Knowledge Graphs
RDF Schema (RDFS)
Previously on “Knowledge Graphs”

• Is RDF more powerful than XML?

• XML is a markup language for information

• In XML, arbitrary elements and attributes can be defined

• XML tag names are meaningless for a computer

• RDF is a markup language for information

• In RDF, arbitrary classes and predicates can be defined

• RDF class and predicate names are meaningless for a computer
Today: Schemas and Ontologies

• Last week’s slides:
  • Node types (“classes”) and edge types (“properties”)
    – Are also referred to the “schema” of the graph (aka “ontology”)
    – Can be defined with further restrictions
      • e.g., an edge of type “author” links a publication to a person
  • Schemas and ontologies bring semantics to knowledge graphs
• Today:
  – Building simple ontologies with RDF Schema
  – Elements of RDF Schema
  – Automatic deduction with RDF Schema
Semantic Web Stack

here be dragons...

Knowledge Graph Technologies (This lecture)

Technical Foundations

Berners-Lee (2009): Semantic Web and Linked Data
What is Missing up to Now?

• Basic premise: knowledge graphs should encode information so that humans and computers can understand it

• But what does understand actually mean?

"Madrid is the capital of Spain."
Semantics

• Let's look at that sentence:
  – "Madrid is the capital of Spain."

• Published in a knowledge graph (i.e., using RDF):

• How many pieces of information can we (i.e., humans) derive from that sentence?
  – (1 piece of information = 1 statement <S,P,O>)
  – Estimations? Opinions?
Let's look at that sentence:

- "Madrid is the capital of Spain."

We can get the following information:

- "Madrid is the capital of Spain."
- "Spain is a state."
- "Madrid is a city."
- "Madrid is located in Spain."
- "Barcelona is not the capital of Spain."
- "Madrid is not the capital of France."
- "Madrid is not a state."
- ...
How do Semantics Work?

Cities are capitals of states. Each state has exactly one capital. A city cannot be the capital of more than one state. ...

"Madrid is the capital of Spain."
• Saussure's idea of a *linguistic sign*

• Ferdinand de Saussure (1857-1913):
  – Signifier (signifiant) and signified (signifié) cannot be separated from each other

"tree"
An Excursion to Linguistics

- The triangle of reference

So, how do Semantics Work?

• Lexical semantics
  – Meaning of a word is defined by relations to other words

• Extensional semantics
  – Meaning of a word is defined by the set of its instances

• Intensional semantics, e.g., feature-based semantics
  – Meaning of a word is defined by features of the instances

• Prototype semantics
  – Meaning of a word is defined by proximity to a prototypical instance

• ...
Lexical Semantics

- Defining semantics by establishing relations between words

Diagram:
- Weapon
- Synonym
- Arm
- Firearm
- Hyponym
- Arm
- Shoulder
- Body Part
- Hyponym
- Meronym
- Homonym
Extensional Semantics

• Listing instances
  – EU members are Austria, Belgium, Bulgaria, …, Sweden, UK.

• *Angela Merkel* == *Chancellor of Germany*
  – both terms have the same extension
Intensional Semantics

- Describes features of things, i.e., *semes*
- A seme is a feature that distinguishes the meaning of two words

<table>
<thead>
<tr>
<th>Word</th>
<th>has wings</th>
<th>can swim</th>
<th>has fur</th>
<th>can fly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duck</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Bird</td>
<td>+</td>
<td>O</td>
<td>-</td>
<td>O</td>
</tr>
<tr>
<td>Bee</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Dolphin</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Intensional vs. Extensional Semantics

- Intensionally different things can have the same extension
- Classic example: morning star and evening star

<table>
<thead>
<tr>
<th>Word</th>
<th>Celestial body</th>
<th>bright</th>
<th>visible in the morning</th>
<th>visible in the evening</th>
</tr>
</thead>
<tbody>
<tr>
<td>Morning star</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Evening star</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- both have the same extension (i.e., Venus)
Intensional vs. Extensional Semantics

• The extension can change over time without the intension changing
  – e.g., “student”
  – does that change the semantics?

