Foundations of Knowledge Graphs – Part 1

Sebastian Rudolph, TU Dresden
Tutorial @ ICCS 2021
19.09.2021
Knowledge Graphs are everywhere...

Google

Enterprise Knowledge Graph
Next-gen Knowledge Management

Sentiment Symposium 2014

Understanding Who is saying or doing What

IBM
Why Knowledge Graphs?

- Initially, the Web was made for humans reading webpages.
- But there’s too much information out there to be entirely checked by a human with a specific information need.
- Machines can process large amounts of data.
- Normal Web data (such as HTML) is not suitable for content-sensitive machine processing (ambiguous, relies on background knowledge, etc.)
- Knowledge Graphs are concerned with representing information distributed across the Web in a machine-interpretable way.
Web-Wide Linked Open Data – The Vision Becoming True
Why Graphs? Why not, say, XML?

• Task: express ”The Book ’Foundations of Semantic Web Technologies' is published at CRC Press.”

• many options:

<pre>
&lt;published&gt;
 &lt;publisher&gt;CRC Press&lt;/publisher&gt;
 &lt;book&gt;Foundations of Semantic Web Technologies&lt;/book&gt;
&lt;/published&gt;

 &lt;publisher name="CRC Press">
 &lt;published book="Foundations of Semantic Web Technologies/&gt;
 &lt;/publisher&gt;

 &lt;book name="Foundations of Semantic Web Technologies">&
 &lt;published publisher="CRC Press”/&gt;
&lt;/book&gt;
</pre>

• ambiguity and tree structure inappropriate for intended purpose
RDF
RDF: Graphs instead of Trees

- Solution: representation by directed graphs

RDF

• “Resource Description Framework”
• W3C Recommendation (http://www.w3.org/RDF)
• RDF is a data model (not one specific syntax)
  • originally designed for providing metadata for Web resources, later used for more general purposes
  • encodes structured information
  • universal machine-readable exchange format
Building blocks for RDF Graphs

• URIs
• literals
• blank nodes (aka: empty nodes, bnodes)
RDF Triples

- constituents of an RDF triple

- terms inspired by linguistics but doesn’t always coincide

- eligible instantiations:
  - subject: URI or bnode
  - predicate: URI
  - object: URI or bnode or literal
Simple Semantics

- RDF is focused on information exchange and interoperability
- Answers of RDF tools to entailment queries should coincide
- Therefore, formal semantics needed
- Defined in a model-theoretic way, i.e. we start by defining interpretations
Simple Semantics - Interpretations

- **names**
- **literals**
- **URIs**

**resources**

**properties**

**vocabulary** $\mathcal{V}$

**interpretation** $\mathcal{I}$

$I_L$, $I_S$, $I_{EXT}$
Simple Semantics

• When is a triple valid in an interpretation?

• A graph is valid, if all its triples are

• This settles the case for "grounded" graphs

• Graph with blank nodes is valid if they can be mapped to elements such that the condition on the right is satisfied
Simple Entailment

• model theory defines simple entailment
• this is essentially graph matching with bnodes being wildcards

Example: the graph

simply entails the graph
RDF Schema
Schema Knowledge with RDF(S)

- RDF allows for specification of factual data
- = propositions about single resources (individuals) and their relationships
- desirable: propositions about generic groups of individuals, such as the class of publishers, of organizations, or of persons
- in database terminology: schema knowledge
- RDF Schema (RDFS): part of the RDF W3C recommendation
- rationale: stick to graph-shaped representation, i.e., schema knowledge to be represented using triples
Classes and Instances


• characterizes the specific book as an instance of the (self-defined) class of textbooks

• class-membership not exclusive:

• URIs can be “typed” as class-identifiers:
  ex:Textbook rdf:type rdfs:Class.
Subclasses

• we want to express that every textbook is a book, e.g., that every instance of the class ex:Textbook is “automatically” recognized as an instance of the class ex:Book

• realized by rdfs:subClassOf property:

  ex:Textbook  rdfs:subClassOf  ex:Book .

