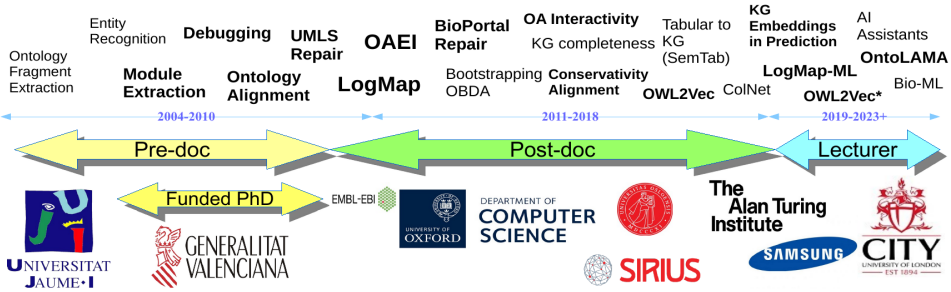
Introduction to Knowledge Graphs

Ernesto Jiménez-Ruiz
Lecturer in Artificial Intelligence

Before we start...

Research journey



- 2019-: Lecturer in Artificial Intelligence.
  City, University of London
  - Main topic: **Neural-Symbolic integration**
- 2021-: Organiser of the **Alan Turing Institute Interest group on Knowledge Graphs**
  - https://www.turing.ac.uk/research/interest-groups/knowledge-graphs

Contact

- **Ernesto Jiménez-Ruiz**
- City, University of London
- Contact:
  - ernesto.jimenez-ruiz@city.ac.uk
  - https://www.city.ac.uk/people/academics/ernesto-jimenez-ruiz

GitHub Repository

https://github.com/city-knowledge-graphs/phd-course-uji

Agenda

- Four sessions split into two days
- **Morning sessions**:
  - Theory: 9:30-11:00
  - Break 30 min
  - Hands-on: 11:30-13:00
- Lunch break (1hour)
- **Afternoon sessions**:
  - Theory: 14:00-15:30
  - Break 30 min
  - Hands-on: 16:00-17:30

Project submission

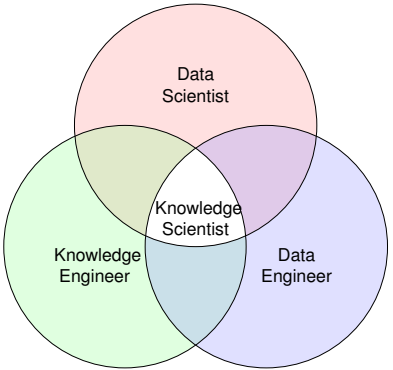
- Students need to work on a **project** (max 2 students per group). There are two options:
  - Create a (simple) system that performs KG to KG alignment.
  - Create a (simple) system that performs CSV to KG matching.
- Submission:
  - **When:** June 30, 23:59 CEST
  - **What:** a link to the GitHub repository where the system codes are
  - **How:** via a Google form (see GitHub)

Course Organization

1. Introduction to Knowledge Graphs
   - Lab: Creation of a small knowledge graph and ontology.
2. Reasoning and Querying with Knowledge Graphs
   - Lab: First steps with the SPARQL query language.
3. Matching: KG-to-KG and CSV-to-KG
   - Lab: Creation of a (simple) matching system.
4. Knowledge Graphs and Language Models
   - Lab: Ontology Embeddings with OWL2Vec*.

Skills to be gained

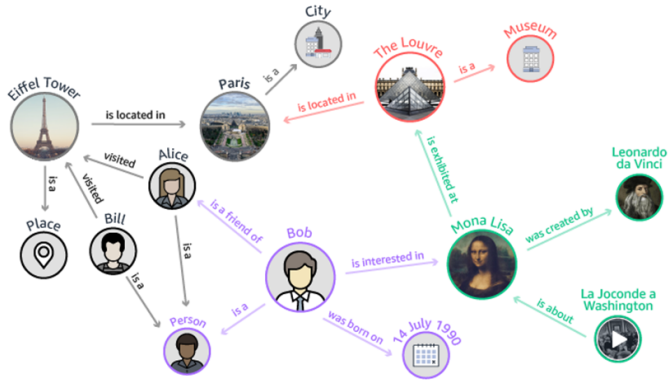
- **Data Engineer:** harnesses and collects data.
- **Data Scientist:** draws value from data.
- **Knowledge Engineer:** encodes domain expertise.
- **Knowledge Scientist:** adds context to the data to make it more useful, clean, reliable and ready to be used.



George Fletcher and others. Knowledge Scientists: Unlocking the data-driven organization. 2020

About Knowledge Graphs (i)

– **(data graph)** capture information about entities of interest (like people, places or events), and
– **(knowledge)** with an underlying formal model



Knowledge Graphs: `https://arxiv.org/abs/2003.02320`.

About Knowledge Graphs (ii)

- In data science and AI, knowledge graphs are commonly used to:
  - Serve as bridges between humans and systems;
  - Facilitate access to and integration of data sources;
  - Represent enterprise data to drive products and make them more “intelligent”; and
  - Add context and depth to other, more data-driven AI techniques such as machine learning (*e.g.*, **Neurosymbolic AI**)

Interest Group @ The Alan Turing Institute: `https://www.turing.ac.uk/research/interest-groups/knowledge-graphs`

Why Ontologies and KGs? (Experiences from Industry)

– Knowledge graphs are **not complex**: they are the simplest way to layer human expertise onto stored data.