• Intension may also change over time
  – technological achievements (e.g., intension of *ship*)
  – changes in moral values (e.g., intension of *marriage*)

• Extension may also be empty, e.g.
  – Unicorn
  – Martian
  – Yeti (?)
Intensional vs. Extension Semantics

• ...explained by two well-known experts in the field :-)

...
Prototype Semantics

• A small experiment:
  – Close your eyes, and imagine a bird!
Prototype Semantics

- So far, intensional and extensional semantics are based on boolean logics (i.e., there's only “true” and “false”)
- Prototype Semantics: a more fuzzy variant

Jean Aitchison: Words in the Mind (1987)
Semantic Shift

“Mask” 2019

“Mask” 2022
How do Semantics Work?

- We have learned: Semantics define the meaning of words
- That is what we do with ontologies
  - using methods from lexical, intensional, and extensional semantics

http://walkinthewords.blogspot.com/2008/05/linguistic-cartoon-favorites-semantics.html
How do Semantics Work?

Cities are capitals of states.
Each state has exactly one capital.

A city cannot be the capital of more than one state.

"Madrid is the capital of Spain."
Semantics Formalized

\[
\text{City}(x) \iff \exists y: \text{capitalOf}(x,y) \\
\text{State}(y) \iff \exists x: \text{capitalOf}(x,y) \\
\text{locatedIn}(x,y) \iff \text{capitalOf}(x,y) \\
\ldots
\]

Ontologies

• "An ontology is an explicit specification of a conceptualization."\(^1\)

• Ontologies encode the knowledge about a domain

• They form a common vocabulary
  – and describe the semantics of its terms

What is an Ontology?

• Ontology (without a or the) is the philosophical study of being
  – greek: ὄντος (things that are), λόγος (the study)
  – A sub discipline of philosophy

• In computer science (with a or the)
  – a formalized description of a domain
  – a shared vocabulary
  – a logical theory
Ontologies – Further Definitions

• Guarino und Giaretta (1995):
  "a logical theory which gives an explicit, partial account of a conceptualization"

• Uschold und Gruninger (1996):
  "shared understanding of some domain of interest"
  "an explicit account or representation of some part of a conceptualisation"

• Guarino (1998):
  "a set of logical axioms designed to account for the intended meaning of a vocabulary"
Essential Properties of Ontologies

• Explicit
  – Meaning is not “hidden” between the lines

• Formal
  – e.g., using logic or rule languages

• Shared
  – Martin Hepp: "Autists don't build ontologies"
  – An ontology just for one person does not make much sense

• Partial
  – There will (probably) never be a full ontology of everything in the world
Classifications of Ontologies

Degree of Formality

- Informal ontologies
  - Catalog
  - Glossary
  - Thesaurus
- Light-weight ontologies
  - Informal Taxonomy
- Formal ontologies
  - Formal Taxonomy
  - Formal Instances
- Heavy-weight ontologies
  - Frames
  - Value Restrictions
  - Logic Constraints

The Oldest Ontology

Porphyry, Greek philosopher, ca. 234-305

**Figure 1.1** Tree of Porphyry, translated from a version by Peter of Spain (1239)
Encoding Simple Ontologies: RDFS

- A W3C Standard since 2004

- Most important element: classes

  :State a rdfs:Class .

- Classes form hierarchies

  :EuropeanState rdfs:subClassOf :State .
Multiple inheritance is possible

Convention for this course: unlabeled arrows = rdfs:subClassOf
Properties in RDF Schema

• Properties are the other important element
• resemble two-valued predicates in predicate logic

:capitalOf a rdf:Property .

• Properties also form hierarchies

:capitalOf rdfs:subPropertyOf :locatedIn .
Domains and Ranges of Properties

- In general, properties exist independently from classes
  - i.e., they are first class citizens
  - this is different than OOP or ERM

- Defining the domain and range of a property:
  
  :capitalOf rdfs:domain :City .
  :capitalOf rdfs:range :Country .

- Domain and range are inherited by sub properties
  - They can also be further restricted
Predefined Properties

- We have already seen

  rdf:type
  rdfs:subClassOf
  rdfs:subPropertyOf
  rdfs:domain
  rdfs:range
Further Predefined Properties

• Labels:
  :Germany rdfs:label "Deutschland"@de .
  :Germany rdfs:label "Germany"@en .

