS and OWL2 ontologies (as well as RIF rule sets):
- [http://www.w3.org/TR/sparql11-entailment/](http://www.w3.org/TR/sparql11-entailment/)

Particularly:

- RDF Entailment Regime
- RDFS Entailment Regime
- D-Entailment Regime
- OWL 2 RDF-Based Semantics Entailment Regime
- OWL 2 Direct Semantics Entailment Regime

- Won’t go into details of those, but sketch the main ideas!
General Idea: Answer Queries with implicit answers

```xml
<http://dbpedia.org/resource/Tim_Berners-Lee>
  foaf:homepage
    <http://www.w3.org/People/Berners-Lee/> .

  foaf:name rdfs:subPropertyOf rdfs:label .
  foaf:homepage a owl:InverseFunctionalProperty .

<http://dblp.l3s.de/d2r/page/authors/Tim_Berners-Lee>
  foaf:homepage
    <http://www.w3.org/People/Berners-Lee/> ;
  foaf:name "Tim Berners-Lee".
```

```
SELECT ?P ?O
```

```
"Tim Berners-Lee"
```
OWL2 and SPARQL1.1

General Idea: Answer Queries with implicit answers
E.g. Graph/Ontology:

```
foaf:Person rdfs:subClassOf foaf:Agent .
foaf:Person rdfs:subClassOf
    [ a owl:Restriction ;
    owl:onProperty :hasFather ;
    owl:someValuesFrom foaf:Person. ]
foaf:knows rdfs:range foaf:Person.

:jeff a Person .
```

```
SELECT ?X { ?X a foaf:Person }
```

Pure SPARQL 1.0 returns only :Jeff,
should also return :aidan
Challenges+Pitfalls:

- Possibly Infinite answers (by RDFS ContainerMembership properties, OWL datatype reasoning, etc.)

- Conjunctive Queries: non-distinguished variables

- SPARQL 1.1 features: e.g. Aggregates
Pragmatic Solution within SPARQL1.1:

- Possibly Infinite answers (by RDFS ContainerMembership properties, OWL datatype reasoning, etc.)
  - Restrict answers to rdf:/rdfs:/owl:vocabulary minus rdf:_1 … rdf:_n plus terms occurring in the data graph

- Non-distinguished variables
  - No non-distinguished variables, answers must result from BGP matching, projection a post-processing step not part of SPARQL entailment regimes.

- SPARQL 1.1 other features: e.g. Aggregates, etc.
  - Again not affected, answers must result from BGP matching, projection a post-processing step not part of entailment.

- Simple, BUT: maybe not always entirely intuitive, so
  - Good to know what to expect … ;-)
Possibly Infinite answers RDF(S): Container Membership

Graph:
```
:rr2010Proceedings :hasEditors [ a rdf:Seq;
   rdf:_1 :pascal_hitzler.
   rdf:_2 :thomas_lukasiewicz.
 ]
```

Query with RDFS Entailment in mind:
```
SELECT ?CM { ?CM a rdfs:ContainerMembershipProperty}
```

Entailed by RDFS (axiomatic Triples):
```
rdfs:_1 a rdfs:ContainerMembershipProperty .
rdfs:_2 a rdfs:ContainerMembershipProperty .
rdfs:_3 a rdfs:ContainerMembershipProperty .
rdfs:_4 a rdfs:ContainerMembershipProperty .
...
```
Possibly Infinite answers RDF(S): Container Membership

Graph:
```
:rr2010Proceedings :hasEditors [ a rdf:Seq;
   rdf:_1 :pascal_hitzler.
   rdf:_2 :thomas_lukasiewicz.
]
```

Query with RDFS Entailment in mind:
```
SELECT ?CM { ?CM a rdfs:ContainerMembershipProperty}
```

SPARQL 1.1 restricts answers to `rdf:/rdfs:/owl:vocabulary` minus `rdf:_1` … `rdf:_n` plus terms occurring in the data graph

So, the only answers in SPARQL1.1 are:
```
{ ?CM/rdfs:_1, ?CM/rdfs:_2, }
```
Possibly Infinite answers OWL: datatype reasoning

Stupid way to say Peter is 50 in OWL:

```owl
ex:Peter a [ a owl:Restriction ;
               owl:onProperty ex:age ;
               owl:allValuesFrom [ rdf:type rdfs:Datatype .
                                 owl:oneOf ("50"^^xsd:integer) ] ]
```

Stupid query asking What is NOT Peters age:

```sparql
SELECT ?x WHERE {
    ex:Peter a [ a owl:Restriction ; owl:onProperty ex:age ;
               owl:allValuesFrom [ a rdfs:Datatype ;
                                 owl:datatypeComplementOf [ a
                                                 rdfs:Datatype ;
                                 owl:oneOf (?x) ] ] ]
}
```

**Theoretical answer:** all literal different from 50

**No danger in SPARQL 1.1 restricts answers to rdf:/rdfs:/owl:vocabulary minus rdf:_1 … rdf:_n plus terms occurring in the data graph**
Now What about Non-distinguished variables?