Juan Sequeda and Ora Lassila. **Designing and Building Enterprise Knowledge Graphs**. 2021
Ora Lassila. **On the broad applicability of Semantic Web technologies**. (COST DKG Talk Series. 2021
`https://www.youtube.com/watch?v=f9wautaqWUs`
Juan Sequeda and Ora Lassila. `https://watch.knowledgegraph.tech/knowledge-espresso/videos/knowledge-espresso-with-ora-lassila-and-juan-sequeda`

Why Ontologies and KGs? (Experiences from Industry)

- Knowledge graphs are **not complex**: they are the simplest way to layer human expertise onto stored data.
- Investing in knowledge graphs is **resilient**: an initial investment that pays off by allowing companies to pre-prepare for unanticipated and unfolding use cases

Juan Sequeda and Ora Lassila. **Designing and Building Enterprise Knowledge Graphs**. 2021
Ora Lassila. **On the broad applicability of Semantic Web technologies**. (COST DKG Talk Series. 2021
`https://www.youtube.com/watch?v=f9wautaqWUs`
Juan Sequeda and Ora Lassila. `https://watch.knowledgegraph.tech/knowledge-espresso/videos/knowledge-espresso-with-ora-lassila-and-juan-sequeda`

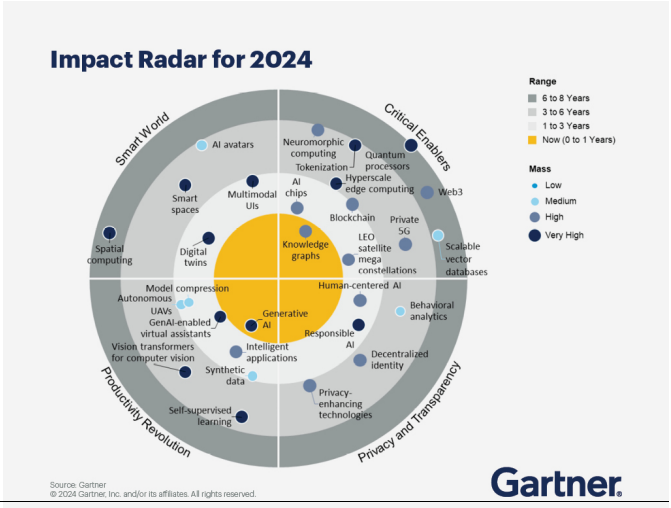
Why Ontologies and KGs? (Experiences from Industry)

- Knowledge graphs are **not complex**: they are the simplest way to layer human expertise onto stored data.
- Investing in knowledge graphs is **resilient**: an initial investment that pays off by allowing companies to pre-prepare for unanticipated and unfolding use cases
- **Unifying logical representation** for data and semantics to get out the data science enterprise mess.

Juan Sequeda and Ora Lassila. **Designing and Building Enterprise Knowledge Graphs**. 2021
Ora Lassila. **On the broad applicability of Semantic Web technologies**. (COST DKG Talk Series. 2021
`https://www.youtube.com/watch?v=f9wautaqWUs`
Juan Sequeda and Ora Lassila. `https://watch.knowledgegraph.tech/knowledge-espresso/videos/knowledge-espresso-with-ora-lassila-and-juan-sequeda`

Knowledge Graphs according to Gartner

30 Emerging Technologies That Will Guide Your Business Decisions:
Gartner Web



PART I: RDF-based Knowledge Graphs

Graph Data Models

Data Graphs

- **Foundation** of any Knowledge Graph
- **Intuitive abstractions** to capture entities and their relationships.
- **Do not require schema** to start adding data.
- **Flexibility** to assemble data from multiple sources.

Data Graphs

- **Foundation** of any Knowledge Graph
- **Intuitive abstractions** to capture entities and their relationships.
- **Do not require schema** to start adding data.
- **Flexibility** to assemble data from multiple sources.
- Enable **queries over paths** of arbitrary length.
- Allow the use of **graph analytics** techniques.
- (Potential) correspondence with **logical** unary and binary **predicates**.

Types of graph-structured data models (i)

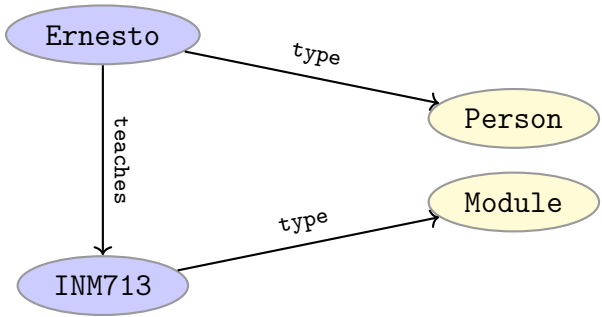
- Directed edge-labelled (multi)graphs (multi-relational graphs)
- Property graphs
- Heterogeneous graphs (heterogeneous information within a node)
- Hypergraphs (edges connecting set of nodes)
- Hypernodes (nested graphs in a node)

Types of graph-structured data models (i)

- **Directed edge-labelled (multi)graphs** (multi-relational graphs)
- **Property graphs**
- Heterogeneous graphs (heterogeneous information within a node)
- Hypergraphs (edges connecting set of nodes)
- Hypernodes (nested graphs in a node)

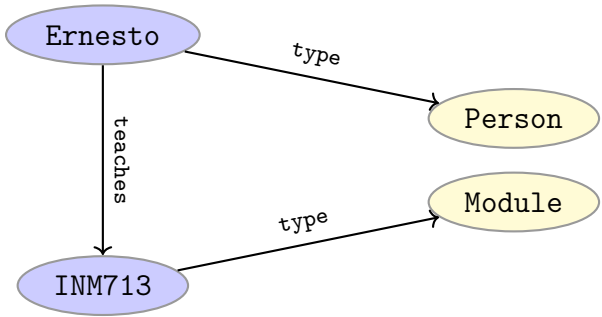
Types of graph-structured data models (ii)

Directed edge-labelled (multi)graphs.



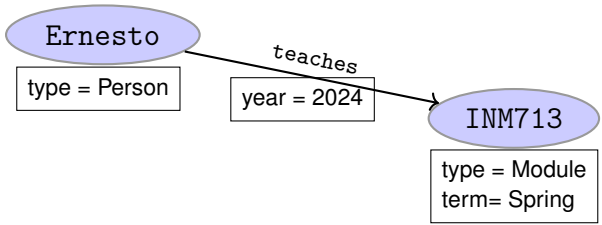
Types of graph-structured data models (ii)

Directed edge-labelled (multi)graphs. (In this course).



Types of graph-structured data models (iii)

Property graphs



RDF: Resource Description Framework

RDF in a nutshell

- Resource Description Framework (RDF)
- A **standardised data model** based on the directed edge-labelled graph model.
  - **Nodes**: Internationalised Resource Identifiers (IRIs), literals, and blank nodes (nodes without identifier).
  - **Edges**: IRIs
- **W3C recommendation**.
- Conceptual **modelling of resources**.