• Comments:
  :Germany rdfs:comment "Germany as a political entity."@en .

• Links to other resources:
  :Germany rdfs:seeAlso <http://www.deutschland.de/> .

• Link to defining schema:
  :Country rdfs:isDefinedBy
  <http://foo.bar/countries.rdfs> .
URIs vs. Labels

• A URI is only a unique identifier
  – it does not need to be interpretable
    http://www.countries.org/4327893

• Labels are made for human interpretation
• ...and can come in different languages:
  countries:4327893 rdfs:label "Deutschland"@de .
  countries:4327893 rdfs:label "Germany"@en .
  countries:4327893 rdfs:label "Tyskland"@sv .
  ...

URIs vs. Labels

- Labels and comments can also be assigned to RDFS elements:

  :Country a rdfs:Class .
  :Country rdfs:label "Land"@de .

  :locatedIn a rdf:Property .
  :locatedIn rdfs:label "liegt in"@de .
  :locatedIn rdfs:label "is located in"@en .
  :locatedIn rdfs:comment "refers to geography"@en .
RDF Schema and RDF

• Every RDF Schema document is also an RDF document
• This means: all properties of RDF also hold for RDFS!

• Non-unique Naming Assumption
  
  schema1:Country a rdfs:Class .
  schema2:State a rdfs:Class .

• Open World Assumption
  
  :Country rdfs:subClassOf :GeographicObject .
  :City rdfs:subClassOf :GeographicObject .
Our First Ontology

- States, cities, and capitals

  :State a rdfs:Class .
  :City a rdfs:Class .
  :locatedIn a rdf:Property .
  :capitalOf rdfs:subPropertyOf :locatedIn .
  :capitalOf rdfs:domain :City .
  :capitalOf rdfs:range :State .

What do We Gain Now?

:Country a rdfs:Class .  
:City a rdfs:Class .  
:locatedIn a rdfs:Property .  
:capitalOf rdfs:subPropertyOf :locatedIn .  
:capitalOf rdfs:domain :City .  
:capitalOf rdfs:range :Country .  

What do We Gain Now?

+ :capitalOf rdfs:domain :City
→ :Madrid a :City .

+ :capitalOf rdfs:range:Country
→ :Spain a :Country .

+ :capitalOf rdfs:subPropertyOf :locatedIn .
→ :Madrid :locatedIn :Spain .
Reasoning with RDF

• RDF Schema allows for *deductive* reasoning on RDF

• This means:
  – given facts and rules,
  – we can derive new facts

• The corresponding tools are called *reasoner*

• Opposite of deduction: *induction*
  – deriving models from facts
  – see, e.g., lectures on data mining and machine learning
A Bit of History

• Aristotle (384 – 322 BC)
• Syllogisms
  – Deriving facts using rules

• Example:
  All men are mortal.
  Socrates is a man.
  → Socrates is mortal.
A Bit of History

Penguins are black and white. Some old TV shows are black and white. Therefore, some penguins are old TV shows.

Logic: another thing that penguins aren’t very good at.

Cartoon Copyright: Randy Glasbergen, http://www.glasbergen.com/
Interpretation and Entailment

• Entailment
  – The set of all consequences of a graph

• Mapping a graph to an entailment is called *interpretation*

• Simplest Interpretation:
  – \(<s,p,o> \in G \rightarrow <s,p,o> \in \text{Entailment}\)

• This interpretation creates all statements explicitly contained in the graph.
• But the *implicit* statements are the interesting ones!
Interpretation using Deduction Rules

- RDF interpretation can be done using RDFS deduction rules
- Those create an entailment
  - using existing resources, literals, and properties
  - creating additional triples like \(<s,p,o>\)
  - e.g.,
    - \(<\text{Madrid}, \text{rdf:type}, \text{City}>\)
    - \(<\text{Madrid}, \text{located\_in}, \text{Spain}>\)
- Note:
  - no new resources, literals, or properties are created!
Reasoning with Deduction Rules