• rdfs:subClassOf is defined to be transitive and reflexive

• rule of thumb:

  rdf:type        means       ∈
  rdfs:subClassOf means       ⊆
Properties

• technical term for relations, correspondencies
• property names usually occur in predicate position in factoid RDF triples
• properties characterize, how two resources are related
• mathematically: set of pairs:
  married_with = {(Adam,Eve), (Brad,Angelina), ...}
• URI can be marked as property name by typing it accordingly:

  ex:publishedBy rdf:type rdf:Property .
Subproperties

• in analogy to subclass relationships
• representation in RDFS via `rdfs:subPropertyOf` e.g.:
  \[ \text{ex:happilyMarriedWith} \; \text{rdf:subPropertyOf} \; \text{rdf:marriedWith} \]

• then, given

  \[ \text{ex:Markus} \; \text{ex:happilyMarriedWith} \; \text{ex:Anja} \]

we can deduce

  \[ \text{ex:Markus} \; \text{ex:marriedWith} \; \text{ex:Anja} \]
Property Restrictions

• properties may give hints what types the linked resources have, e.g. we know that `ex:publishedBy` connects publications with publishers

• i.e., for all URIs $a$, $b$ where we know
  
  $a$ `ex:publishedBy` $b$ .
  
  we want to automatically follow:
  
  $a$ `rdf:type` `ex:Publication` .
  $b$ `rdf:type` `ex:Publisher` .

• this generic correspondence can be encoded in RDFS:
  
  `ex:publishedBy` `rdfs:domain` `ex:Publication` .
  `ex:publishedBy` `rdfs:range` `ex:Publisher` .
RDFS Entailment – Automation

- RDFS entailment can be decided via rule-like deduction calculus (NP-complete)
RDFS Semantics – Example

```

rdf:authorOf rdfs:subPropertyOf ex:creatorOf.

ex:creatorOf rdfs:domain ex:Artist.

ex:Artist rdfs:subClassOf ex:Person.
```

```


ex:shakespeare rdf:type ex:Person.
```
OWL
OWL – Overview

• Web Ontology Language
  • W3C Recommendation for the Semantic Web, 2009

• Semantic Web KR language based on description logics (DLs)
  • OWL DL is essentially the description logic $\text{SROIQ}(D)$
  • KR for web resources, using URIs.
  • Using web-enabled syntaxes, e.g. based on XML or RDF

• Purpose:
  • RDF(S) not expressive enough for expressing complex information
  • OWL provides more expressivity while still allowing for automated deduction
OWL by example

ex:Healthy rdfs:subClassOf [owl:complementOf ex:Dead] .

Healthy beings are not dead.

ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)] .

Every cat is alive or dead.

ex:owns rdfs:subPropertyOf ex:caresFor .

If somebody owns something, (s)he cares for it.

ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf ( [ rdf:type owl:Restriction ; owl:onProperty ex:owns ; owl:someValuesFrom ex:Cat], [ rdf:type owl:Restriction ; owl:onProperty ex:caresFor ; owl:allValuesFrom ex:Healthy] ) ] .

A happy cat owner owns a cat and all beings he cares for are healthy.

ex:schrödinger rdf:type ex:HappyCatOwner .

Schrödinger is a happy cat owner.
Behind the scenes...

```
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (  
    [ rdf:type owl:Restriction ; owl:onProperty ex:owns ; owl:someValuesFrom ex:Cat],  
    [ rdf:type owl:Restriction ; owl:onProperty ex:caresFor ; owl:allValuesFrom ex:Healthy]  
  ) ].
```
OWL Direct Semantics

• model theory (aka extensional semantics)
• OWL DL Interpretation:
Typical Inference Problems

Given a knowledge base KB, we might want to know:

- whether the knowledge in KB is consistent,
- whether KB entails a class membership
  (e.g. `ex:schrödinger rdf:type ex:Alive .),
- whether a class is (un)satisfiable
  (e.g. `[owl:intersectionOf ( ex:Dead , ex:Alive)]),
- whether KB entails a subclass statement
  (e.g. `ex:Alive rdfs:subClassOf ex:Healthy .),
- etc.
Reducing Inference Problems

• Many inference problems can be reduced to knowledge base consistency checking.

• Technique: claim the opposite and look what happens...

• **Class membership:**

  KB entails

  \[ \text{ex:schrödinger rdf:type ex:Alive} . \]

  iff adding

  \[ \text{ex:schrödinger rdf:type [owl:complementOf ex:Alive]} . \]

  to KB makes it inconsistent.
Reducing Inference Problems

• Many inference problems can be reduced to knowledge base consistency checking.

• Technique: claim the opposite and look what happens...