RDF in a nutshell

- Resource Description Framework (RDF)
- A **standardised data model** based on the directed edge-labelled graph model.
  - **Nodes**: Internationalised Resource Identifiers (IRIs), literals, and blank nodes (nodes without identifier).
  - **Edges**: IRIs
- **W3C recommendation**.
- Conceptual **modelling of resources**.
- In this course we will study **RDF-based Knowledge Graphs**.

All you need are triples

RDF Triples

- All information in RDF is expressed using a *triple* pattern.
- A triple consists of a subject, a predicate, and an object.
  Examples:

subject	predicate	object
England	has capital	London
England	has king	Charles III
Charles III	born year	1948

RDF Triples

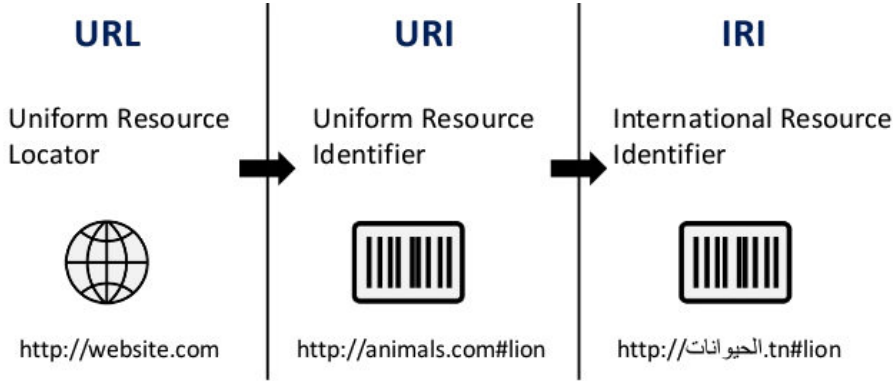
- All information in RDF is expressed using a *triple* pattern.
- A triple consists of a subject, a predicate, and an object.
  Examples:

subject	predicate	object
England	has capital	London
England	has king	Charles III
Charles III	born year	1948

- Another word for an RDF triple is a *statement* or *fact*.
- The elements of an RDF triple are either: *URI references*, *literals*, *blank nodes*.

All you need are triples... and URIs

Identifying resources



Identifying resources in RDF

- RDF talks about *resources*.
- Resources are identified by (unique) IRIs/URIs.

Identifying resources in RDF

- RDF talks about *resources*.
- Resources are identified by (unique) IRIs/URIs.
- E.g., in `dbpedia.org` KG graph:

England:	`http://dbpedia.org/resource/England`
has capital:	`http://dbpedia.org/ontology/capital`
London:	`http://dbpedia.org/resource/London`
has king:	`http://dbpedia.org/ontology/monarch`
Charles III:	`http://dbpedia.org/resource/Charles,_Prince_of_Wales`

Identifying resources in RDF

- RDF talks about *resources*.
- Resources are identified by (unique) IRIs/URIs.
- E.g., in `dbpedia.org` KG graph:

England:	`http://dbpedia.org/resource/England`
has capital:	`http://dbpedia.org/ontology/capital`
London:	`http://dbpedia.org/resource/London`
has king:	`http://dbpedia.org/ontology/monarch`
Charles III:	`http://dbpedia.org/resource/Charles,_Prince_of_Wales`

- URIs/IRIs are not necessarily dereferenceable; but they could be: *Web of Data*.

Building an URI

`scheme:[//authority]path[?query][#fragment]` (in brackets optional elements)

- scheme: `http`, `https`, `ftp`, `file`, etc.
- authority: typically the domain, `//www.city.ac.uk`
- path: `/sst/cs/academics`
- query: additional information for a Web service (not used in our URIs)
- fragment: to address a sub-part of a retrieved resource, `#ernesto`

e.g., `http://www.city.ac.uk/sst/cs/academics#ernesto`

Pascal Hitzler, Markus Krotzsch, Sebastian Rudolph. Foundations of Semantic Web Technologies. CRC Press 2009.

Identifying resources: URIs and (local) CURIs

- **URIs are often long** and hard to read and write.
- Most serialisations use an **abbreviation** mechanism.
  - Define “prefixes”, “namespaces”.
  - RDF/XML format: XML namespaces and entities.

Identifying resources: URIs and (local) CURIs

- **URIs are often long** and hard to read and write.
- Most serialisations use an **abbreviation** mechanism.
  - Define “prefixes”, “namespaces”.
  - RDF/XML format: XML namespaces and entities.
- E.g., in Turtle serialisation:

```
@prefix dbr:  <http://dbpedia.org/resource/> .
@prefix dbo:  <http://dbpedia.org/ontology/> .
```

- A *CURI* (Compact URI) or *QName* like ‘`dbr:London`’ stands for `http://dbpedia.org/resource/London`

URIs, QNames and data value

– We can then state that England's capital is London as:

```
<http://dbpedia.org/resource/England
<http://dbpedia.org/ontology/capital>
<http://dbpedia.org/resource/London> .
```

– Or using prefixes:

```
dbr:England  dbo:capital  dbr:London .
```

URIs, QNames and data value

– We can then state that England's capital is London as:

```
<http://dbpedia.org/resource/England
<http://dbpedia.org/ontology/capital>
<http://dbpedia.org/resource/London> .
```

– Or using prefixes:

```
dbr:England  dbo:capital  dbr:London .
```

– But what if we want to state that London's population is **9,304,000**?

– We cannot have one URI for every integer, decimal number, string, etc.

All you need are triples... and URIs... and literals

Literals in RDF

- Literals are used to represent **data values**.
- A literal in a RDF consist of these elements:
  - A lexical form: “London”, “9,304,000”, “Londres”
  - A (supported) datatype IRI (from OWL, RDF or XML Schema datatypes): *e.g.*, ^^xsd:integer
  - A language tag (*e.g.*, “en”, “es”)

Literals in RDF

- Literals are used to represent **data values**.
- A literal in a RDF consist of these elements:
  - A lexical form: “London”, “9,304,000”, “Londres”
  - A (supported) datatype IRI (from OWL, RDF or XML Schema datatypes): *e.g.*, ^^xsd:integer
  - A language tag (*e.g.*, “en”, “es”)
- Examples:

```
dbr:London dbo:population "9,304,000"^^xsd:integer .
dbr:London rdfs:label "London"^^xsd:string .
dbr:London rdfs:label "Londres"@es .
```

RDF Graph = set of triples with URIs and literals (and blank nodes)

