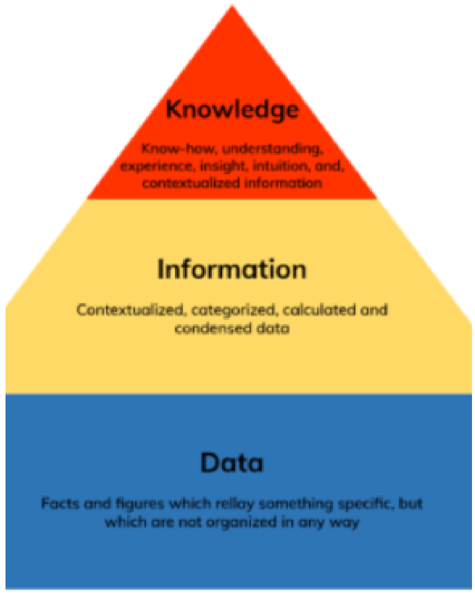
Lecture 1: Introduction

- Data - Information - Knowledge
- Tacit/informal vs explicit/formal knowledge
- Towards Knowledge Graphs
- Propositional logic for expressing knowledge



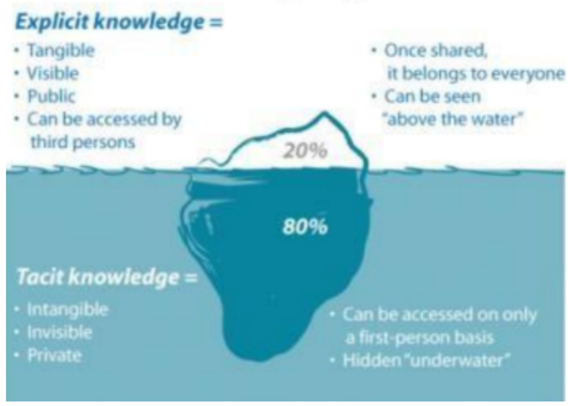
Data - Information - Knowledge

- **Data** are individual facts that are out of context, have no meaning, and are difficult to understand.
- **Information** is a set of data in context with relevance to one or more people at a point in time or for a period of time.
- **Knowledge** is the fact or condition of knowing something with familiarity gained through experience or association.

Knowledge is information that has been retained with an understanding about the significance of that information.

To increase the usefulness of data and bake sense of certain values data needs to be **processed** and given a context.
What can be done with information/data requires **knowledge**. Knowledge = information + rules.

<u>DATA PREPARATION ACCOUNTS FOR 80% OF THE WORK OF A DATA SCIENTIST</u>



Tacit/informal vs explicit/formal knowledge

- **Tacit knowledge** (*implicit knowledge*) is the knowledge a person retains in their mind
- **Explicit knowledge** (formal knowledge) is knowledge that has been formalized, codified and stored.

Formal knowledge can help us to *interpret* and *reuse* data and make it reusable for other purposes ⇒ goal = predictable inference.

Knowledge graphs

A useful way of representing data, information and knowledge …
... that are **heterogeneous**
...in such a way that others can **interpret** a piece of data correctly
...by making the **semantics** of a piece of information explicit
...using **graph** (network)
...explicitly in the **Web**

We don't have a **web of data** because data is controlled by applications and **each application keeps it to itself.**

Imagine a Semantic Web of Data

- Websites publish their information in a **machine readable** format;

- The data published by different sources is **linked**;
- Enough **domain knowledge** is available to machines to make use of the information;
- Machines can **find and combine** published information in appropriate ways to answer the user's information needs.

4 PROPOSALS

P1. Give all things a name
P2. The names are addresses on the Web
P3. Relations form a graph between things
P1 + P2 + P3 = A (global) graph of Linked Data
P4. Make explicit the meaning of things

- Not just the **data**, but its underlying **model** as well
    - Assign **types** to things
    - Assign **types** to relations
    - Organize types in a **hierarchy**
- Rules for **calculating** with that knowledge impose **constraints** on possible interpretations

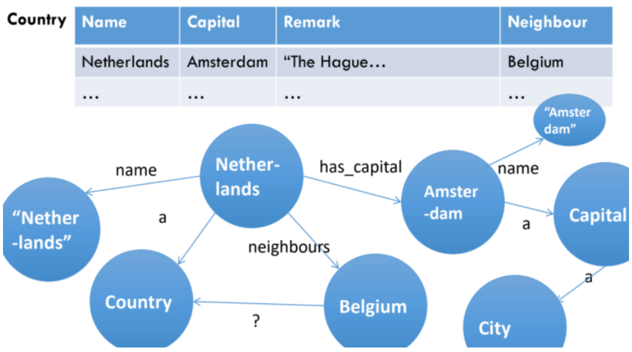
Querying vs Inferencing

To be able to do **inferences** over the Web of Data, the model needs to be **understandable by machines**, **formal semantics** needs to be shared**, inferences** should be predictable.

Lecture 2: Knowledge Graphs and Formal foundation

- Knowledge Graphs
- Formal Systems
- Propositional Logic as a Formal System
- Simple Knowledge Graph Logic

Formally representing Knowledge Graphs



What is a logic?

- A formal language
- **Syntax**
    - Which expressions are legal
- **Semantic**
    - What legal expressions mean
    - The meaning of each sentence with respect to interpretations
- **Calculus:**
    - How to determine meaning for legal expressions

A logic of arithmetic: SYNTAX

- Unambiguous definitions of what sentences are well-formed.
    - 2 terms with a comparator between them
    - A term is either a Natural Number, a Variable, or a complex term.
    - A complex term is an operator +, -, *, / applied to two terms in infix notation with parentheses: "(term 1 operator term 2)"

A logic of arithmetic: SEMANTICS

- Truth is defined in terms of assignment for variables. Let V be the set of variables, then I^v: V → N is an *assignment*, a function that assigns natural numbers to each variable.
- x + 2 >= y is true w.r.t. an assignment I^v where $I^v(x) = 7$ and $I^v(y) = 1$, and many more.

We say that I^v is a <u>model</u> of a formula F if $I^v(F)$ is true.

