TECHNISCHE
UNIVERSITÄT
DRESDEN

Computational
Logic ∴ Group

Foundations of Knowledge Graphs – Part 1

Sebastian Rudolph, TU Dresden

Tutorial @ ICCS 2021
19.09.2021

Knowledge Graphs are everywhere...

Wikidata: the new Rosetta Stone
"The promise of linked data seems to have finally arrived."

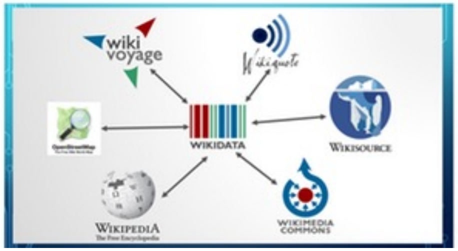
WIKIDATA

Free knowledge base *that anyone can edit*

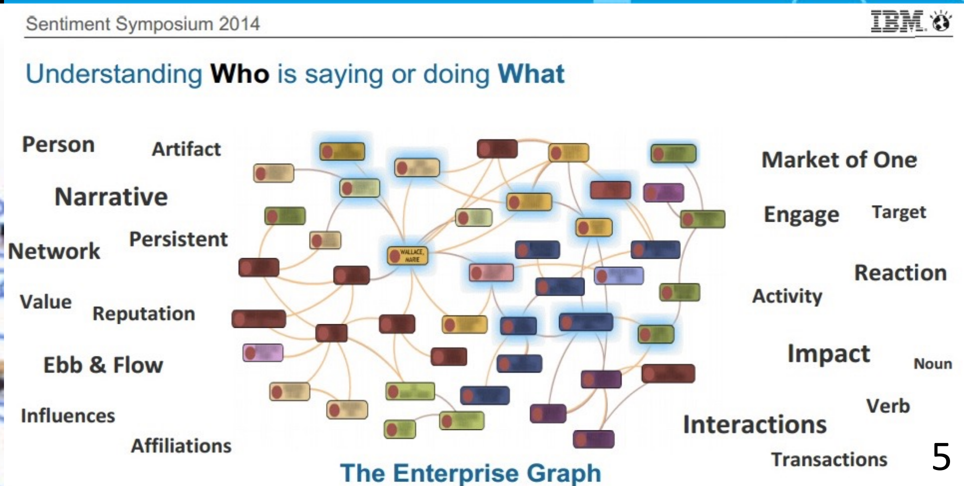
Launched in 2012

Integrated with Wikipedia and other sister projects

Statistics (February 2017)
Over 25M items
Over 130M statements



Enterprise Knowledge Graph
Next-gen Knowledge Management



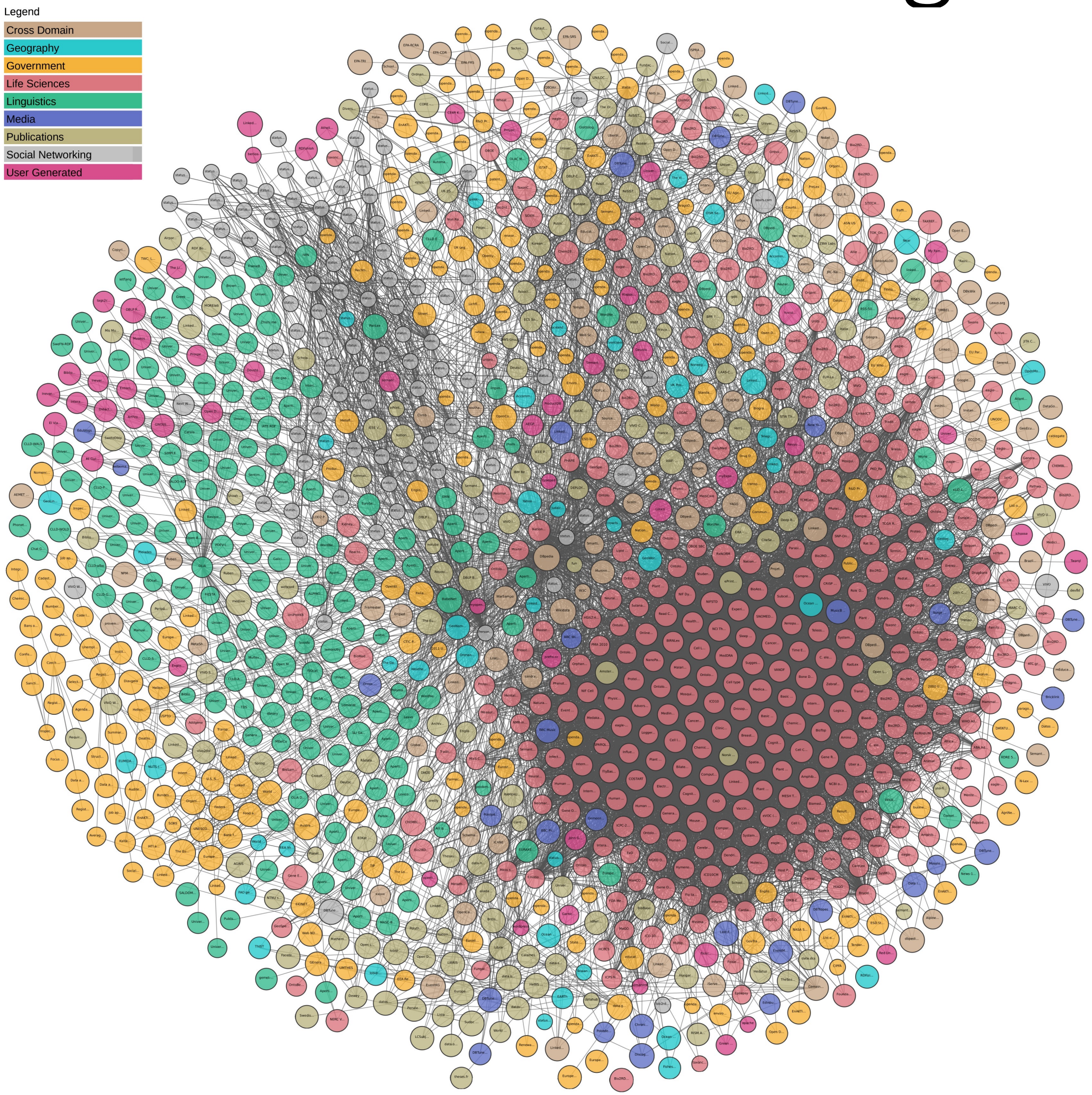
Sentiment Symposium 2014

Understanding **Who** is saying or doing **What**

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:

```
<published>
<publisher>CRC Press</publisher>
<book>Foundations of Semantic Web Technologies</book>
</published>
```

```
                    <publisher name="CRC Press">
                    <published book=”Foundations of Semantic Web Technologies/>
                    </publisher>
```

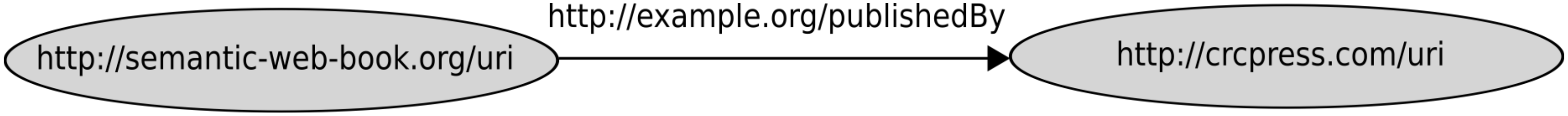
```
<book name="Foudations of Semantic Web Technologies">
<published publisher="CRC Press”/>
</book>
```