Example RDF-based (Knowledge) Graph

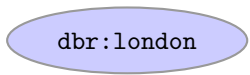
London is a city in England called Londres in Spanish

```
dbr:london rdf:type dbo:City .
dbr:london dbo:locationCountry dbr:england .
dbr:london rdfs:label "Londres"@es .
```

Example RDF-based (Knowledge) Graph

London is a city in England called Londres in Spanish

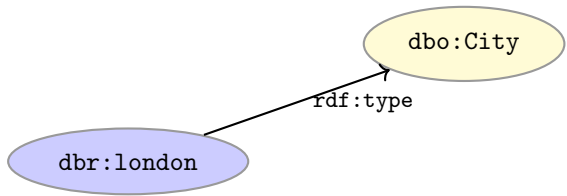
```
dbr:london rdf:type dbo:City .
dbr:london dbo:locationCountry dbr:england .
dbr:london rdfs:label "Londres"@es .
```



Example RDF-based (Knowledge) Graph

London is a city in England called Londres in Spanish

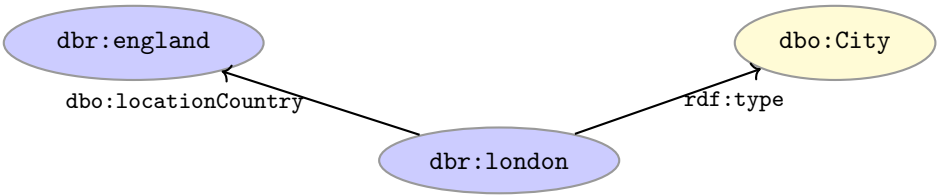
```
dbr:london rdf:type dbo:City .
dbr:london dbo:locationCountry dbr:england .
dbr:london rdfs:label "Londres"@es .
```



Example RDF-based (Knowledge) Graph

London is a city in England called Londres in Spanish

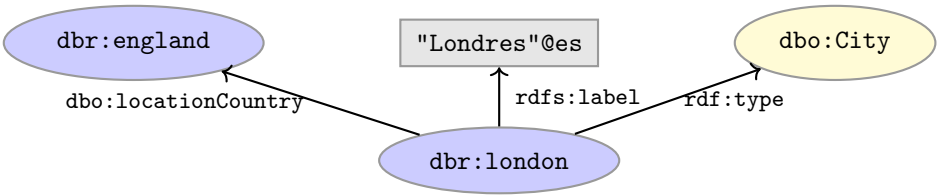
```
dbr:london rdf:type dbo:City .
dbr:london dbo:locationCountry dbr:england .
dbr:london rdfs:label "Londres"@es .
```



Example RDF-based (Knowledge) Graph

London is a city in England called Londres in Spanish

```
dbr:london rdf:type dbo:City .
dbr:london dbo:locationCountry dbr:england .
dbr:london rdfs:label "Londres"@es .
```

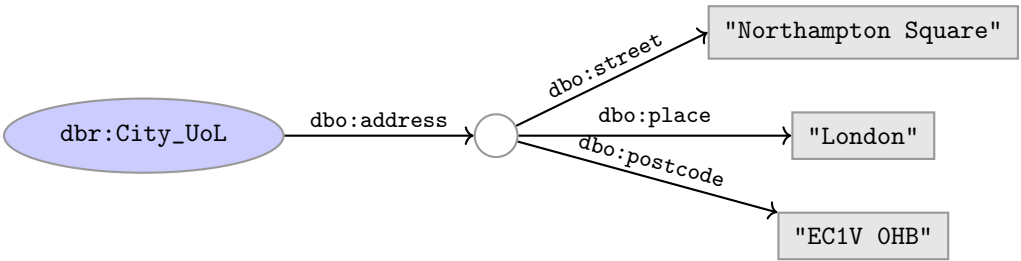


RDF Named Graphs

- We can **refer to a set of triples** with a name (*i.e.*, URI)
- Why?
  - **Independent** sources: 1 file, 1 graph.
  - Add specific **context** to a set of triples (*e.g.*, provenance).
  - Allowing to **query** a specific set of triples.
- Triples in a named graph sometimes referred as **quads**.

RDF Graphs: Blank nodes

- Blank nodes are like resources without a URI.
- Use when resource is unknown, or has no (natural) identifier. *e.g.*,:



RDF Triple Grammar

– Literals and blank nodes may not appear everywhere in triples:

	s	p	o
• URI references may occur in all positions	✔	✔	✔
• Literals may only occur in object position	✘	✘	✔
• Blank nodes can not occur in predicate position	✔	✘	✔

RDF Triple Grammar

– Literals and blank nodes may not appear everywhere in triples:

	s	p	o
• URI references may occur in all positions	✔	✔	✔
• Literals may only occur in object position	✘	✘	✔
• Blank nodes can not occur in predicate position	✔	✘	✔

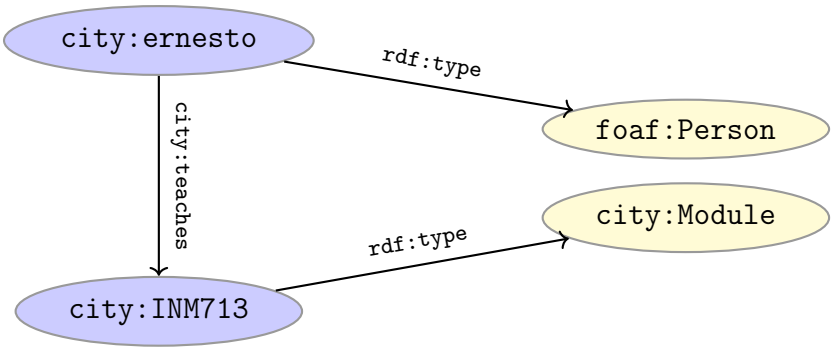
– Why?
  – Literals are just values, no relationships from literals allowed.
  – Blank nodes in predicate position deemed “too meaningless” and confusing.

RDF and Property Graphs

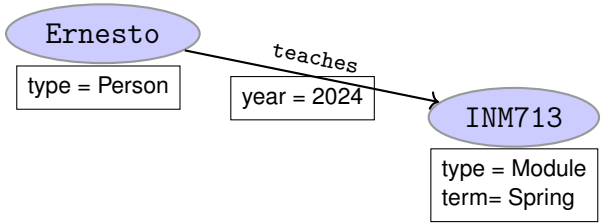
Annotating statements/triples in RDF?