• Deduction rules are an interpretation function
• Simple reasoning algorithm (a.k.a. forward chaining):

Given: an RDF Graph G
a set of deduction rules R
Entailment E = G
Repeat
  M := { }
  For all rules in R
    For each statement S in E
      Apply R to S
      If E does not contain consequence
        Add consequence to M
    Add all elements in M to E
  until M = { }
# Deduction Rules RDF Schema (Selection)

<table>
<thead>
<tr>
<th>ID</th>
<th>Condition</th>
<th>Consequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs2</td>
<td>( s \ p \ o ) . ( p \ \text{rdfs:domain} \ c ) .</td>
<td>( s \ \text{rdf:type} \ c ) .</td>
</tr>
<tr>
<td>rdfs3</td>
<td>( s \ p \ o ) . ( p \ \text{rdfs:range} \ c ) .</td>
<td>( o \ \text{rdf:type} \ c ) .</td>
</tr>
<tr>
<td>rdfs7</td>
<td>( p_1 \ \text{rdfs:subPropertyOf} \ p_2 ) . ( s \ p_1 \ o ) .</td>
<td>( s \ p_2 \ o ) .</td>
</tr>
<tr>
<td>rdfs9</td>
<td>( s \ \text{rdf:type} \ c_1 ) . ( c_1 \ \text{rdfs:subClassOf} \ c_2 ) .</td>
<td>( s \ \text{rdf:type} \ c_2 ) .</td>
</tr>
<tr>
<td>rdfs10</td>
<td>( c \ \text{rdf:type} \ \text{rdfs:Class} ) .</td>
<td>( c \ \text{rdfs:subClassOf} \ c ) .</td>
</tr>
<tr>
<td>rdfs11</td>
<td>( c_1 \ \text{rdfs:subClassOf} \ c_2 ) . ( c_2 \ \text{rdfs:subClassOf} \ c_3 ) .</td>
<td>( c_1 \ \text{rdfs:subClassOf} \ c_3 ) .</td>
</tr>
</tbody>
</table>

**rdfs:subclassOf is reflexive and transitive** *(same for rdfs:subPropertyOf)*

Applying Deduction Rules

- Another Example

```turtle
:Employee a rdfs:Class .
:Employee rdfs:subClassOf :Human .
:Room a rdfs:Class .
:worksIn rdfs:subPropertyOf :hasOffice .
:hasOffice rdfs:domain :Employee .
:hasOffice rdfs:range :Room .

:Tim :worksIn :B626B01 .
```
Applying Deduction Rules

- Example:
  - \( :\text{Tim} :\text{worksIn} :\text{B626B01} . \)
  - \( :\text{worksIn} \text{ rdfs:subPropertyOf} :\text{hasOffice} . \)

<table>
<thead>
<tr>
<th>ID</th>
<th>Condition</th>
<th>Consequence</th>
</tr>
</thead>
</table>
| rdfs7 | \( p_1 \text{ rdfs:subPropertyOf} p_2 . \)
        | \( s\ p_1\ o . \)                          |                           |

\(-\rightarrow :\text{Tim} :\text{hasOffice} :\text{B626B01} . \)
Applying Deduction Rules

• Example:

\[ \text{Tim} :\text{hasOffice} :\text{B626B01} . \]
\[ :\text{hasOffice} \text{ rdfs:domain} :\text{Employee} . \]

\begin{tabular}{|c|c|c|}
\hline
ID & Condition & Consequence \\
\hline
rdfs2 & s p o . & s \text{ rdf:type c} . \\
 & p \text{ rdfs:domain} c . & \text{Tim rdf:type Employee} . \\
\hline
\end{tabular}
Applying Deduction Rules

- Example:

\[
\text{:Tim rdf:type :Employee.}
\]
\[
\text{:Employee rdfs:subClassOf :Human .}
\]

\[
\text{s= :Tim}
\]
\[
\text{c1=:Employee}
\]
\[
\text{c2=:Human}
\]