Entailment: predictable inference!

- **A formula F entails another formula G (F |= G) if for all variable assignments $I^v(F)$ is true implies that $I^v(G)$ is true.**
- F entails G if G is true in all models of F

Example

- *If $I^v(x + y < 6)$ is true, then $I^v(x) + I^v(y) < 6$ is true.*
- *But then $I^v(x) < 6 - I^v(y)$, which implies $I^v(x) < 6$, as $I^v(y) >= 0$, as it is a Natural number. But then $I^v(x)$ is also smaller than 10.*

A logic of concept hierarchies: SYNTAX

A concept is "an abstraction or generalization from experience or the result of a transformation of existing ideas".

Syntax

- Let C be a finite set of **concept names.**
- If c1 and c2 in C, then (c1 subclassOf c2) is an **axiom** in LCH
- An LCH **knowledge base** is a set of LCH axioms

Semantics

- Let U be a universe, a set of arbitrary objects.
  I^c: C → P(U) is a function that **assigns subsets of the domain** to concept names.
- An axiom (c1 subclassOf c2) is true w.r.t an assignment if
  $I^c(c1) \subseteq I^c(c2)$ is then called a model for the axiom
- An assignment is a model of a knowledge base if it's a model of all its axioms
- An axiom (c1 subclassOf c2) is entailed by a knowledge base KB if it is true in all models of KB.

Propositional logic as a Formal System

A *declarative sentence* (*or proposition*) is a statement that is *true* or *false.*

Syntax

- Propositional variables: e.g. p, q, r, …
- Connectives: e.g. → , v, …
- Declarative sentences (propositions)
  Infix notation: $(p_1 \Rightarrow p_2)$
  Prefix notation: $[\Rightarrow,[[p_1],[p_2]]]$

Semantic equivalence: Formulas x and y are *semantically equivalent,* notation x ≡ y, if they have identical columns in their truth tables.

Tautology: A formula is a tautology if: its column in a truth table has T on every line ⇒ it's true for every valuation.

Contradiction: False for every valuation

Semantic entailment: A formula x is semantically entailed by the premises y_1, … y_n

($y_1, \dots y_n$ |= x) if for every valuation that makes all formulas $y_1, \dots y_n$ true it also makes x true

Simple Knowledge Graph Logic

SYNTAX

- Two different types of “things”:
    - **Vocabulary V** (no distinction between Objects and Literals)
    - **Predicates P** (Relations)
- Start with the **Triples**
    - T = V x P x V
    - Inductively: If $r_1, r_2 \in V$, $p \in P$ then also $(r_1, p, r_2) \in T$
- Construct Knowledge Graphs
    - A knowledge Graph is a set of triples $t \in T$.

SEMANTICS (grounded graphs)

- Syntactic objects are strings ()
- We need to assign values to triples, they are the mathematical object of a graph:
- Let an interpretation I consist of:
    - A set IR, a **universe,** a set of arbitrary objects
    - A function I^R: V → IR assigns an element of the domain to each word in the vocabulary
    - A function I^P: P → Powerset(IR x IR)

A triple (s p o) is true w.r.t. an interpretation I = (IR, I^R, I^P) iff $(I^R(s), I^R(o)) \in I^P(p)$, which is then called **a model** for the triple.

An interpretation is a **model of a knowledge base** if it is a model of all its triples
A set of triples is **entailed** by a knowledge graph if it is a subgraph of the knowledge graph

Lecture 3: Knowledge Graph Logic and KGs on the Web

- Knowledge and Data on the Web
- Knowledge Graphs on the Web
- Principles for Linked and Semantic Data on the Web
- RDF: Syntax(es), Semantics
- SPARQL query language

Knowledge and Data on the current Web

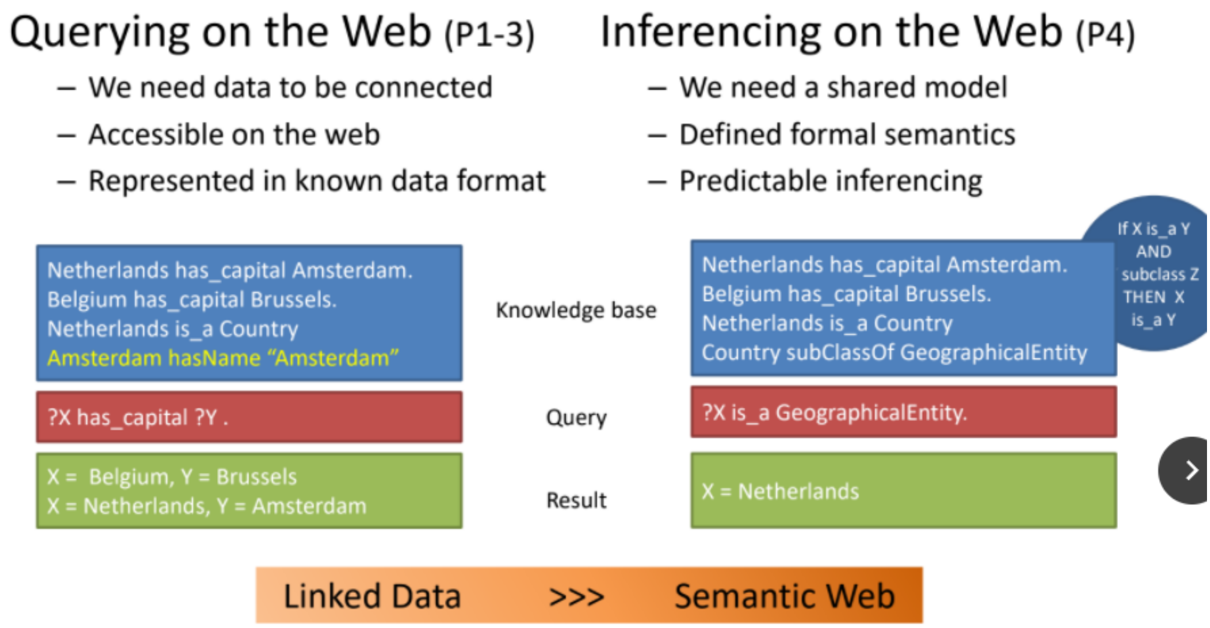
- The Web of Documents
    - Many pages on the internet use data that’s usable by humans but machines cannot read and use it.
- Web data != Web **of** Data
    - JSON only prescribes **data structure**
    - Data is **accessible** but still locked in **silos** ⇒ Data ownership = market share
    - Data **integration** is a major issue
- Linking datasets: **Linked Data**, navigable for **machines**, useful for **people**!