How to encode that **“Ernesto teaches INM713 in 2024”**?

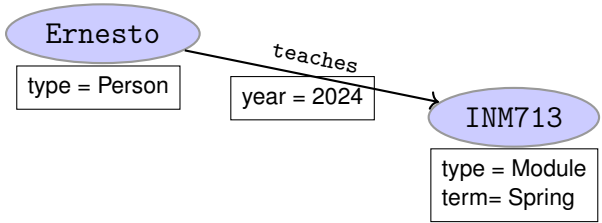
(*i.e.*, **Metadata of a triple**: provenance, beliefs, uncertainty, creator, time, etc.)



Using property Graphs

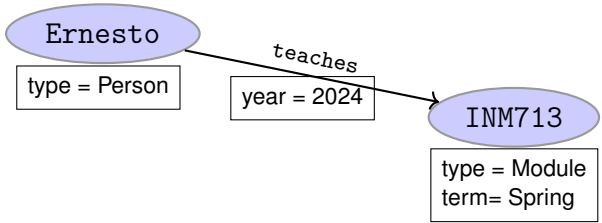


Using property Graphs



But can we do it in RDF?

Using property Graphs



But can we do it in RDF? Yes!

Annotating statements/triples in RDF (i)

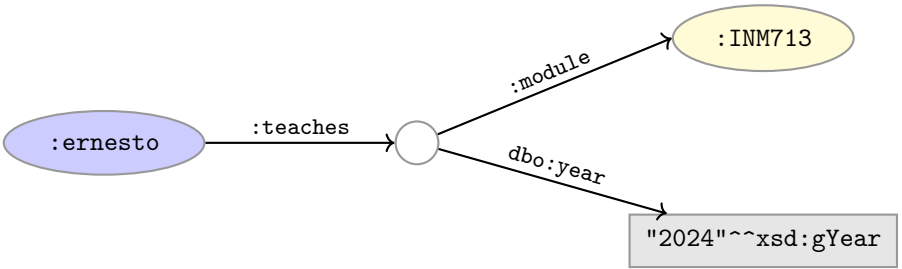
RDF Solution 1: **Named Graphs**

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dbo: <http://dbpedia.org/ontology/>
@prefix : <http://www.example.org/university/london/city#> .

:AcademicYear2024 {
   :ernesto :teaches :INM713 .
   :INM713 rdf:type :Module .
   :INM713 :module_type "Elective"
}
:AcademicYear2024 dbo:year "2024"^^xsd:gYear .
```

Annotating statements/triples in RDF (ii)

RDF Solution 2: **n-ary relationships** (and blank-nodes)



Annotating statements/triples in RDF (iii)

RDF Solution 3: **Reification**

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dbo: <http://dbpedia.org/ontology/>
@prefix : <http://www.example.org/university/london/city#> .

:ernesto :teaches :INM713 .
_:s rdf:type rdf:Statement
    rdf:subject :ernesto;
    rdf:predicate :teaches ;
    rdf:object :INM713 ;
    dbo:year "2024"^^xsd:gYear .
:aidan :likes _:s .
```

Annotating statements/triples in RDF (iv)

RDF Solution 4: **RDF*** or **RDF-Star** (soon a W3C recommendation)

- Uses triple operator « »
- Compact solution: less triples, less space, faster to load
- Closer to Property Graphs

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dbo: <http://dbpedia.org/ontology/>
@prefix : <http://www.example.org/university/london/city#> .

:ernesto :teaches :INM713 .
<<:ernesto :teaches :INM713>> dbo:year "2024"^^xsd:gYear .
```

O. Hartig and B. Thompson. Foundations of an Alternative Approach to Reification in RDF. https://arxiv.org/abs/1406.3399.
Latest information: https://w3c.github.io/rdf-star/

RDF and Property Graphs

- RDF workarounds/extensions to annotate triples.
- Is this a limitation?

RDF and Property Graphs

- RDF workarounds/extensions to annotate triples.
- Is this a limitation?
- More a **feature** due to its relation to **(logic-based) unary and binary predicates**.

RDF and Property Graphs

- RDF workarounds/extensions to annotate triples.
- Is this a limitation?
- More a **feature** due to its relation to **(logic-based) unary and binary predicates**.
- **(OWL-layered) RDF-based KGs have clear semantics**.
- Property graphs do not have the same semantic commitment.

PART II: OWL-based Knowledge Graphs

OWL

- Acronym for ***The Web Ontology Language***.
- A **W3C recommendation**:
  - OWL 1 (2004): `http://www.w3.org/TR/owl-ref/`
  - OWL 2 (2009): `https://www.w3.org/TR/owl2-overview/`

OWL

- Acronym for ***The Web Ontology Language***.
- A **W3C recommendation**:
  - OWL 1 (2004): `http://www.w3.org/TR/owl-ref/`
  - OWL 2 (2009): `https://www.w3.org/TR/owl2-overview/`
- Built on **Description Logics (DL)**.
  - OWL 1: $\mathcal{SHOIN}$(D)
  - OWL 2: $\mathcal{SROIQ}$(D)
- Combines **DL expressiveness with RDF technology**
  - OWL-layered RDF-based knowledge graphs or just **OWL-based KGs**

Description Logics and OWL

– **Origin:** semantic networks and other graph-based models and the attempt to formalise them with First Order Logic (FOL).

Description Logics and OWL

- **Origin:** semantic networks and other graph-based models and the attempt to formalise them with First Order Logic (FOL).
- Core reasoning problems in **FOL** are **undecidable**.
- **Trade-off** between **expressiveness** and computational properties

Description Logics and OWL

- **Origin:** semantic networks and other graph-based models and the attempt to formalise them with First Order Logic (FOL).
- Core reasoning problems in **FOL** are **undecidable**.
- **Trade-off** between **expressiveness** and computational properties
- **Description Logics** (DL):
  - Family of knowledge representation languages
  - Decidable subset of FOL
  - Original called: *Terminological language* or *concept language*
  - OWL is based on DL

OWL 1, OWL 2 (profiles) and RDFS

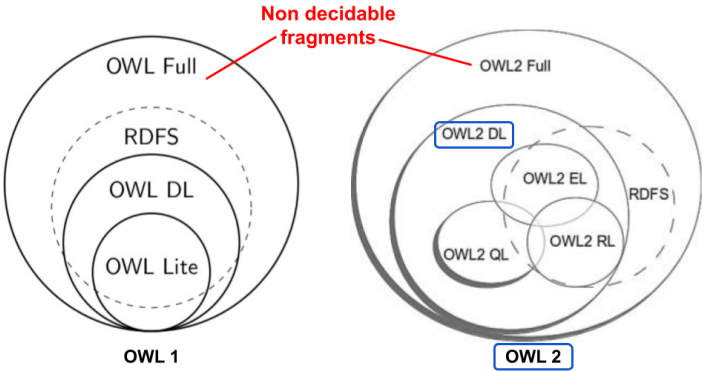
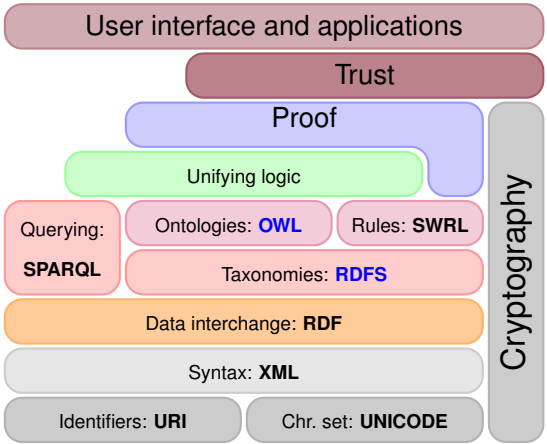


Image adapted from Olivier Cure and Guillaume Blin. RDF Database Systems (Chapter 3). 2015. Elsevier.

Semantic Web Technology Stack



RDF, RDFS and OWL

– RDF imposes a basic grammar. A triple consists of *subject*, *predicate*, and *object*

– But one could still define:

```
dbr:london rdf:type "some string"^^xsd:string .
```

RDF, RDFS and OWL

- RDF imposes a basic grammar. A triple consists of *subject*, *predicate*, and *object*
- But one could still define:
  `dbr:london rdf:type "some string"^^xsd:string .`
- RDF Schema (RDFS) extends the **grammar** for the **"expected"** triples (*e.g.*, `rdf:type rdf:range rdfs:Resource .`), extends the vocabulary, and include a set of **inference rules**.

RDF, RDFS and OWL

- RDF imposes a basic grammar. A triple consists of *subject*, *predicate*, and *object*
- But one could still define:
  `dbr:london rdf:type "some string"^^xsd:string .`
- RDF Schema (RDFS) extends the **grammar** for the **"expected"** triples (*e.g.*, `rdf:type rdf:range rdfs:Resource .`), extends the vocabulary, and include a set of **inference rules**.
- RDFS was the first attempt of a language to formalize simple taxonomic ontologies. **OWL** builds on top of RDFS and provide means to **validate RDF data**.

Modelling with OWL 2: What is an Ontology?

Ontologies (information sciences)

- Core idea of **knowledge graphs** is the **enhancement of the graph data model** with...
  - "...a **formal specifications** of a shared domain conceptualization"
  - "...an **abstract symbolic representations** of a domain expressed in a formal language"
  - ...or at least **some sort of schema** specifying how things in the graph are expected to connect to each other.

Thomas R. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. 1993
Pim Borst, Hans Akkermans, and Jan Top. Engineering ontologies. 1999.

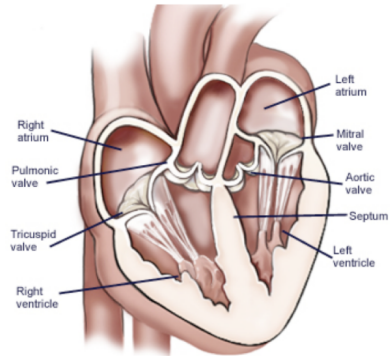
Ontologies as domain models (i)

- A model is a **simplified** (abstract) **representation** of certain aspects of the real world.
- Models help people **communicate**.
- Models explain and **make predictions**.
- Models **mediate** among multiple viewpoints.

Dean Allemang, James Hendler. Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL.

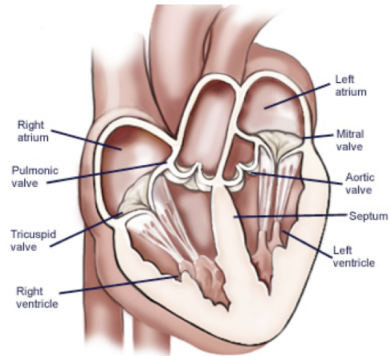
Ontologies as domain models (ii)

- include **vocabulary** relevant to a domain (*e.g.*, with RDF and IRIs)
- specify meaning (**semantics**) of terms (*e.g.*, with OWL)
  - Heart is a muscular organ that is part of the circulatory system



Ontologies as domain models (ii)

- include **vocabulary** relevant to a domain (*e.g.*, with RDF and IRIs)
- specify meaning (**semantics**) of terms (*e.g.*, with OWL)
  - Heart is a muscular organ that is part of the circulatory system
- are **formalised** using a suitable logic language (*e.g.*, with OWL)
  - Heart SUBCLASSOF MuscularOrgan AND (isPartOf SOME CirculatorySystem)



Some Ontology Axioms as RDF triples (i)

A KG with a small Ontology:

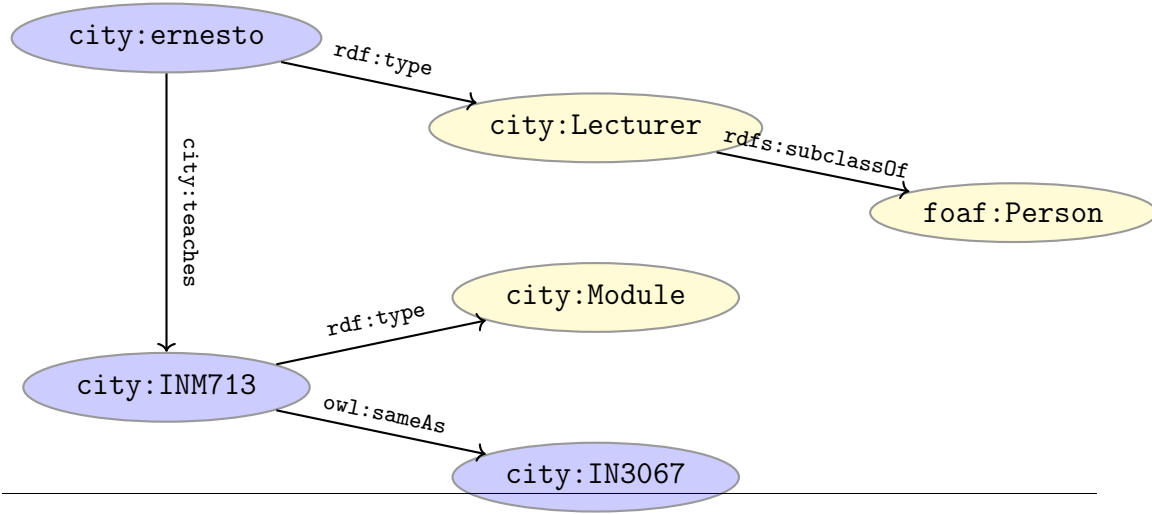
```
#Data
city:ernesto city:teaches city:INM713 .
city:IN3067 owl:sameAs city:INM713 .