E.g. Graph

```turtle
foaf:Person rdfs:subClassOf foaf:Agent .
foaf:Person rdfs:subclassOf
    [ a owl:Restriction ;
    owl:onProperty :hasFather ;
    owl:someValuesFrom foaf:Person. ]
foaf:knows rdfs:range foaf:Person.
:jeff a Person
:jeff foaf:knows :aidan
```

```sparql
SELECT ?X ?Y { ?X :hasFather ?Y }
```

No answer, because no known value for ?Y in the data graph (here, ?Y is a distinguished variable, according to the previous definition)
Now What about Non-distinguished variables?

E.g. Graph

```sparql
foaf:Person rdfs:subClassOf foaf:Agent .
foaf:Person rdfs:subclassOf
    [ a owl:Restriction ;
    owl:onProperty :hasFather ;
    owl:someValuesFrom foaf:Person. ]
foaf:knows rdfs:range foaf:Person.
:jeff a Person
:jeff foaf:knows :aidan
```

SELECT ?X { ?X :hasFather ?Y }

But what about this one? ?Y looks like a “non-distinguished” variable

Solution: In SPARQL 1.1 answers must result from BGP matching, projection a post-processing step not part of entailment, i.e. SPARQL1.1 treats ALL variables as distinguished → so, still no answer.
Non-distinguished variables:

Simple Solution may seem not always intuitive, but:

- OWL Entailment in SPARQL based on BGP matching, i.e.
  - always only returns results with named individuals
  - Doesn’t affect SELECT: BGP matching takes place before projection
  - That is: **non-distinguished variables can’t occur “by design”**

- Conjunctive queries with non-distinguished variable still an open research problem for OWL:
  - Decidable for SHIQ, [B. Glimm et al. 2008]
  - Decidable for OWL 1 DL without transitive properties [B. Glimm, KR-10]
  - Particularly though: Decidability for the $SROIQ$ Description Logics still unknown…
Once again: SPARQL entailment defined only at the level of BGP matching

→ SPARQL1.1 Algebra is layered “on top”, no interaction

```
:person1 rdf:type [ owl:unionOf (:male :female) ]

SELECT ?X { {?X rdf:type :male } UNION {?X rdf:type :female } }
```

→ No result!
Similar as before… aggregates are evaluated as post-processing after BGP matching, so, no effect:

```
foaf:Person rdfs:subClassOf foaf:Agent .
foaf:Person rdfs:subClassOf
    [ a owl:Restriction ;
      owl:onProperty :hasFather ;
      owl:someValuesFrom foaf:Person. ]
:jeff a Person .
:jeff foaf:knows :aidan
foaf:knows rdfs:range foaf:Person.
```

SELECT ?X { ?X a foaf:Person }

Under RDFS/OWL entailment returns: {?X/jeff, ?X/aidan}
Similar as before… aggregates are evaluated as post-processing after BGP matching, so, no effect:

```
foaf:Person rdfs:subClassOf foaf:Agent .
foaf:Person rdfs:subclassOf
    [ a owl:Restriction ;
      owl:onProperty :hasFather ;
      owl:someValuesFrom foaf:Person. ]
:jeff a Person
:jeff foaf:knows :aidan
foaf:knows rdfs:range foaf:Person.
:jeff :hasFather [a Person].
:jeff owl:sameAs :aidan.
```

```
SELECT (Count(?X) AS ?Y) { ?X a foaf:Person }
```

Under RDFS/OWL entailment returns: `{?Y/3}`
Lessons learnt

OWL adds more expressivity on top of what can be said in RDF Schema about properties and Classes

OWL 2
1) adds more expressivity on top
   2) defines tractable fragments that are implementable efficiently

OWL+SPARQL gives implicit answers, but poses some challenges…

Will – by the end of the week – publish some last small assignment on:

Mini-assignment:
1. Write down statement (vii) from Unit3, slide 37 in Turtle Syntax.
2. Freestyle: Write your own ontology in OWL... Be creative! Your ontology should allow some useful inferences from your FOAF file.