4 PROPOSALS

P1. Give all things a name

P2. The names are addresses on the Web
P3. Relations form a graph between things
P1 + P2 + P3 = A (global) graph of Linked Data
P4. Make explicit the meaning of things
- Not just the **data**, but its underlying **model** as well
    - Assign **types** to things
    - Assign **types** to relations
    - Organize types in a **hierarchy**
- Rules for **calculating** with that knowledge impose **constraints** on possible interpretations

Calculating with Knowledge = **inferencing** or **reasoning**
Inferencing is algorithmic manipulation of **knowledge** to derive new knowledge, where the **meaning** of words is **not needed**!



Linked Open Data?
- **Open Data** is about *licenses* to allow reuse
- **Linked Data** is about *technology* for interoperability
- Structured data not documents
- Graph (networked) data!
- W3C Web **standards** stack

Technology: standards, standards, standards
Uniform Resource Identifier (URI) to avoid naming conflicts
The **Hypertext Transport Protocol** (HTTP) to access and move data
Namespaces and **namespace prefixes** to group and abbreviate URI's ⇒ saves memory
The **Resource Description Framework** (RDF) as common data model
- Various **programming libraries** for accessing and manipulating RDF
- **SPARQL** language for querying data over HTTP
- Triple stores for **storing** large volumes of RDF data
- A family of increasingly expressive **formal languages** for **modelling**
- **Ontology editors** for defining semantics
- **Reasoners** for inferring new knowledge

RDF is a **data model** for data interchange on the Web

- It facilitates **data merging** even if the underlying schemas/models differ
- It extends the **linking structure** of the Web to use URIs to name the **relationships** between things
- It allows data to be mixed, exposed and shared

RDF - Triples

- All information in RDF is expressed as **triples** (subject, predicate, object)
- triple = statement or fact
- Elements of an RDF triple are either a URI reference, a blank node or a literal)

A RDF talks about **resources** (almost anything is a resource), resources are **identified by** URIs, or URIs **denote** resources. (URIs can only refer to a resource, they are not the resource, and multiple URIs can denote the **same** resource)

Internationalised Resource Identifiers (IRIs) are URIs that allow unicode characters.

RDF - Literals

- Literals are used to represent “literal” data values
- All literals have a **data type**, which are also **resources**
- One can specify the **language** of a string using a **language tag**

RDF - Graphs

- An RDF graph is a **set of triples**
- In practice, many RDF Graphs **have URIs** themselves

RDF - Why HTTP URIs, triples and graphs

- HTTP URIs have **a global scope**m unique throughout the Web
    - Grounded in **society** and help to avoid **name clashes**
- HTTP URIs are also **addresses**
    - Exploits Web **browsing** and tracks data by **following** the resource identifiers
- Any information can be transformed to **triples**

    Tabular | row | column | cell

    Trees | parent | path | child
- Relationships are made **explicit** in triples
- Graphs is a single but **high versatile** format
- basic set operations are well-defined
    - **merging** two RDF graphs? take their union

Blank nodes are resources without a URI ⇒ existential quantifiers

	subject	predicate	object
URI References	✓	✓	✓
Literals	✗	✗	✓
Blank Nodes	✓	✗	✓

Turtle - a convenient, human readable/writable syntax

- **Comments** start with a ‘hash’ character
- **Full URIs** are surrounded by < and >
- **Statements** are triples. They are terminated by a period.
- Use ‘a’ to abbreviate rdf:type
- **Namespace prefixes** are declared with **@prefix**
- A **default namespace** can be declared as well ⇒ **@prefix** : <http://example.org/>.
- **Literal values** are enclosed in double quotes
    - Possibly with **language** or **type** information.
    - **Numbers** and **booleans** can be written without quotes

… statements may share a **subject** with ';'

```
dbpedia:Amsterdam      :officialName    "Amsterdam";
                       :areaTotal       219320000 ;
                       :leaderName      dbpedia:Eberhard_van_der_Laan .
```

… statements may share **subject** and **predicate** with ','

```
dbpedia:The_Netherlands  rdfs:label      "Nederland"@nl,
                                         "The Netherlands"@en,
                                         "Pays-Bas"@fr .
```

… and in combination:

```
dbpedia:The_Netherlands  rdfs:label      "Nederland"@nl,
                                         "The Netherlands"@en,
                                         "Pays-Bas"@fr ;
                         :capital        dbpedia:Amsterdam .
```

Blank nodes are designated with **underscores** or **[…]** .

```
dbpedia:VU       dbo:address    _:someAddress .

_:someAddress    dbo:place      dbpedia:Amsterdam ;
                                dbo:street      "De Boelelaan" ;
                                dbo:number      "1081" ;
                                dbo:postcode    "1081 HV" .
```

```
dbpedia:VU       dbo:address
          [
                 dbo:place      dbpedia:Amsterdam ;
                 dbo:street     "De Boelelaan" ;
                 dbo:number     "1081" ;
                 dbo:postcode   "1081 HV"
          ] .
```

NB: underscore-styled node names are only unique within the file

Lecture 4: Knowledge Graph Logic and KGs on the Web

- RDF Semantics
- Querying RDF with SPARQL

Model theoretic semantics of RDF

- If, in a graph G, we replace each blank node x by A(x), then we obtain a graph G’, which we call a *grounding* of G.
- I |= G iff I |= G’ for **at least one** grounding G’ of G
- Theorem: A |= B iff B can be obtained from A by replacing some nodes in A by blank nodes

RDF- Blank Nodes as Variables

- Blank nodes are **resources** without a URI (in current context!)
- You can use them as a placeholder (even when a URI might exist for it somewhere else)

Formal versus **Social Semantics?**

- RDF allows us to:
    - **Very precisely** state what is true
    - Let anybody do that about anything they want
    - The point is to **collaborate** on defining things precisely: industry standards organisations

Querying with SPARQL

RDF: Where to find it?