- ambiguity and tree structure inappropriate for intended purpose

RDF

RDF: Graphs instead of Trees

- Solution: representation by directed graphs

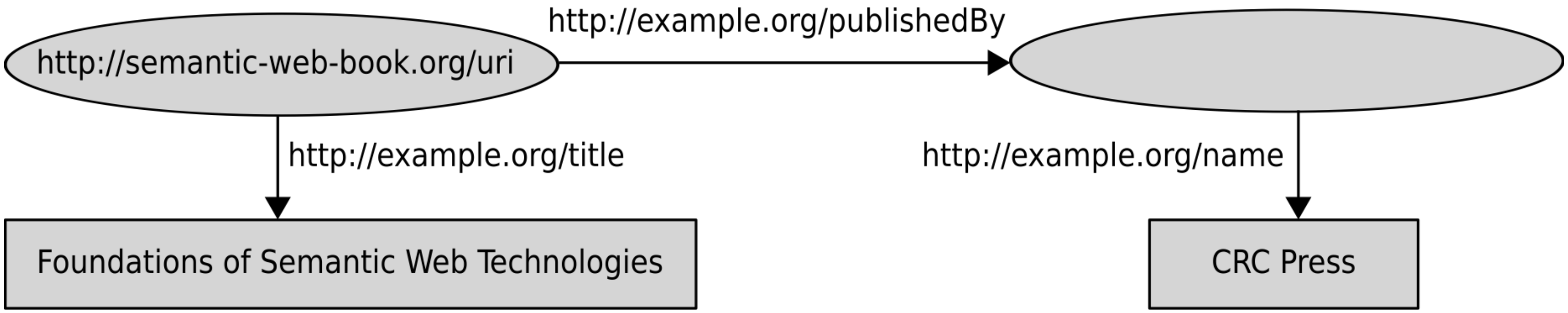

http://semantic-web-book.org/uri
http://example.org/publishedBy
http://crcpress.com/uri

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 informationen
  - universal machine-readable exchange format

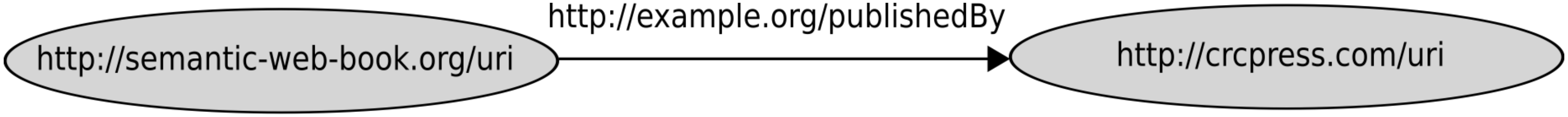
Building blocks for RDF Graphs

- URIs
- literals
- blank nodes (aka: empty nodes, bnodes)


http://semantic-web-book.org/uri
http://example.org/publishedBy
http://example.org/title
http://example.org/name
Foundations of Semantic Web Technologies
CRC Press

RDF Triples

- constitutents of an RDF triple


http://semantic-web-book.org/uri
http://example.org/publishedBy
http://crcpress.com/uri

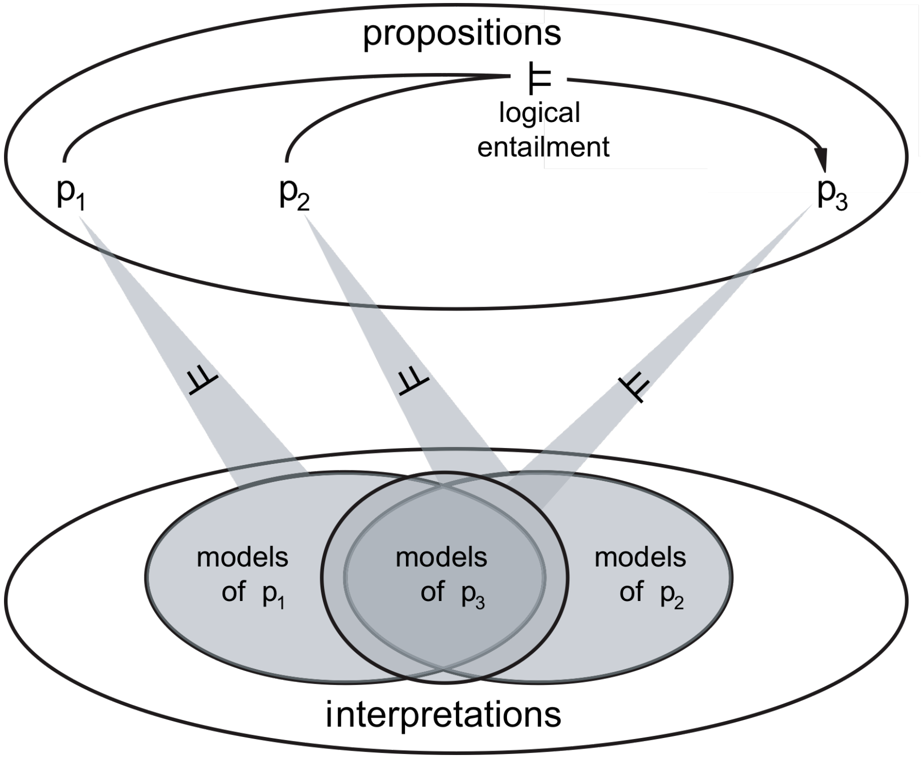
subject *predicate* *object*

- terms inspired by linguistics but doesn’t always coincide

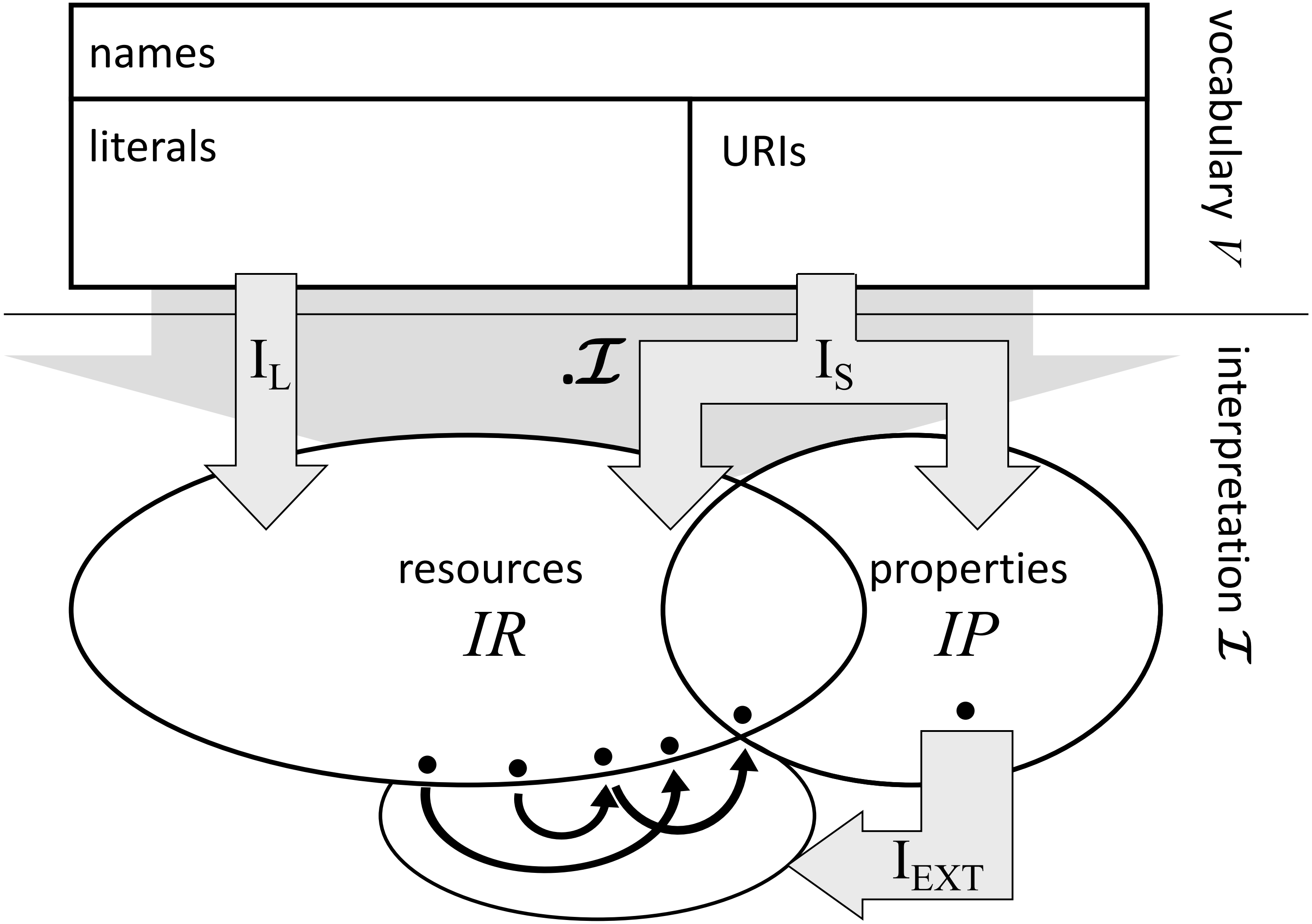
- eligible instantiations:
  subject : URI or bnode
  predicate : URI
  object : URI or bnode or literal