#Linking data to ontology
city:INM713 rdf:type city:Module .
city:ernesto rdf:type city:Lecturer .

#Ontology
city:Lecturer rdfs:subClassOf foad:Person .
```

(*) OWL Ontology axioms may not have a 1-to-1 triple representations

Some Ontology Axioms as RDF triples (ii)

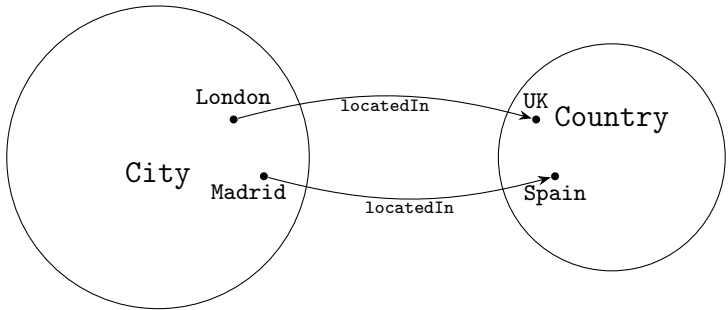


Modelling with OWL 2: Basic intuitions

OWL Ontologies: Basic intuitions (i)

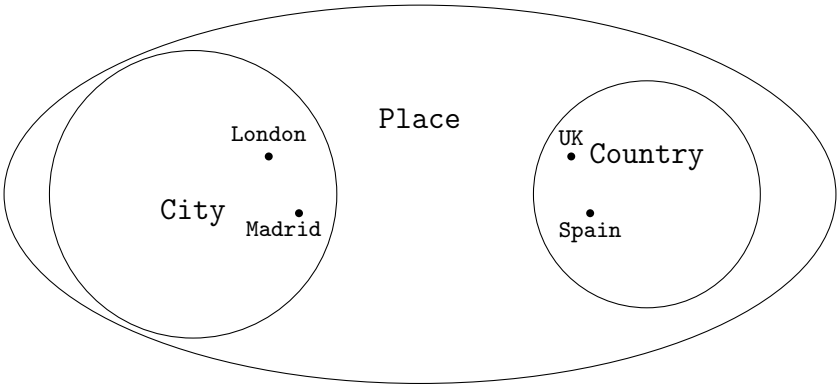
Classes interpreted as sets.

- City = {London, Madrid, . . . }
- Country = {UK, Spain, . . . }



OWL Ontologies: Basic intuitions (ii)

City and Country are subsets of the set Place



OWL Ontologies: Basic intuitions (iii)

Predicates can also be seen as sets (of pairs)

- locatedIn = { <London, UK>, <Madrid, Spain>, . . . }
- teaches = {<ernesto, INM713>, <horroks, KRR123>, . . . }

OWL Ontologies: Basic intuitions (iv)

From KG triples to the underlying Semantics

Interpretations ($\mathcal{I}$) can be seen as functions that bridge OWL entities to their representation within set theory.

KG Triples	DL Syntax	Semantics
`:london :location :england .`	$location(london, england)$	$\langle london^{\mathcal{I}}, england^{\mathcal{I}} \rangle \in location^{\mathcal{I}}$
`:london rdf:type :City .`	$City(london)$	$london^{\mathcal{I}} \in City^{\mathcal{I}}$
`:City rdfs:subClassOf :Place .`	$City \sqsubseteq Place$	$City^{\mathcal{I}} \subseteq Place^{\mathcal{I}}$
`:capitalOf rdfs:subPropOf :location .`	$capitalOf \sqsubseteq location$	$capitalOf^{\mathcal{I}} \subseteq location^{\mathcal{I}}$

Modelling with OWL 2: Entities

OWL 2 entities: classes and individuals

`owl:Class`: to represent classes (*i.e.*, set of individuals).

- Atomic (with a IRI) `:Person rdf:type owl:Class`
- Complex (built from other entities)

OWL 2 entities: classes and individuals

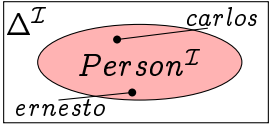
`owl:Class`: to represent classes (*i.e.*, set of individuals).

- Atomic (with a IRI) `:Person rdf:type owl:Class`
- Complex (built from other entities)

`owl:NamedIndividual`, to represent individuals

```
:ernesto rdf:type owl:NamedIndividual
:ernesto rdf:type :Person
```

$:ernesto^{\mathcal{I}} \in :Person^{\mathcal{I}}$



OWL 2 entities: Top and Bottom classes

`owl:Thing`

- $\top$ (in DL syntax)
- Class containing **all individuals**.
- Its interpretation is $\Delta^{\mathcal{I}}$.
- For every `owl:Class` C, C is a subclass of `owl:Thing`.

OWL 2 entities: Top and Bottom classes

`owl:Thing`

- $\top$ (in DL syntax)
- Class containing **all individuals**.
- Its interpretation is $\Delta^{\mathcal{I}}$.
- For every `owl:Class` C, C is a subclass of `owl:Thing`.

`owl:Nothing`

- $\bot$ (in DL syntax)
- **empty class** containing no individuals.
- Its interpretation is the empty set $\emptyset$
- For every `owl:Class` C, `owl:Nothing` is a subclass of C

OWL 2 entities: properties

- `owl:DatatypeProperty` . Targets data values.