1. As **separate files,** e.g. as .ttl, .rdf .nt, etc.
2. **Integrated** with Web pages (RDFa/Microdata)
3. Accessible through **content negotiation**
4. In RDF-specific databases called **triple stores**

SPARQL - Summary

- Linked Data is usually stored in **triple stores**
- **SPARQL** is the query language for the Web of Linked Data
- Queries are sent to **SPARQL endpoints** over **HTTP**, and they describe **graph patterns** with **variables**.

- **six** types of queries ⇒ SELECT, CONSTRUCT, INSERT, DELETE, ASK, DESCRIBE
- SELECT returns a table with variable bindings
- CONSTRUCT returns an RDF graph
- ASK returns true or false
- DESCRIBE returns an RDF graph
- INSERT is like CONSTRUCT, but inserts the graph into the triple store

- WHERE clause specifies a **graph pattern** ⇒ an RDF graph with some nodes & edges as **variables**
- UNION ⇒ at least one should match
- OPTIONAL ⇒ does not need to match
- FILTER ⇒ clause have to be validated
- ORDER BY ⇒ to sort by desc for example
- DISTINCT results
- LIMIT the number of results

SPARQL - Summary

- Linked Data is usually stored in **triple stores**
- **SPARQL** is the query language for the Web of Linked Data
- Queries are sent to **SPARQL endpoints** over **HTTP**
- Queries describe **graph patterns** with **variables**
- Graph patterns **match** the **RDF graphs** stored in the triple store
- Results are returned as a table with variable bindings

Lecture 5: RDF and Inference

- Publishing and Consuming RDF
- Entailment and Inferencing

Triplestores are purpose-build (graph) databases to deal with RDF data.
Data can be stored persistently on disk or in memory.
There is a fast query because of: Dictionaries, Indexing, Statistics
Triplestores: Optimized for fast querying

- **Dictionary** ⇒ replace names by addresses
- Data is **indexed** for fast access
- Triple stores use this to do efficient handling of JOINS in SPARQL

Formal Semantics

- **Knowledge Representation -** Principles
    - A set of **reserved symbols**
    - A set of **variables** that may be assigned values (e.g. *p* and *q*)
    - A **formal semantics** defining the meaning of formulas (e.g. when they are true) and the semantic relation between formulas (e.g. entailment)
    - A set of **inference rules** for manipulating **formulas** that use those symbols

- **Knowledge Representation** - Formulas
    - Conditions on class membership
      all mammals are warm-blooded
    - Relations between classes
      all cities are populated
    - Assertions of class membership
      Amsterdam is a city
    - Characteristics of properties
      hasCapital only relates countries to cities
    - Assertions of property relations
      hasCapital(Netherlands, Amsterdam)
    - Assertions of equality
      morning star = evening star = venus

Specify that something (denoted by a URI) is a **class** (or a property).
ex:Country rdf:type rdfs:Class .

Specify that something is a **member** of a **class** (strikingly similar)
ex:Netherlands rdf:type ex:EuropeanCountry .

Specify that something is a **subclass** of another **class**
ex:EuropeanCountry rdfs:subClassOf ex:Country .

Specify that some **property** always relates members of **specific classes**
ex:containsCity rdfs:domain ex:Country .
ex:containsCity rdfs:range ex:City .

Specify that some **property** is a specification of another property
ex:hasCapital rdfs:subPropertyOf ex:containsCity .
ex:hasCapital rdfs:range ex:Capital .

Formulas are axioms that restrict the possible interpretations of the world.
Entailment is defined as truth in these restricted interpretations

RDF Schema - Summary

- WIthout **formal semantics**, the Web of Data is **meaningless**
- “Distinction” between **classes, properties** and **instances (schema vs. data)**
- RDF Schema **reserved symbols**
  rdfs:Class, rdfs:domain, rdfs:range etc.
- **Entailment rules** are expressed using **reserved symbols**
- Inferencing is the application of **entailment rules** to **formulas** to produce **new facts**
- RDF Schema is **not** very expressive

1.	s	p	o	⟹	s p o *OR* o	rdf:type rdf:type rdf:type rdf:type	rdfs:Resource rdf:Property rdfs:Resource rdfs:Literal
2.	s	rdf:type	o	⟹	o	rdf:type	rdfs:Class
3.	s	rdf:type	rdfs:Class	⟹	s	rdfs:subClassOf	s
4.	p	rdf:type	rdf:Property	⟹	p	rdfs:subPropertyOf	p
5.	s y	rdfs:subClassOf rdfs:subClassOf	y z	⟹	s	rdfs:subClassOf	z
6.	p q	rdfs:subPropertyOf rdfs:subPropertyOf	q r	⟹	p	rdfs:subPropertyOf	r
7.	s p	p rdfs:subPropertyOf	o q	⟹	s	q	o
8.	s p	p rdfs:domain	o x	⟹	s	rdf:type	x
9.	s p	p rdfs:range	o x	⟹	o	rdf:type	x
10:	s o	rdf:type rdfs:subClassOf	o t	→	s	rdf:type	t

Lecture 6: RDF and Schema

RDF Schema - Observation

- No **strict** distinction between schema and data level
- RDF Schema **entailment** rules do not include **negation**
- No notion of **equality**
- Not prohibited to use the **reserved symbols** in formulas
-

- Terms for classes
  - rdfs:Class
  - rdfs:subClassOf
- Terms for properties
  - rdfs:domain
  - rdfs:range
  - rdfs:subPropertyOf
- Special classes
  - rdfs:Resource
  - rdfs:Literal
  - rdfs:Datatype
- Terms for collections
  - rdfs:member
  - rdfs:Container
  - rdfs:ContainerMembershipProperty
- Special properties
  - rdfs:comment
  - rdfs:seeAlso
  - rdfs:isDefinedBy
  - rdfs:label