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



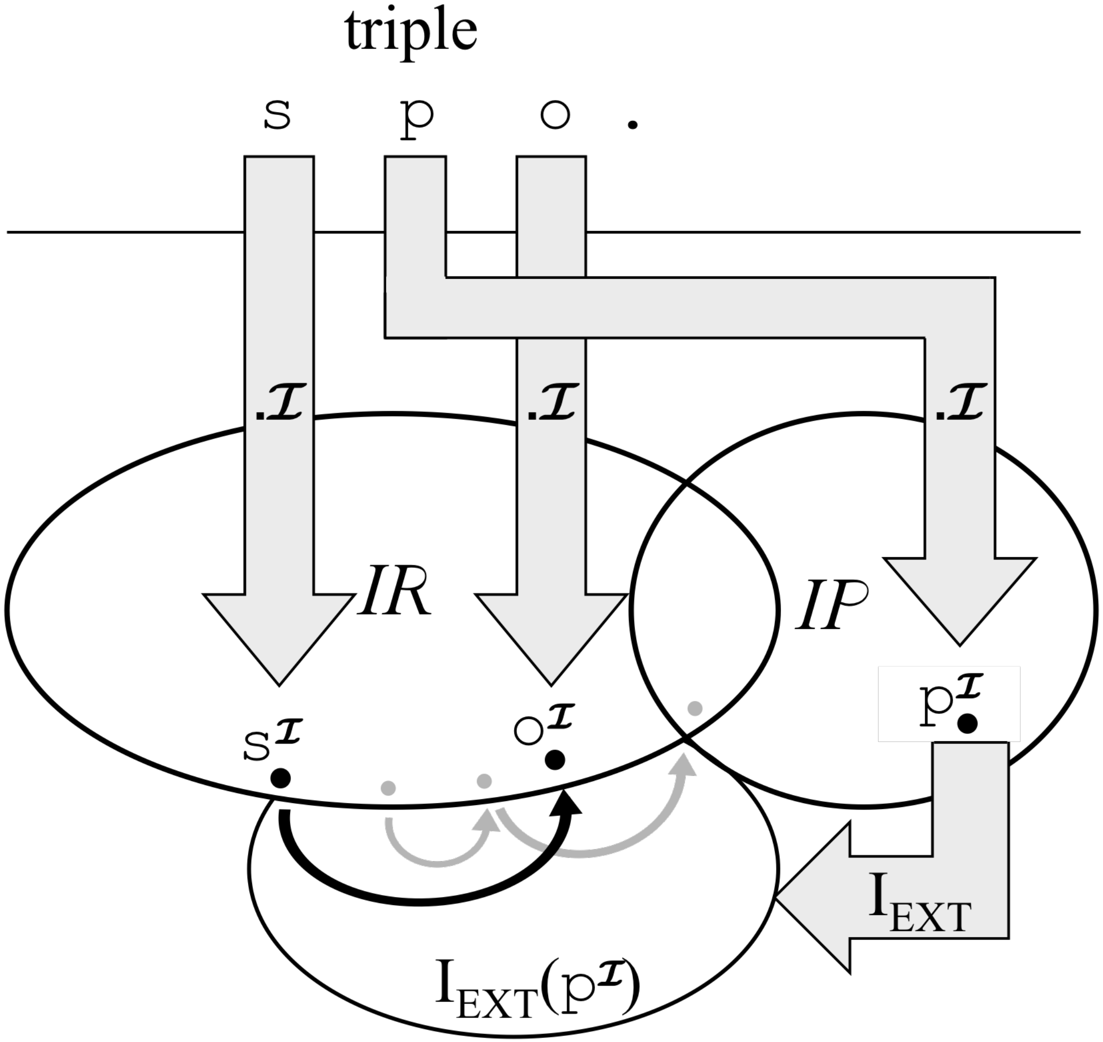
Simple Semantics - Interpretations



Simple Semantics

- when is a triple valid in an interpretation?

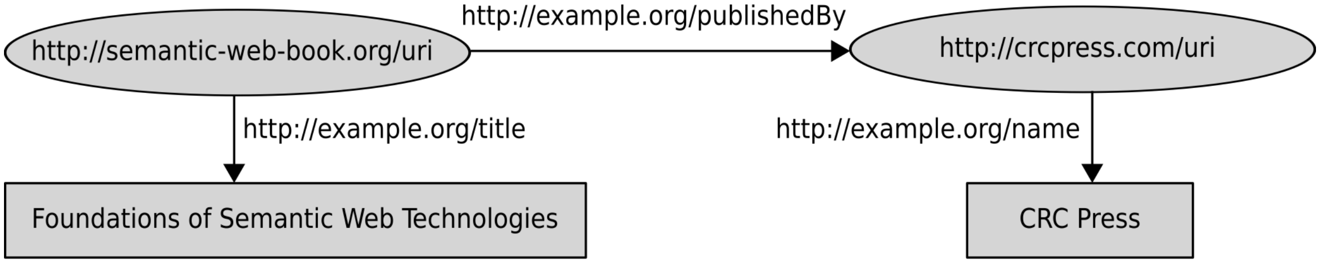
- a graph is valid, if all its triples are
- this settles the case for „grounded“ graphs
- graph with blank nodes is valid if they can be mapped to elements such that the condition on the right is satisfied



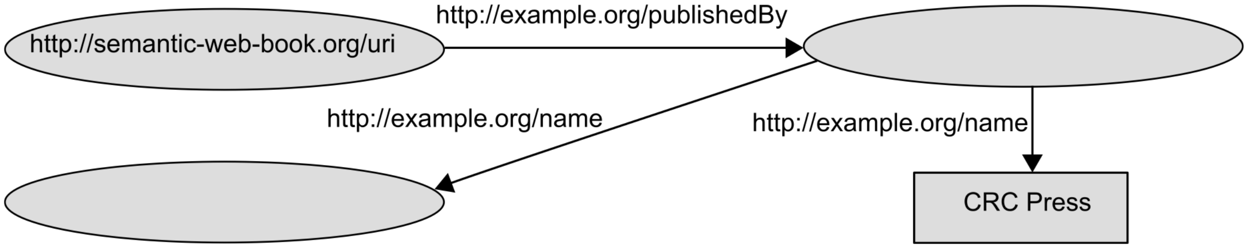
Simple Entailment

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

Example: the graph


http://semantic-web-book.org/uri
http://example.org/publishedBy
http://crcpress.com/uri
http://example.org/title
http://example.org/name
Foundations of Semantic Web Technologies
CRC Press