• **Class (un)satisfiability:**
  KB entails unsatisfiability of
  \[
  \text{[owl:intersectionOf} ( \text{ex:Dead} , \text{ex:Alive})]\]
  iff adding
  \[
  \text{ex:n rdf:type [owl:intersectionOf} ( \text{ex:Dead} , \text{ex:Alive})].
  \]
  to KB makes it inconsistent.
Reducing Inference Problems

• Many inference problems can be reduced to knowledge base consistency checking.
• Technique: claim the opposite and look what happens...

• **Subclass entailment:**
  
  KB entails
  
  \[
  \text{ex:Alive} \ rdfs:subClassOf \text{ex:Healthy}.
  \]

  iff adding
  
  \[
  \text{ex:n rdf:type [owl:intersectionOf (ex:Alive, [owl:complementOf ex:Healthy])].}
  \]

  to KB makes it inconsistent.
Reasoning in OWL

• But how to determine whether a KB is consistent?
• One option: translate to FOL and use standard methods.
• But: OWL is decidable while FOL isn’t.
• Still: FOL inferencing techniques (tableaux, resolution, type elimination) can be turned into decision procedures for OWL.
OWL Reasoning with Tableaux

• Tableaux methods are most frequent.
• Basic idea: try to build a model of the given KB. If this fails, the KB is inconsistent, otherwise consistent.
• Warning! The following example is simplified for better presentation (but demonstrates the essential features of tableaux-based methods). Consult the literature for a comprehensive treatment.
ex:Healthy rdfs:subClassOf [owl:complementOf ex:Dead].
ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)].
ex:owns rdfs:subPropertyOf ex:caresFor.
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (  
  [ rdf:type owl:Restriction; owl:onProperty ex:owns; owl:someValuesFrom ex:Cat],  
  [ rdf:type owl:Restriction; owl:onProperty ex:caresFor; owl:allValuesFrom ex:Healthy]) ].
ex:schrödinger rdf:type ex:HappyCatOwner.
ex:Healthy rdfs:subClassOf [owl:complementOf ex:Dead].
ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)].
ex:owns rdfs:subPropertyOf ex:caresFor.
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (  
  [ rdf:type owl:Restriction; owl:onProperty ex:owns; owl:someValuesFrom ex:Cat],
  [ rdf:type owl:Restriction; owl:onProperty ex:caresFor; owl:allValuesFrom ex:Healthy] )].
ex:schrödinger rdf:type ex:HappyCatOwner.

Knowledge Base

Tableau

ex:HappyCatOwner
ex:Healthy rdfs:subClassOf [owl:complementOf ex:Dead] .
ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (  
  [ rdf:type owl:Restriction ; owl:onProperty ex:owns ; owl:someValuesFrom ex:Cat],  
  [ rdf:type owl:Restriction ; owl:onProperty ex:caresFor ; owl:allValuesFrom ex:Healthy] )] .
ex:schrödinger rdf:type ex:HappyCatOwner .
ex:Healthy rdfs:subClassOf [owl:complementOf ex:Dead].
ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)].
ex:owns rdfs:subPropertyOf ex:caresFor.
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:schrödinger rdf:type ex:HappyCatOwner).
ex:Healthy rdfs:subClassOf [owl:complementOf ex:Dead].
ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)].
ex:owns rdfs:subPropertyOf ex:caresFor.
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (
  [ rdf:type owl:Restriction; owl:onProperty ex:owns; owl:someValuesFrom ex:Cat],
  [ rdf:type owl:Restriction; owl:onProperty ex:caresFor; owl:allValuesFrom ex:Healthy])].
ex:schrödinger rdf:type ex:HappyCatOwner.

Knowledge Base

Tableau
ex:Healthy rdfs:subClassOf [owl:complementOf ex:Dead].
ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)].
ex:owns rdfs:subPropertyOf ex:caresFor.
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (  
  [ rdf:type owl:Restriction ; owl:onProperty ex:owns ; owl:someValuesFrom ex:Cat],  
  [ rdf:type owl:Restriction ; owl:onProperty ex:caresFor ; owl:allValuesFrom ex:Healthy]) ].
ex:schrödinger rdf:type ex:HappyCatOwner.