RDF Schema - Additional Stuff

- rdfs:label ⇒ Specify the human readable label for a resource
- rdfs:comment ⇒ Comment on a resource
- rdfs:seeAlso ⇒ Refer to another resource

RDFS and other vocabularies

- **RDF** vocabulary and **RDFS** ⇒ reserved terms needed for the data model
  - rdf:Property ... is type of all properties
  - rdf:type … links resource to type
- **FOAF** Friend of a Friend ⇒ persons and relations between persons
  - foaf:Person … a person, alive, dead, imaginary
  - foaf:name … the name of a person
  - foaf:mbox … the email URI of a person
  - foaf:knows ... one person knows another person
- Dublin Core (**DC** and **DCTerms**) ⇒ bibliographic attributes (author, title, etc.)
  - dct:creator… a document's main author
  - dct:created… the creation date
  - dct:description… a natural language description
  - dct:title… the title of the document
- **DBPedia** Ontology ⇒ at the heart of the Web of Data
- **Geonames** ⇒ locations
- Data Cube (**QB**) ⇒ statistical data
- Simple Knowledge Organization System (**SKOS**) ⇒ concept hierarchies and mappings
- Web Ontology Language (**OWL**) ⇒ formal constraints on class membership

Schema.org ⇒ An initiative by Google, Microsoft, Yahoo etc. for search engine optimization

SKOS - RDFS Vocabulary for Thesaurus Modeling
→ **Thesauri**

- "Standard" terminology in a particular domain
- Often a **hierarchy** without formal semantics
- **WordNet** is a lexical resource
- **Getty** thesauri such as AAT, TGN, ULAN (art & architecture, geographic names, artist names)
- **Iconclass** to describe images
- Medical Subject Headings (**MeSH**)

- A skos:Concept is a "subject" to index things
  cf. an owl:Class is a set of things
- A skos:Concept may correspond both to an instance or a class
- The narrower skos:Concept can be of a different type than the broader skos:Concept
- A skos:Concept is an RDF resource and can have multiple labels
- A skos:Concept is a skos:member of a skos:Collection, or skos:inScheme a skos:Scheme

- skos:broader is more **generic** than rdfs:subClassOf
  … generic (subclass or type)
  … mereological (structural, location, membership, etc.)
  … topic implication (e.g. "cow milk" under "cows")
- skos:Concepts can have **preferred** & **alternate** labels
- skos:related is a **symmetric** relation

Formal semantics vs Social semantics

RDF, RDFS, OWL have a strict formally defined semantics
- wrt. Graphs (RDF)
- wrt. Sets (RDFS/OWL)

FOAF and others have informal semantics
- Defined in textual descriptions
- Defined in their usage online

Summary
- Triple stores are dedicated databases for RDF
  - They do indexing, statistics, joinc etc. well
  - You can sometimes access them via SPARQL endpoints
- RDF Schema is a vocabulary that builds on RDF
  - That defines som classes and properties on classes and properties
  - domain, range, subclass, subproperty
  - Restrict the possible interpretations of the world
  - Provide entailment rules

  Instance ⇒ Data
  Ontology ⇒ Schema

Lecture 7 + 8: Complex modelling and Advanced Inferencing in OWL

RDF Schema is too simple:
- We need more context to **domain** and **range**
  People live in houses, while badgers live in setts
- Properties may have **cardinality**
  The USA has only one president
- Properties may have other **characteristics**

OWL: The Web Ontology Language
- Trade-off **expressive power** and **computational efficiency**
- The **restricted language thesis**
- **Decidability** plays a central role

- Birth of **description logic**

OWL: Description Logics

- Well **understood** and very **expressive** language (really complex)
- **Concepts, properties, individuals**
- No **unique naming assumption** ⇒ Classes. instances with different names can be the same thing
- **Open World Assumption (**OWA**)** ⇒ Notting is assumed to be **true** or **false** unless it is *explicit* knowledge or *derivable* from axioms or known facts
- Concepts (classes) are interpreted as **sets**
- Axioms **restrict** the potential members of a class

Open World Assumption (OWA)

Closed World Assumption ⇒ If something is not explicitly stated to be true, it is assured to be false → Works well in closed environments

Open World Assumption ⇒ We do not make any assumptions based on the absence of statements. → Works well on the Web

OWL

- Extension of RDF Schema and syntax
- Features for **defining classes,** and **properties**
- Built on **description logics** (DL)
- Strict separation of **instances, classes** and **properties**

ADVANCED MODELLING AND INFERENCING

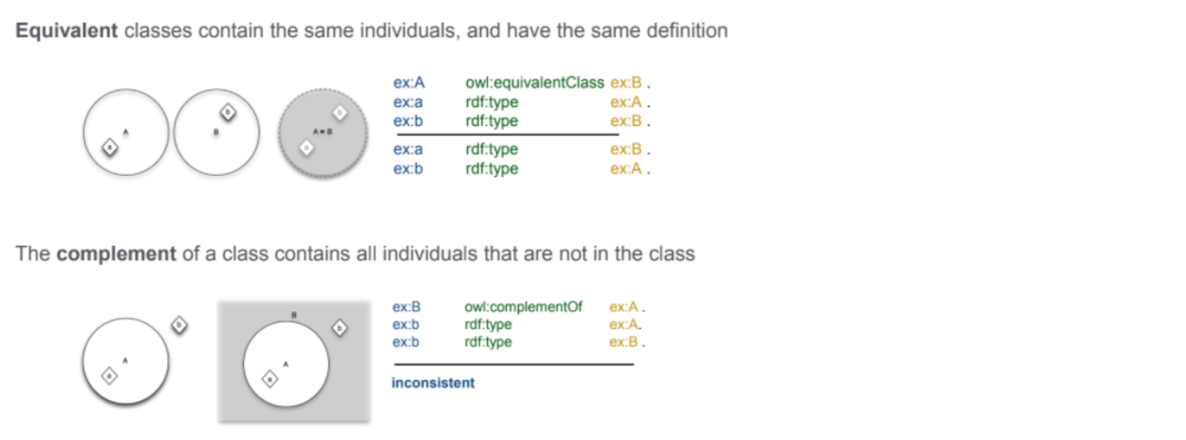
OWL CLASSES

- owl:Thing
- owl:Nothing

Class Axioms - Overview

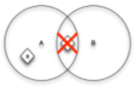
- Every class is of type owlClass (and only rdfs:Class by transitivity)
- Every class is a subclass of owl:Thing
- Every individual is of type owl:Thing (and of owl:NamedIndividual)
- No individual is of type owl:Nothing
- No class is a subclass of owl: Nothing ⇐ Used to indicate inconsistency