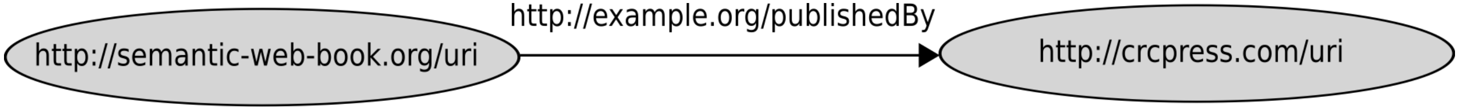
simply entails the graph


http://semantic-web-book.org/uri
http://example.org/publishedBy
http://example.org/name
http://example.org/name
CRC Press

RDF Schema

Schema Knowledge with RDF(S)

- RDF allows for specification of factual data


http://semantic-web-book.org/uri
http://example.org/publishedBy
http://crcpress.com/uri

- = 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

```
book:uri  rdf:type  ex:Textbook .
```

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

- class-membership not exclusive:

  ```
  book:uri  rdf:type  ex:Enjoyable .
  ```

- 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	$\in$
rdfs:subClassOf	means	$\subseteq$

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.:

```
ex:happilyMarriedWith   rdf:subPropertyOf   rdf:marriedWith   .
```

- then, given

```
ex:Markus  ex:happilyMarriedWith   ex:Anja  .
```

we can deduce

```
ex:Markus  ex:marriedWith   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:

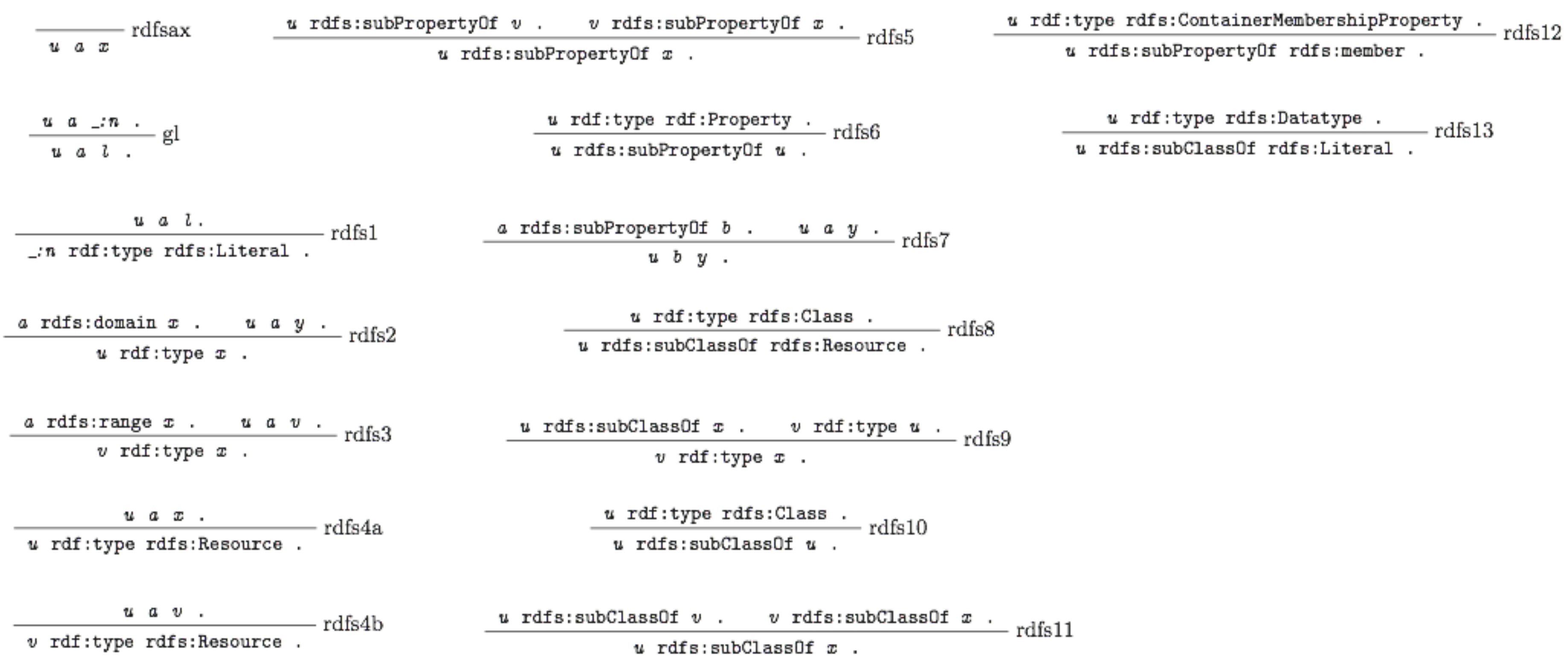
  ```
  a  rdf:type  ex:Publication  .
  b  rdf:type  ex:Publisher .
  ```

- this generic correspondency 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)



$$\frac{}{u\ a\ x}\ \text{rdfsax}$$

$$\frac{u\ a\ \texttt{\_:}n\ .}{u\ a\ l\ .}\ \text{gl}$$

$$\frac{u\ a\ l\,.}{\texttt{\_:}n\ \texttt{rdf:type}\ \texttt{rdfs:Literal}\ .}\ \text{rdfs1}$$

$$\frac{a\ \texttt{rdfs:domain}\ x\ . \qquad u\ a\ y\ .}{u\ \texttt{rdf:type}\ x\ .}\ \text{rdfs2}$$

$$\frac{a\ \texttt{rdfs:range}\ x\ . \qquad u\ a\ v\ .}{v\ \texttt{rdf:type}\ x\ .}\ \text{rdfs3}$$

$$\frac{u\ a\ x\ .}{u\ \texttt{rdf:type}\ \texttt{rdfs:Resource}\ .}\ \text{rdfs4a}$$

$$\frac{u\ a\ v\ .}{v\ \texttt{rdf:type}\ \texttt{rdfs:Resource}\ .}\ \text{rdfs4b}$$

$$\frac{u\ \texttt{rdfs:subPropertyOf}\ v\ . \qquad v\ \texttt{rdfs:subPropertyOf}\ x\ .}{u\ \texttt{rdfs:subPropertyOf}\ x\ .}\ \text{rdfs5}$$

$$\frac{u\ \texttt{rdf:type}\ \texttt{rdf:Property}\ .}{u\ \texttt{rdfs:subPropertyOf}\ u\ .}\ \text{rdfs6}$$

$$\frac{a\ \texttt{rdfs:subPropertyOf}\ b\ . \qquad u\ a\ y\ .}{u\ b\ y\ .}\ \text{rdfs7}$$

$$\frac{u\ \texttt{rdf:type}\ \texttt{rdfs:Class}\ .}{u\ \texttt{rdfs:subClassOf}\ \texttt{rdfs:Resource}\ .}\ \text{rdfs8}$$

$$\frac{u\ \texttt{rdfs:subClassOf}\ x\ . \qquad v\ \texttt{rdf:type}\ u\ .}{v\ \texttt{rdf:type}\ x\ .}\ \text{rdfs9}$$

$$\frac{u\ \texttt{rdf:type}\ \texttt{rdfs:Class}\ .}{u\ \texttt{rdfs:subClassOf}\ u\ .}\ \text{rdfs10}$$

$$\frac{u\ \texttt{rdfs:subClassOf}\ v\ . \qquad v\ \texttt{rdfs:subClassOf}\ x\ .}{u\ \texttt{rdfs:subClassOf}\ x\ .}\ \text{rdfs11}$$

$$\frac{u\ \texttt{rdf:type}\ \texttt{rdfs:ContainerMembershipProperty}\ .}{u\ \texttt{rdfs:subPropertyOf}\ \texttt{rdfs:member}\ .}\ \text{rdfs12}$$

$$\frac{u\ \texttt{rdf:type}\ \texttt{rdfs:Datatype}\ .}{u\ \texttt{rdfs:subClassOf}\ \texttt{rdfs:Literal}\ .}\ \text{rdfs13}$$

RDFS Semantics – Example