Knowledge Base

Tableau
ex:Healthy rdfs:subClassOf [owl:complementOf ex:Dead].
ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)].
ex:owns rdfs:subPropertyOf ex:caresFor.
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf ( [rdf:type owl:Restriction; owl:onProperty ex:owns; owl:someValuesFrom ex:Cat], [rdf:type owl:Restriction; owl:onProperty ex:caresFor; owl:allValuesFrom ex:Healthy])].
ex:schrödinger rdf:type ex:HappyCatOwner.
ex:Healthy rdfs:subClassOf [owl:complementOf ex:Dead] .
ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)]
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (  
[ rdf:type owl:Restriction ; owl:onProperty ex:owns ; owl:someValuesFrom ex:Cat],  
[ rdf:type owl:Restriction ; owl:onProperty ex:caresFor ; owl:allValuesFrom ex:Healthy]) ] .
ex:schrödinger rdf:type ex:HappyCatOwner .

Knowledge Base

Tableau
ex:Healthy rdfs:subClassOf [owl:complementOf ex:Dead].
ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)].
ex:owns rdfs:subPropertyOf ex:caresFor.
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf ( [ rdf:type owl:Restriction; owl:onProperty ex:owns; owl:someValuesFrom ex:Cat], [ rdf:type owl:Restriction; owl:onProperty ex:caresFor; owl:allValuesFrom ex:Healthy])].
ex:schrödinger rdf:type ex:HappyCatOwner.

Knowledge Base

Tableau
ex:Healthy rdfs:subClassOf [owl:complementOf ex:Dead] .
ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf ( RDF_Association, ex:Cat), RDF_Association, ex:Healthy)] .
ex:schrödinger rdf:type ex:HappyCatOwner .

Knowledge Base

Tableau
ex:Healthy rdfs:subClassOf [owl:complementOf ex:Dead] .
ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:schrödinger rdf:type ex:HappyCatOwner .

Knowledge Base

ex:owns
ex:caresFor

ex:HappyCatOwner
[owl:intersectionOf (ex:Cat, ex:Healthy)]
ex:Cat
ex:Healthy

Tableau

ex:owns
ex:caresFor

ex:HappyCatOwner
[owl:intersectionOf (ex:Cat, ex:Healthy)]
ex:Cat
ex:Healthy

ex:owns
ex:caresFor

ex:HappyCatOwner
[owl:intersectionOf (ex:Cat, ex:Dead)]
ex:Cat
ex:Dead

ex:owns
ex:caresFor

ex:HappyCatOwner
[owl:intersectionOf (ex:Cat, ex:Alive)]
ex:Cat
ex:Alive

ex:owns
ex:caresFor

ex:HappyCatOwner
[owl:intersectionOf (ex:Cat, ex:Dead)]
ex:Cat
ex:Dead

ex:owns
ex:caresFor

ex:HappyCatOwner
[owl:intersectionOf (ex:Cat, ex:Alive)]
ex:Cat
ex:Alive

45
ex:Healthy rdfs:subClassOf [owl:complementOf ex:Dead] .
ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf ( ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf ( ex:Dead, ex:Alive)] .
ex:Healthy rdfs:subClassOf [owl:complementOf ex:Dead] .
ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:schrödinger rdf:type ex:HappyCatOwner .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor.
ex:Healthy rdfs:subClassOf [owl:complementOf ex:Dead] .
ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (rdfs:subClassOf ex:Cat, rdfs:subClassOf ex:Alive, owl:someValuesFrom ex:Healthy, owl:allValuesFrom ex:Healthy)] .
ex:schrödinger rdf:type ex:HappyCatOwner .

Knowledge Base

Tableau

ex:owns ex:caresFor

ex:HappyCatOwner
[owl:intersectionOf (ex:Dead, ex:Alive)]

ex:Cat ex:Healthy
[owl:unionOf (ex:Dead, ex:Alive)]

ex:owns

ex:HappyCatOwner
[owl:intersectionOf (ex:Dead, ex:Alive), ex:complementOf ex:Dead]

ex:Cat ex:Healthy ex:Alive
[owl:intersectionOf (ex:Dead, ex:Alive), owl:complementOf ex:Dead]
Query Languages for (RDF) Knowledge Graphs?

• How to access information that was specified in RDF or OWL?

• Querying information in RDF(S): Simple/RDF/RDFS entailment
  – “Can a certain RDF graph be derived from the given data?”

• Querying information in OWL: Logical entailment
  – “Can a subclass relation be derived from the ontology?”
  – “What are the instances of a given OWL class?”
Are OWL and RDF entailment enough?