Equivalence and Complement



Disjointness

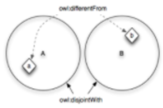
Disjoint classes do not contain the same individuals



ex:A owl:disjointWith ex:B .
ex:a rdf:type ex:A .
ex:b rdf:type ex:A, ex:B .

inconsistent

The **complement** and **disjointness** allow us to infer that individuals are different.



ex:A owl:disjointWith ex:B .
ex:a rdf:type ex:A .
ex:b rdf:type ex:B .

ex:a owl:differentFrom ex:b .

Union and Disjoint Union

The **union** contains all individuals that belong to the classes of the union

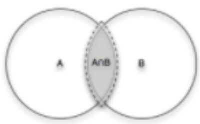
ex:C owl:unionOf (ex:A ex:B).
ex:a rdf:type ex:A .
ex:b rdf:type ex:B .

A **disjoint union** is a union of mutually **disjoint** classes

ex:Person owl:disjointUnionOf (ex:Lefthanded ex:Righthanded) .

Intersection and Enumeration

The **intersection** contains individuals that each belong to **both** classes



ex:C owl:intersectionOf (ex:A ex:B).
ex:a rdf:type ex:A .
ex:a rdf:type ex:B .

ex:a rdf:type ex:C .

You can **enumerate** all members of a class

ex:A owl:oneOf (ex:a1 ex:a2 ex:a3) .
ex:a1 rdf:type owl:Thing .

ex:a1 rdf:type ex:A .

Property Types - Object vs Data

- **Object** Properties ⇒ They have only non-literals as range
- **Datatype** Properties ⇒ Only literals as range
- **Annotation** Properties ⇒ Can't used in restrictions
- These property categories are **disjoint** and every property belongs to one of these categories

Property Types - Symmetric Property

Used to specify that a property **always** holds in both directions

if	type(p,SymmetricProperty) **and** p(x,y)
then	p(y,x)

ex:p	rdf:type	owl:SymmetricProperty
ex:x	ex:p	ex:y .

ex:y	ex:p	ex:x .

Property Types - Transitive Property

Used to specify that a property **propagates** over itself

if	type(p,TransitiveProperty) **and** p(x,y) **and** p(y,z)
then	p(x,z)

ex:p	rdf:type	owl:TransitiveProperty .
ex:x	ex:p	ex:y .
ex:y	ex:p	ex:z .

ex:x	ex:p	ex:z .

Property Types - Inverse Functional Property

Used to specify that a value for the property **uniquely identifies** an instance

if	type(p,InverseFunctionalProperty) **and** p(x,y) **and** p(z,y)
then	x = z

ex:p	rdf:type	owl:InverseFunctionalProperty .
ex:x	ex:p	ex:y .
ex:z	ex:p	ex:y .

ex:x	owl:sameAs	ex:z .

owl:sameAs allows us to specify that two URIs denote the **same** individual!

Property Types - Irreflexive Property

Used to specify that **no individual** is **ever** related to itself via that propert

if	type(p,IrreflexiveProperty) p(x,y)
then	p(x,x) **is inconsistent**

ex:p	rdf:type	owl:IrreflexiveProperty .
ex:x1	ex:p	ex:x2 .

ex:x1	owl:differentFrom	ex:x2 .

owl:differentFrom allows us to specify that two URIs denote a **different** individual!

Property Types - Asymmetric Property

Used to specify that a property **never** holds in both directions

if	type(p,AsymmetricProperty) **and** p(x,y)
then	p(y,x) **is inconsistent**

ex:p	rdf:type	owl:AsymmetricProperty .
ex:x1	ex:p	ex:y .
ex:y	ex:p	ex:x2 .

ex:x1	owl:differentFrom	ex:x2 .

owl:differentFrom allows us to specify that two URIs denote a **different** individual!

Property Types - Functional Property

Used to specify that a property has **only one** value for any particular instance

if	type(p,FunctionalProperty) **and** p(x,y) **and** p(x,z)
then	y = z

ex:p	rdf:type	owl:FunctionalProperty .
ex:x	ex:p	ex:y .
ex:x	ex:p	ex:z .

ex:y	owl:sameAs	ex:z .

owl:sameAs allows us to specify that two URIs denote the **same** individual!

Property Types - Reflexive Property

Used to specify that **every individual** is **always** related to **itself** by that property

if	type(p,ReflexiveProperty)
then	$\forall$x p(x,x)

ex:p	rdf:type	owl:ReflexiveProperty .

ex:x	ex:p	ex:x .