```
ex:shakespeare   ex:authorOf          ex:hamlet .
rdf:authorOf     rdfs:subPropertyOf   ex:creatorOf .
ex:creatorOf     rdfs:domain          ex:Artist .
ex:Artist        rdfs:subClassOf      ex:Person .
```

$$\begin{array}{l}
\underbrace{\texttt{ex:shakespeare ex:authorOf ex:hamlet. ex:authorOf rdfs:subPropertyOf ex:creatorOf.}}\\
\underbrace{\texttt{ex:shakespeare ex:creatorOf ex:hamlet. ex:creatorOf rdfs:domain ex:Artist.}}\\
\underbrace{\texttt{ex:shakespeare rdf:type ex:Artist.   ex:Artist rdfs:subClassOf ex:Person.}}\\
\texttt{ex:shakespeare rdf:type ex:Person.}
\end{array}$$

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 $\mathcal{SROIQ}$(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]
   ) ] .
```

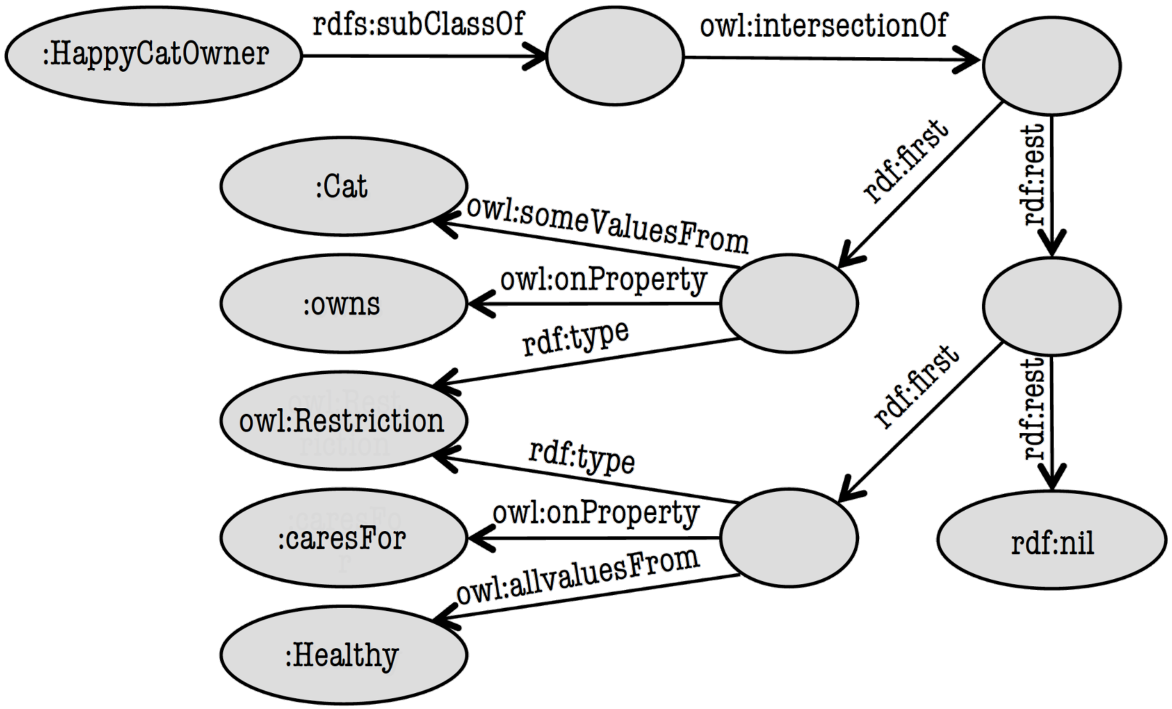
A happy cat owner owns a cat and all beings he cares for are healthy.

```
ex:schrödinger rdf:type  ex:HappyCatOwner .
```

Schrödinger is a happy cat owner.

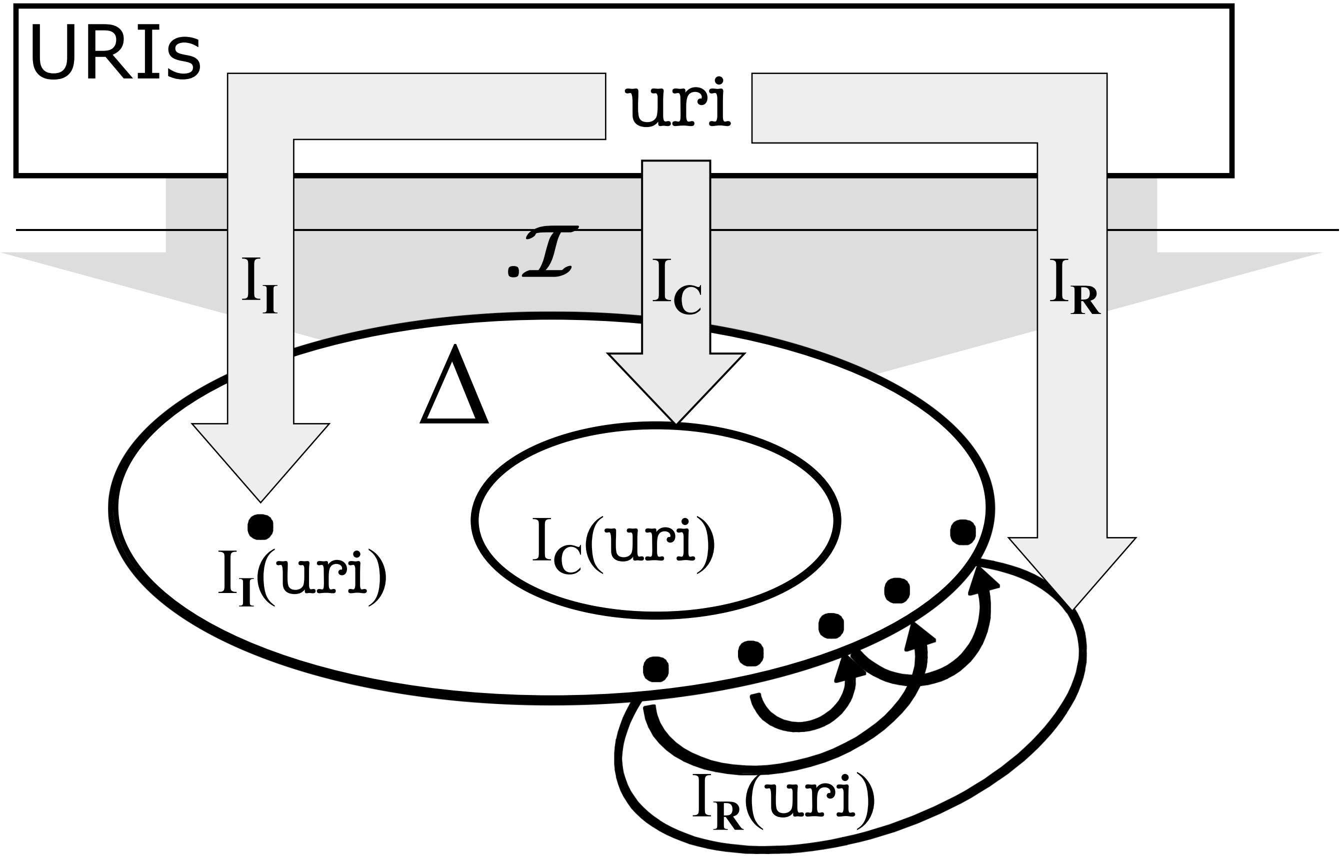
Behind the scenes…

```
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

  ```
  ex:schrödinger rdf:type ex:Alive .
  ```

  iff adding

  ```
  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

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

  iff adding

  ```
  ex:n rdf:type [owl:intersectionOf ( ex:Dead , 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

  ```
          ex:Alive rdfs:subClassOf ex:Healthy .
  ```

  iff adding

  ```
   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.