<table>
<thead>
<tr>
<th>ID</th>
<th>Condition</th>
<th>Consequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs9</td>
<td>s rdf:type c1 .</td>
<td>s rdf:type c2 .</td>
</tr>
<tr>
<td></td>
<td>c1 rdfs:subClassOf c2 .</td>
<td></td>
</tr>
</tbody>
</table>

\[
\rightarrow \text{:Tim rdf:type :Human .}
\]
Forward Chaining

- Example revisited:
  
  :Employee a rdfs:Class .
  :Employee rdfs:subClassOf :Human .
  :Room a rdfs:Class .
  :worksIn rdfs:subPropertyOf :hasOffice .
  :hasOffice rdfs:domain :Employee .
  :hasOffice rdfs:range :Room .

- :Tim :worksIn :B626B01 .

- \( \rightarrow :\text{Tim hasOffice} :B626B01 \)
  
  \( \rightarrow :\text{Tim rdf:type} \ \text{Employee} . \)
  
  \( \rightarrow :\text{Tim rdf:type} \ \text{Human} . \)
What if there are Multiple Domains/Ranges?

• Example for social networks:

```rdfs
:knows rdfs:domain :Person .
:knows rdfs:domain :MemberOfSocialNetwork .
```

• What should be the semantics here?
  – Everybody who knows someone
    is a person \textit{and} a member of a social network
  – Everybody who knows someone
    is a person \textit{or} a member of a social network
The Rules will Tell Us

:knows rdfs:domain :Person. (a0)
:knows rdfs:domain :MemberOfSocialNetwork . (a1)
:Peter :knows :Stephen . (a2)

(rdfs2+a0+a2) :Peter rdf:type :Person . (a3)
(rdfs2+a1+a2) :Peter rdf:type :MemberOfSocialNetwork . (a4)
...

• This chain works for each object
  – it is always contained in both classes
    → i.e., the intersection semantics hold
What have We Gained?

- Let's look at that sentence:
  - "Madrid is the capital of Spain."

- We can get the following information:
  - "Madrid is the capital of Spain." ✔
  - "Spain is a state." ✔
  - "Madrid is a city." ✔
  - "Madrid is located in Spain." ✔
  - "Barcelona is not the capital of Spain." ✗
  - "Madrid is not the capital of France." ✗
  - "Madrid is not a state." ✗
  - ...

What we Cannot Express (up to Now)

- "Every state has exactly one capital"
  - Property cardinalities
- "Every city can only be the capital of one state."
  - Functional properties
- "A city cannot be a state at the same time."
  - Class disjointness
- ...

- For those, we need more expressive languages than RDFS!
What we Cannot Express (up to Now)

• "Every state has exactly one capital"
  – i.e., "A state cannot have more than one capital."

• “Every city can only be the capital of one state."
  – i.e., "A city cannot be the capital of two different states."

• "A city cannot be a state at the same time."
What we Cannot Express (up to Now)

• Note: there is no negation in RDF and RDFS

• This means, we cannot produce any contradictions
  – This makes reasoning easy
  – But it also restricts the utility
  – Example:
    Mammals do not lay eggs
    Penguins lay eggs
    → Penguins are not mammals

• We will get to know formalisms that support negation
  – and learn how to do reasoning with them
What we Cannot Express (up to Now)

• The missing negation perfectly fits the AAA principle
  – Anybody can say anything about anything

• ...and the Open World Assumption

• Any new knowledge will always fit to the knowledge that is already there
  – This principle is called “monotonicity”
What we Cannot Express (up to Now)

• Kurt Gödel (1906-1978)
• Logic systems are either
  – not very powerful or
  – not free of contradictions
• RDF Schema belongs to the first class
What we Cannot Express (up to Now)

• Jim Hendler (*1957)

• "A little semantics goes a long way."
Just a moment

- "We cannot produce any contradictions"
- so what about
  - :Peter a :Baby .
  - :Peter a :Adult .

- That is a contradiction!

- Well, it is – for us human beings
- But a computer will not know
  - Non-unique name assumption!
Semantic Web Stack

Knowledge Graph Technologies (This lecture)

Technical Foundations

here be dragons...

Berners-Lee (2009): Semantic Web and Linked Data
Questions?