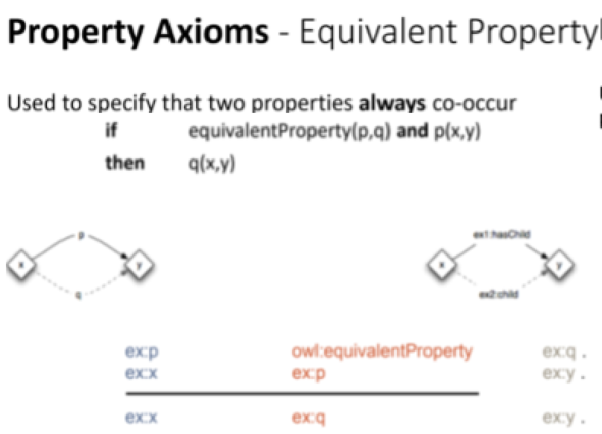
Property Axioms - Inverse Property

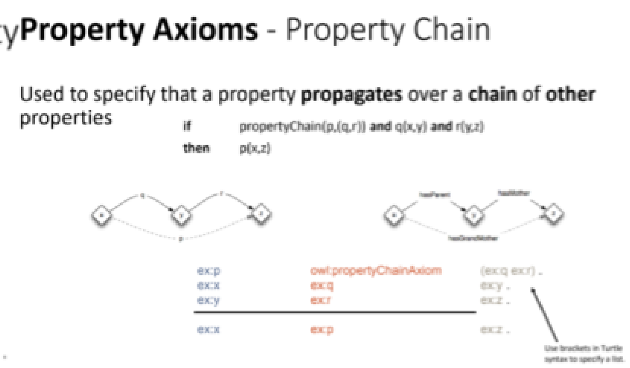
Used to specify that one property is **always** the inverse of another property

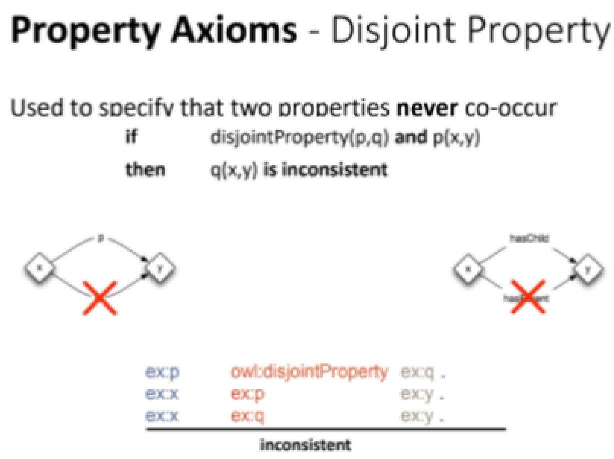
if	inverseOf(p,q) **and** p(x,y)
then	q(y,x)

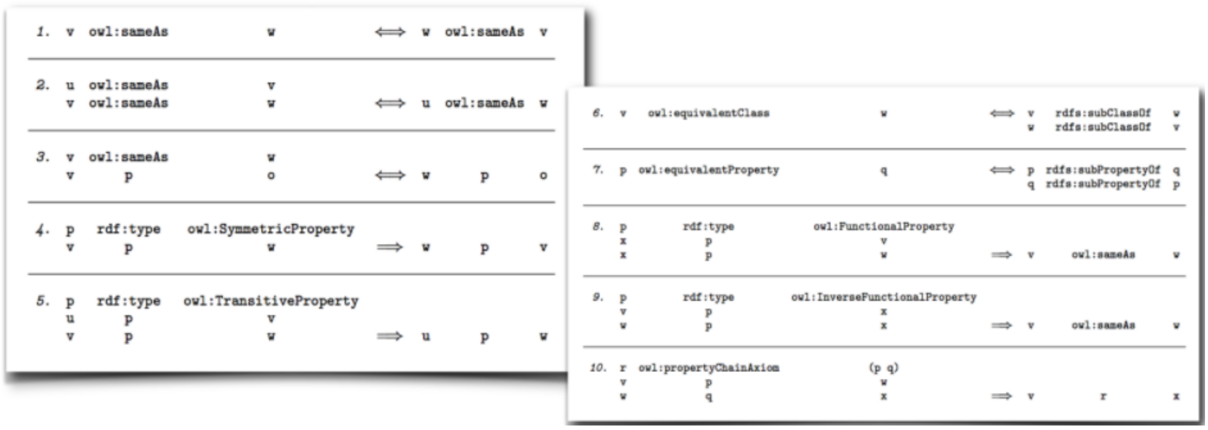
ex:p	owl:inverseOf	ex:q .
ex:x	ex:p	ex:y .

ex:y	ex:q	ex:x .









1.	v	owl:sameAs	w	⟺	w	owl:sameAs	v
2.	u	owl:sameAs	v				
	v	owl:sameAs	w	⟺	u	owl:sameAs	w
3.	v	owl:sameAs	w				
	v	p	o	⟺	w	p	o
4.	p	rdf:type	owl:SymmetricProperty				
	v	p	w	⟹	w	p	v
5.	p	rdf:type	owl:TransitiveProperty				
	u	p	v				
	v	p	w	⟹	u	p	w
6.	v	owl:equivalentClass	w	⟺	v	rdfs:subClassOf	w
					w	rdfs:subClassOf	v
7.	p	owl:equivalentProperty	q	⟺	p	rdfs:subPropertyOf	q
					q	rdfs:subPropertyOf	p
8.	p	rdf:type	owl:FunctionalProperty				
	x	p	v				
	x	p	w	⟹	v	owl:sameAs	w
9.	p	rdf:type	owl:InverseFunctionalProperty				
	v	p	x				
	w	p	x	⟹	v	owl:sameAs	w
10.	r	owl:propertyChainAxiom	(p q)				
	v	p	w				
	w	q	x	⟹	v	r	x

Lecture 9: Ontology Engineering, Ontology Alignment & Data Integrations

Ontology: “An ontology is an **explicit** specification of a **shared conceptualisation** that holds in a particular **context**”

Ontology Engineering concerns the **practical aspects** of building and using ontologies.

- **Methodologies** for building an ontology
- **Types** of ontologies
- Strategies for **reusing** ontologies

Benefits of ontologies

- Communication between people
- Interoperability between software agents

- Reuse of domain knowledge, make domain knowledge explicit, analyse domain knowledge

Ontological Engineering

- When building a **large ontology** it makes sense to follow a **methodology**
- Make sure to have a proper **scope, maximal agreement** and **minimal ontological commitment**.
- Ontological commitment also concerns the **expressiveness** of the language you decide to use: how **strict** do you want your semantics to be?