OWL Reasoning with Tableaux

Knowledge Base

```
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 .
```

Tableau

OWL Reasoning with Tableaux

Knowledge Base

```
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 .
```

Tableau

ex:HappyCatOwner

OWL Reasoning with Tableaux

Knowledge Base

```
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 .
```

Tableau

ex:HappyCatOwner

[owl:intersectionOf (🟩, 🟦)]

OWL Reasoning with Tableaux

Knowledge Base

```
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 .
```

Tableau

ex:HappyCatOwner

[owl:intersectionOf (🟩, 🟦)]

🟩 🟦

OWL Reasoning with Tableaux

Knowledge Base

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

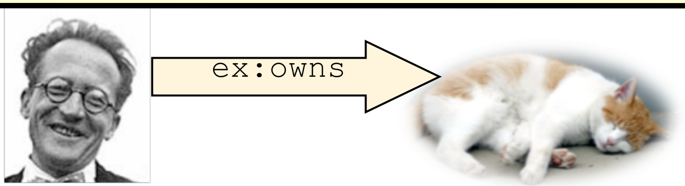
ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)] .

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 .

Tableau



ex:HappyCatOwner

[owl:intersectionOf (🟩, 🟦)]

🟩 🟦

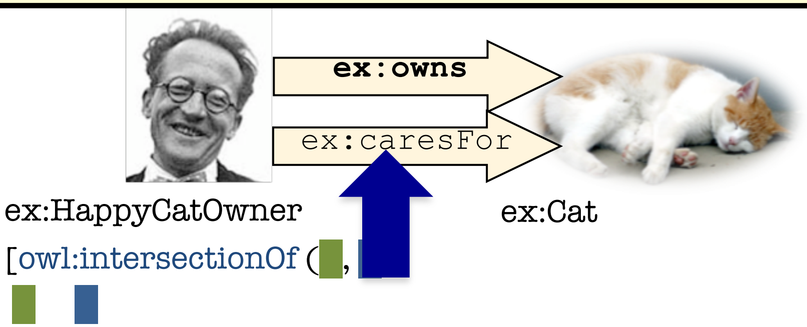
ex:Cat

OWL Reasoning with Tableaux

Knowledge Base

```
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 .
```

Tableau



ex:HappyCatOwner

[owl:intersectionOf (🟩, 🟦

🟩 🟦

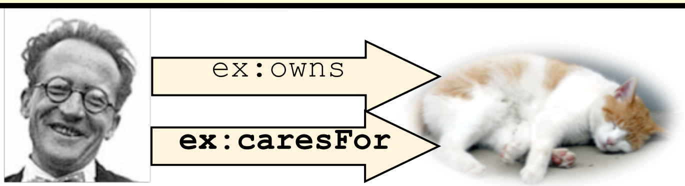
ex:Cat

OWL Reasoning with Tableaux

Knowledge Base

```
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 .
```

Tableau



ex:HappyCatOwner

[owl:intersectionOf (🟩, 🟦)]

🟩 🟦

ex:Cat ex:Healthy

OWL Reasoning with Tableaux

Knowledge Base

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

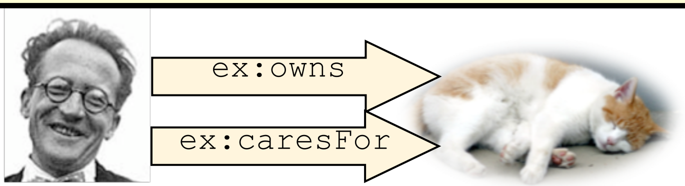
ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)]

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 .

Tableau



ex:HappyCatOwner

[owl:intersectionOf (🟩, 🟦)]

🟩 🟦

ex:Cat ex:Healthy

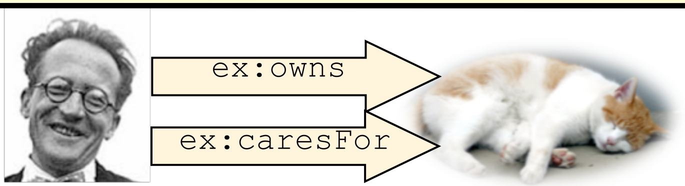
[owl:unionOf
(ex:Dead, ex:Alive)]

OWL Reasoning with Tableaux

Knowledge Base

```
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 .
```

Tableau



ex:HappyCatOwner

[owl:intersectionOf (🟩, 🟦)]

🟩 🟦

ex:Cat ex:Healthy

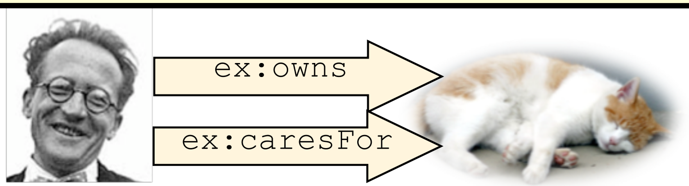
[owl:unionOf
(ex:Dead, ex:Alive)]

OWL Reasoning with Tableaux

Knowledge Base

```
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 .
```

Tableau



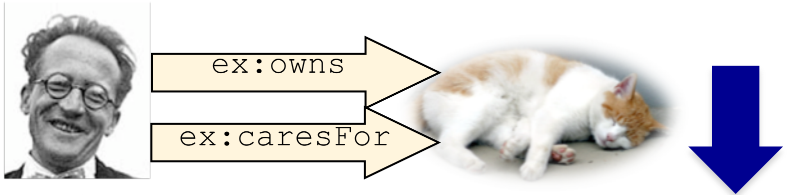
ex:HappyCatOwner

[owl:intersectionOf (🟩, 🟦)]

🟩 🟦

ex:Cat ex:Healthy

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



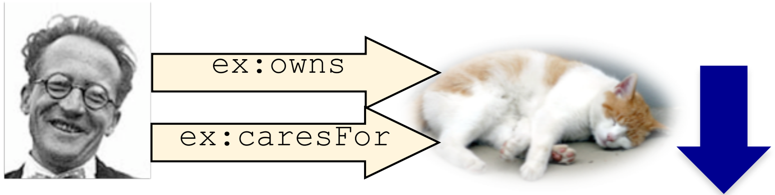
ex:HappyCatOwner

[owl:intersectionOf (🟩, 🟦)]

🟩 🟦

ex:Cat ex:Healthy ex:Dead

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



ex:HappyCatOwner

[owl:intersectionOf (🟩, 🟦)]

🟩 🟦

ex:Cat ex:Healthy ex:Alive

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

OWL Reasoning with Tableaux

Knowledge Base

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

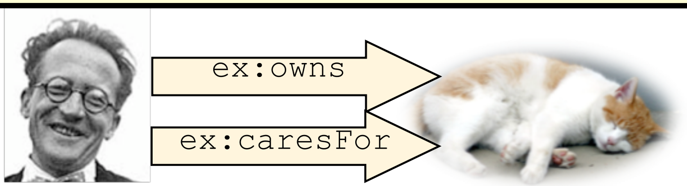
ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)] .

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 .