• Even OWL is too weak for many queries:
• “Who lives together with their parents?”
  (logical expressivity)
• “Who has married parents?”
  (logical expressivity)
• “Which properties connect two given individuals?”
  (schema-level query)
• “Which strings in the ontology are in French language?”
  (datatype expressivity)
SPARQL
Queries for RDF: SPARQL

- SPARQL [sparkle]:
- **SPARQL Protocol And RDF Query Language**
- Query language for data from RDF documents
Basic Queries

• A simple example query:

```prefix
PREFIX ex: <http://example.org/>
SELECT ?title ?author
WHERE
  ?book ex:author ?author . }
```

• Main part is a **query pattern** (WHERE)
  – Patterns use RDF Turtle syntax
  – Variables can be used, even in predicate positions (?variable)

• **Abbreviations** for URIs (PREFIX)

• Query result based on **selected variables** (SELECT)
Query Results

• A simple example document:

```latex
@prefix ex: <http://example.org/> .
ex:SemanticWeb
  ex:publishedBy <http://crc-press.com/uri> ;
ex:title "Foundations of Semantic Web Technologies" ;
```

• Query results are **tables**, each row is one query result:

<table>
<thead>
<tr>
<th>title</th>
<th>author</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Foundations of ...”</td>
<td><a href="http://example.org/Hitzler">http://example.org/Hitzler</a></td>
</tr>
<tr>
<td>“Foundations of ...”</td>
<td><a href="http://example.org/Kr%C3%B6tzsch">http://example.org/Krötzsch</a></td>
</tr>
<tr>
<td>“Foundations of ...”</td>
<td><a href="http://example.org/Rudolph">http://example.org/Rudolph</a></td>
</tr>
</tbody>
</table>
Grouping Query Patterns

• Simple graph patterns are grouped with `{   }`

• Example:

```{ 

{} 
}
```

→ Useful with additional query features
Optional Patterns

• Optional parts can be specified with `OPTIONAL`

• Example:

```reason
  OPTIONAL { ?book ex:author ?author . }
}
```

→ Parts of the result can be `unbound`:

<table>
<thead>
<tr>
<th>book</th>
<th>title</th>
<th>author</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://example.org/book1">http://example.org/book1</a></td>
<td>“title 1”</td>
<td><a href="http://example.org/johndoe">http://example.org/johndoe</a></td>
</tr>
<tr>
<td><a href="http://example.org/book2">http://example.org/book2</a></td>
<td>“title 2”</td>
<td></td>
</tr>
<tr>
<td><a href="http://example.org/book3">http://example.org/book3</a></td>
<td>“title 3”</td>
<td>_:a</td>
</tr>
<tr>
<td><a href="http://example.org/book4">http://example.org/book4</a></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Alternative Patterns

Alternatives can be specified with \textbf{UNION}

Example:

\[
\{ \text{?book ex:title \ ?title . \}} \text{ \textbf{UNION}} \\
\{ \text{?book ex:author \ ?author . \}}
\}
\]

\rightarrow \text{Result = union of results for one of the alternatives}
\rightarrow \text{Parts of the result can be \textit{unbound}}

Note: no interaction between multiple variable occurrences in alternative query parts
Filters

Additional “filter conditions” can be specified with **FILTER**

Example:

```
  FILTER( (?price < 17) && !isBlank(?book) )
}
```

→ Filter condition: “price a number below 17 and book not a blank node”
→ Results that do not match the filter are removed

**SPARQL provides many filter functions:**

- Comparisons (=, <, >, <=, >=, !=),
- arithmetics (+, −, *, /),
- Booleans (&&, ||, !),
- RDF-specific functions (isLiteral(), Lang(), BOUND(), ...)

SPARQL: Summary / More Features

- Based on matching simple graph patterns
- Grouping, optionals, and alternatives
- Filters: “extra-logical” result restrictions

Further features:

- Modifiers: postprocess query result set
  E.g.: ORDER BY ?age LIMIT 10 OFFSET 5
  (→ order by ?age and return 10 results, starting at result 5)

- Result formats: choose encoding of results
  E.g.: SELECT ?name, ?age (→ as in earlier examples)
  CONSTRUCT {?name ex:hasAge ?age .}
  (→ construct RDF graph as result)
...end of Part I.

Questions?