Ontological Commitment

- Each statement in the ontology is a **commitment** to a view of the domain
- **Over-commitment** means that the ontology makes too strong a statement
- Ontologies live in an **open** distributed world. OO models in a **closed** world. *… danger of over-commitment is much larger*
- **Rule of thumb:** choose the minimal ontological commitment needed

Ingredients of ontological engineering

- Classes (concepts) and their hierarchy
- Properties (attributes & relations) and their hierarchy
- Class axioms (disjointness, equality)
- Class restrictions (universal, existential, cardinality)
- Property types (domain/range, functional, transitive)
- Instances (individuals)

Methodology

- **Top down:** start with classes, properties of your domain, work your way down to instance level
- **Bottom up:** start with existing material (data, use cases,...)
- **Middle-out:** start with fundamental concepts, work towards more abstract and more specific terms

Ontology development

1. **Determine domain & scope**
2. **Consider reuse** ⇒ There is almost always an ontology available from a third party that provides at least a useful starting point for your own ontology.
3. **Enumerate terms**

- List all relevant terms
    - **Nouns** and noun phrases form the basis for **class** names
    - **Verbs** or verb phrases form the basis for **property** names
- Traditional **knowledge engineering** tools deliver: the set of terms and their initial structure

4. **Define taxonomy**

- Relevant terms are organised in a taxonomic hierarchy
- Ensure that the hierarchy is indeed a taxonomy

5. **Define properties**

- When defining properties, it makes sense to immediately think of **domain** and **range** of the property

- There is a methodological tension between genericity and flexibility(=inheritance of subclasses)

6. **Define classes and their properties**

- Does the taxonomy need revising?

7. **Define instances**

- The number of instances is much higher than the number of classes thus **ontology population** is usually not done manually but retrieved from legacy data sources

8. **Check for anomalies**

- **advantage** of OWL over RDFS is the possibility to detect inconsistencies
- Examples of common inconsistencies:
- *Incompatible domain and range definitions for symmetric, transitive or inverse properties*
- *Cardinality restrictions*
- *Requirements on property values can conflict with domain & range*

How to **integrate** your own data with that of others - **Alignment and Reuse**

“An ontology should require the **minimal ontological commitment** sufficient to support the intended knowledge sharing activities”	Type	Language
	Thesaurus/Taxonomy Vocabulary Ontology	SKOS RDF Schema OWL

SKOS - RDFS Vocabulary for Thesaurus Modeling
- skos:Concept is a “subject” to index things → an owl:Class is a set of things
- skos:Concept may correspond both to an instance or a class
- The narrower skos:Concept can be of a different type than the broader skos:Concept
- skos:broader is more **generic** than rdfs:subClassOf
    - generic (subclass or type)
    - mereological (structural, location, memberships)
    - topic implication
- skos:Concepts can have **preferred** & **alternate** labels
- skos:related is a **symmetric** relation

Aligning ontologies is an integral part of **data integration**

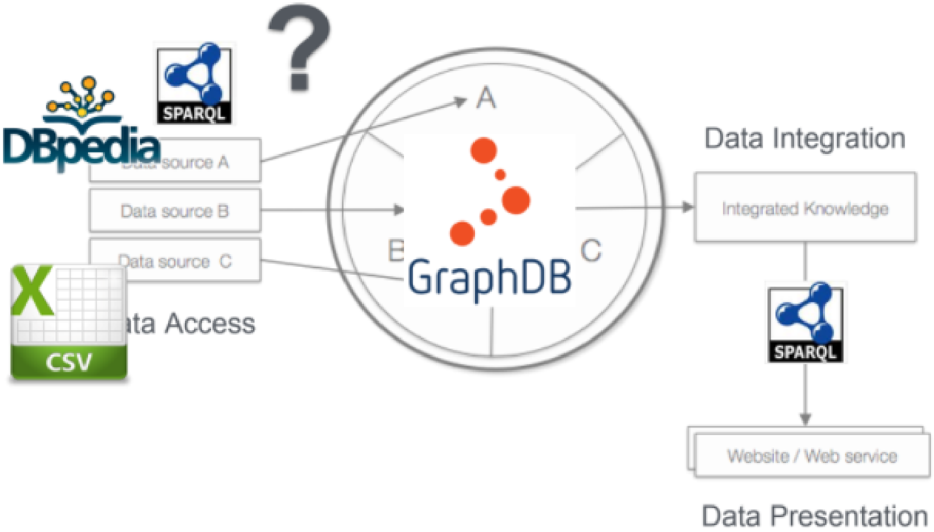
Ontology **Merging & Alignment**	
- Find links between concepts in a **source** and a **target** ontology - **Benefit** from knowledge encoded in other ontology - Enable **access** across applications or collections	 Ontology **alignment** is the process of semantic **mapping** between **classes** and **properties** of two or more ontologies.  Ontology **merging** is the **integration** of two or more ontologies into a **new** coherent ontology. … solution depends on **ontology type**

Mapping & Alignment - Evaluation
- Produced mappings always have a **degree of confidence**
- Judge **individual links** … precision
- Compare to a **reference alignment**

- Compare the **logical entailments** of the model
- **End-to-end** evaluation by using the alignment in an application

Lecture 10: Data integration

Engineering & Integration ⇒ Learning how to **integrate** your own data with that of others
Architecture - Semantic Web Application



Typically **building an ontology** (the classes and properties) is **manual** work.
Typically **populating** the ontology with **instances** is **semi-automatic work**.

The program
Build ontology in Protege → Import ontology in GraphDB → Get some interesting data in CSV → Convert to triples using OntoRefine → Find potential links in DBPedia → Link your data using SPARQL (import data) → Try out interesting SPARQL queries

Publish Linked Data - Strategies
- Integrate it in an **application** (not programmatically accessible)
- Expose it via **URI dereferencing** (Truly **web-based**, limited expressiveness)
- Expose it via a **RESTful API** (Limited expressiveness, does not return RDF)
- Expose it via a **SPARQL endpoint** (**Costly** because of uncontrollable server load)

Data integration summary
- If needed, transform external information into RDF
- Try to maintain the original schema as much as possible
- Use inferencing to derive mappings where possible
- Be careful to choose the best architecture for your application
- Publish your data in one of various ways