Tableau



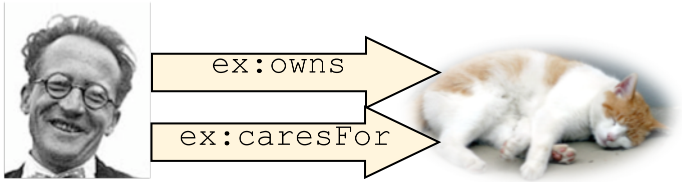
ex:HappyCatOwner

[owl:intersectionOf (🟩, 🟦)]

🟩 🟦

ex:Cat ex:Healthy

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



ex:HappyCatOwner

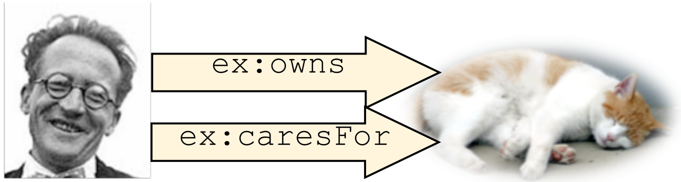
[owl:intersectionOf (🟩, 🟦)]

🟩 🟦

ex:Cat **ex:Healthy** ex:Dead

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

[owl:complementOf
ex:Dead]



ex:HappyCatOwner

[owl:intersectionOf (🟩, 🟦)]

🟩 🟦

ex:Cat ex:Healthy ex:Alive

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

OWL Reasoning with Tableaux

Knowledge Base

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

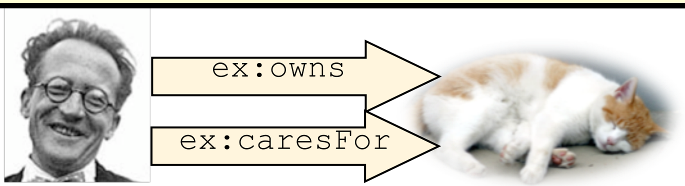
ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)] .

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])] .

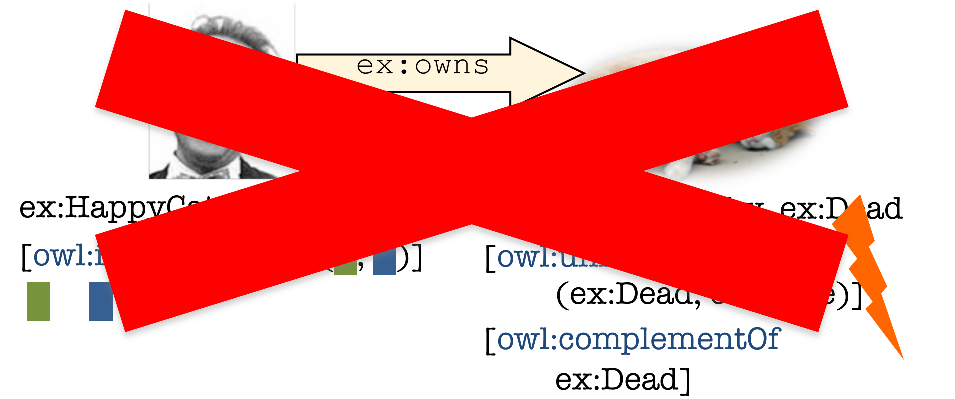
ex:schrödinger rdf:type ex:HappyCatOwner .

Tableau



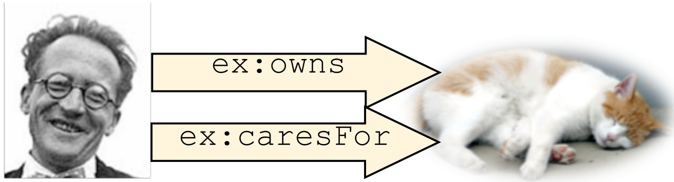
ex:HappyCatOwner
[owl:intersectionOf (🟩, 🟦)]
🟩 🟦

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



~~ex:HappyC~~
~~[owl:i~~ ~~, 🟦)]~~
🟩 🟦

~~y ex:D ad~~
~~[owl:u~~
~~(ex:Dead,~~ ~~e)]~~
[owl:complementOf
ex:Dead]



ex:HappyCatOwner
[owl:intersectionOf (🟩, 🟦)]
🟩 🟦

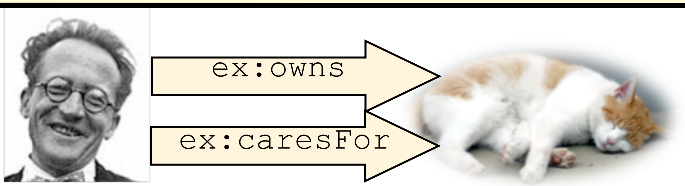
ex:Cat ex:Healthy ex:Alive
[owl:unionOf
(ex:Dead, ex:Alive)]

OWL Reasoning with Tableaux

Knowledge Base

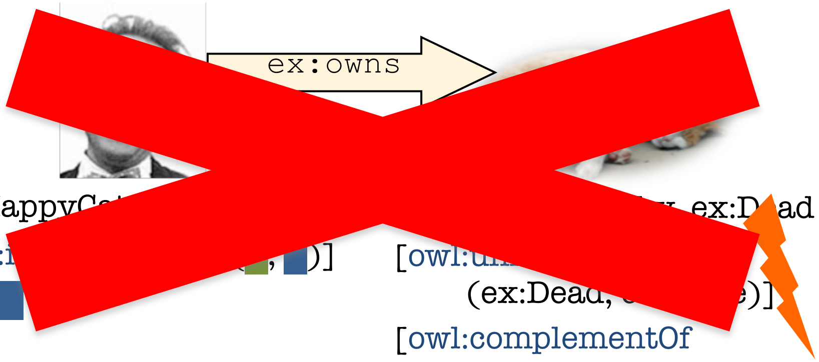
ex:Healthy rdfs:subClassOf [owl:complementOf ex:Dead]
ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)] .
ex:owns rdfs:subPropertyOf ex:caresFor .
ex:HappyCatOwner rdfs:subClassOf [owl:intersectionOf (
[rdf:type owl:Restriction ; owl:onProperty ex:owns ; owl:someValuesFrom ex:Cat],
[rdf:type owl:Restriction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

Tableau



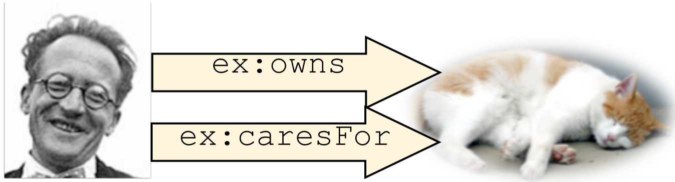
ex:HappyCatOwner
[owl:intersectionOf (🟩, 🟦)]
🟩 🟦

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



ex:HappyC
[owl:i , 🟦)]
🟩 🟦

ex:D ad
[owl:u
(ex:Dead, e)]
[owl:complementOf
ex:Dead]



ex:HappyCatOwner
[owl:intersectionOf (🟩, 🟦)]
🟩 🟦

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

OWL Reasoning with Tableaux

Knowledge Base

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

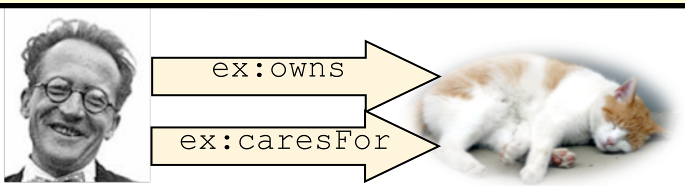
ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)] .

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])] .

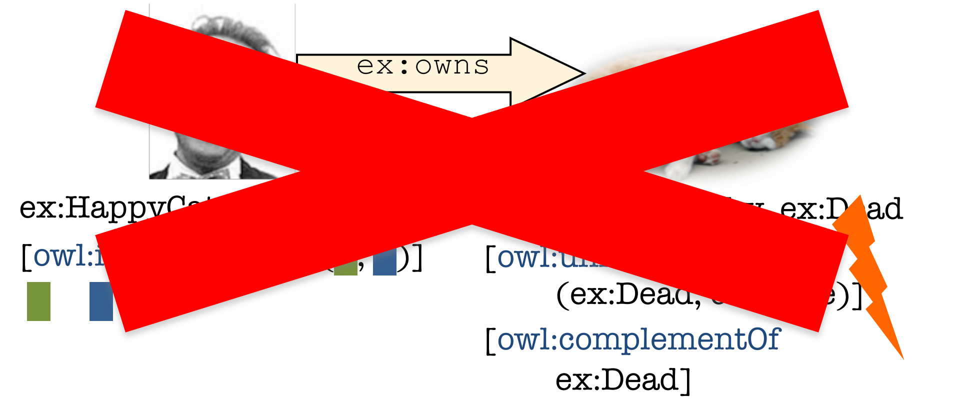
ex:schrödinger rdf:type ex:HappyCatOwner .

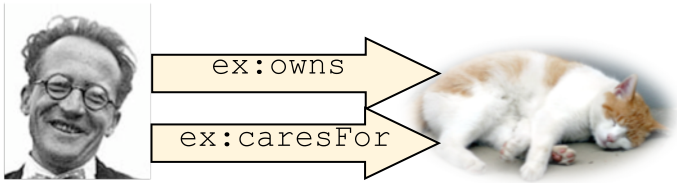
Tableau



ex:HappyCatOwner
[owl:intersectionOf (🟩, 🟦)]
🟩 🟦

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





ex:HappyCatOwner
[owl:intersectionOf (🟩, 🟦)]
🟩 🟦

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

OWL Reasoning with Tableaux

Knowledge Base

Satisfiable

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

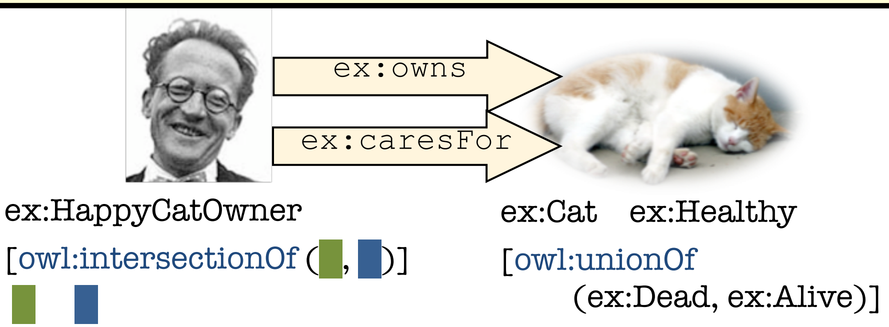
ex:Cat rdfs:subClassOf [owl:unionOf (ex:Dead, ex:Alive)] .

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])] .

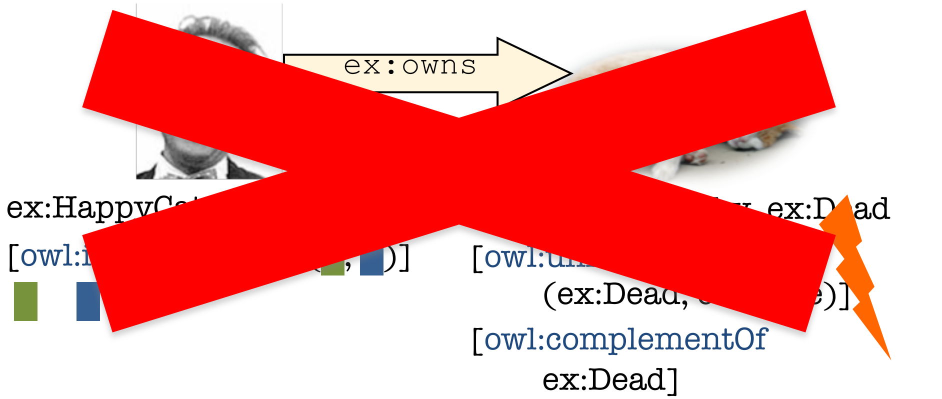
ex:schrödinger rdf:type ex:HappyCatOwner .

Tableau



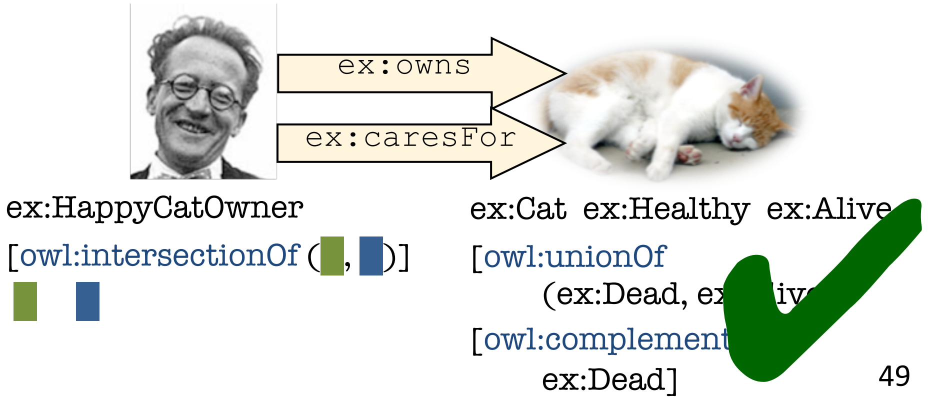
ex:HappyCatOwner
[owl:intersectionOf (🟩, 🟦)]
🟩 🟦

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



ex:HappyC
[owl:i , 🟦)]
🟩 🟦

ex:D ad
[owl:un
(ex:Dead, e)]
[owl:complementOf
ex:Dead]



ex:HappyCatOwner
[owl:intersectionOf (🟩, 🟦)]
🟩 🟦

ex:Cat ex:Healthy ex:Alive
[owl:unionOf
(ex:Dead, ex iv
[owl:complement
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 ex: <http://example.org/>
SELECT ?title ?author
WHERE
{ ?book ex:publishedBy <http://crc-press.com/uri> .
  ?book ex:title  ?title .
  ?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:

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

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

title	author
“Foundations of ...”	http://example.org/Hitzler
“Foundations of ...”	http://example.org/Krötzsch
“Foundations of ...”	http://example.org/Rudolph

Grouping Query Patterns

- Simple graph patterns are grouped with **{ }**

- Example:

```
{ { ?book ex:publishedBy <http://crc-press.com/uri> .
    ?book ex:title  ?title . }
  {}
  ?book ex:author ?author
}
```

→ Useful with additional query features

Optional Patterns

- Optional parts can be specified with **`OPTIONAL`**
- Example:

```
{ ?book ex:publishedBy <http://crc-press.com/uri> .
  OPTIONAL { ?book ex:title  ?title . }
  OPTIONAL { ?book ex:author ?author . }
}
```

- → Parts of the result can be **unbound:**

book	title	author
`http://example.org/book1`	`“title 1”`	`http://example.org/johndoe`
`http://example.org/book2`	`“title 2”`	
`http://example.org/book3`	`“title 3”`	`_:a`
`http://example.org/book4`		

Alternative Patterns

Alternatives can be specified with **`UNION`**

Example:

```
{ ?book ex:publishedBy <http://crc-press.com/uri> .
  { ?book ex:title  ?title . } UNION
  { ?book ex:author ?author . }
}
```

→ Result = union of results for one of the alternatives
→ Parts of the result can be **unbound**

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

Filters

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

Example:

```
{ ?book ex:publishedBy <http://crc-press.com/uri> .
  ?book ex:price  ?price .
  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?