```
:hasName rdf:type owl:DatatypeProperty.
:ernesto :hasName ‘‘Ernesto Jimenez-Ruiz’’ .
```

- `owl:ObjectProperty`. Targets individuals.

```
:teaches rdf:type owl:ObjectProperty
:ernesto :teaches :INM713 .
```

- `owl:AnnotationProperty`. No logical implication.

```
rdfs:label rdf:type owl:AnnotationProperty .
:ernesto rdfs:label ‘‘Ernesto JR’’ .
```

The set of classes, named individuals, annotation, data and object properties are **mutually disjoint**.

Modelling with OWL 2: Axioms and Class Constructs

OWL 2 Axioms

- OWL 2 ontologies are composed by **axioms**.
- **Not all axioms have a 1-to-1 triple representation.**
- Historically, axioms are put in **boxes**.

OWL 2 Axioms

- OWL 2 ontologies are composed by **axioms**.
- **Not all axioms have a 1-to-1 triple representation.**
- Historically, axioms are put in **boxes**.
- **The TBox** (terminological knowledge)
  - is typically **independent of** any actual **instance data**.
  - Class inclusion $C \sqsubseteq D$, equivalence $C \equiv D$
  - The set of property axioms are also referred to as **RBox.**

OWL 2 Axioms

- OWL 2 ontologies are composed by **axioms**.
- **Not all axioms have a 1-to-1 triple representation.**
- Historically, axioms are put in **boxes**.
- **The TBox** (terminological knowledge)
  - is typically **independent of** any actual **instance data**.
  - Class inclusion $C \sqsubseteq D$, equivalence $C \equiv D$
  - The set of property axioms are also referred to as **RBox.**
- **The ABox** (assertional knowledge)
  - contains facts about concrete instances (basically as in RDF)

OWL 2 TBox Axioms

- The TBox (excluding the RBox) is composed by:
  - **Subsumption axioms:** $C \sqsubseteq D$ ($C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$)
  - **Equivalence axioms:** $C \equiv D$ ($C^{\mathcal{I}} = D^{\mathcal{I}}$)

OWL 2 TBox Axioms

- The TBox (excluding the RBox) is composed by:
  - **Subsumption axioms:** $C \sqsubseteq D$ ($C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$)
  - **Equivalence axioms:** $C \equiv D$ ($C^{\mathcal{I}} = D^{\mathcal{I}}$)

- C and D can be **named concepts** (*e.g.*, `dbo:City`) or **complex concepts** built from others.
  - **Negation**: $\neg E$ (not)
  - **Intersection**: $E \sqcap F$ (and)
  - **Union**: $E \sqcup F$ (or)
  - **Property restrictions**: $\exists R.E$ (some), $\forall R.E$ (only)

OWL 2 TBox Axioms

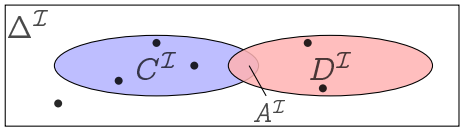
- The TBox (excluding the RBox) is composed by:
  - **Subsumption axioms:** $C \sqsubseteq D$ ($C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$)
  - **Equivalence axioms:** $C \equiv D$ ($C^{\mathcal{I}} = D^{\mathcal{I}}$)

- C and D can be **named concepts** (*e.g.*, `dbo:City`) or **complex concepts** built from others.
  - **Negation**: $\neg E$ (not)
  - **Intersection**: $E \sqcap F$ (and)
  - **Union**: $E \sqcup F$ (or)
  - **Property restrictions**: $\exists R.E$ (some), $\forall R.E$ (only)

- We will focus on the cases where the **LHS concept is atomic**. *e.g.*, `dbo:City` $\sqsubseteq$ `dbo:Place` $\sqcap$ ...

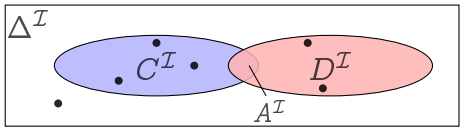
Union and Intersection

- $A \sqsubseteq C \sqcap D$. "A is both C and D."
  - *e.g.*, $\texttt{Mother} \sqsubseteq \texttt{Parent} \sqcap \texttt{Female}$.

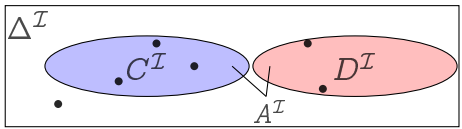


Union and Intersection

- $A \sqsubseteq C \sqcap D$. "A is both C and D."
  - *e.g.*, $\texttt{Mother} \sqsubseteq \texttt{Parent} \sqcap \texttt{Female}$.

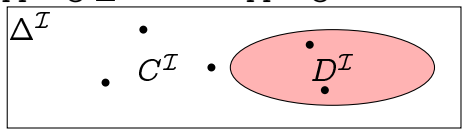


- $A \sqsubseteq C \sqcup D$. "A is C or D."
  - *e.g.*, $\texttt{Parent} \sqsubseteq \texttt{Mother} \sqcup \texttt{Father}$.



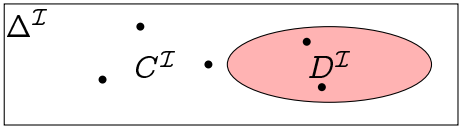
Negation and Disjointness

- $C \sqsubseteq \neg D$: "C is not D."
  - *e.g.*, $\texttt{VegetarianTopping} \sqsubseteq \neg\texttt{MeatTopping}$.

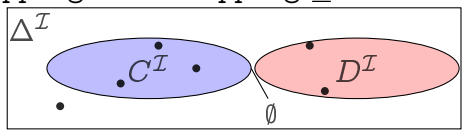


Negation and Disjointness

- $C \sqsubseteq \neg D$: "C is not D."
  - *e.g.*, $\texttt{VegetarianTopping} \sqsubseteq \neg\texttt{MeatTopping}$.



- $C \sqcap D \sqsubseteq \bot$: "Nothing is both a C and a D."
  - Equivalent to $C \sqsubseteq \neg D$ (and $D \sqsubseteq \neg C$).
  - *e.g.*, $\texttt{VegetarianTopping} \sqcap \texttt{MeatTopping} \sqsubseteq \bot$.

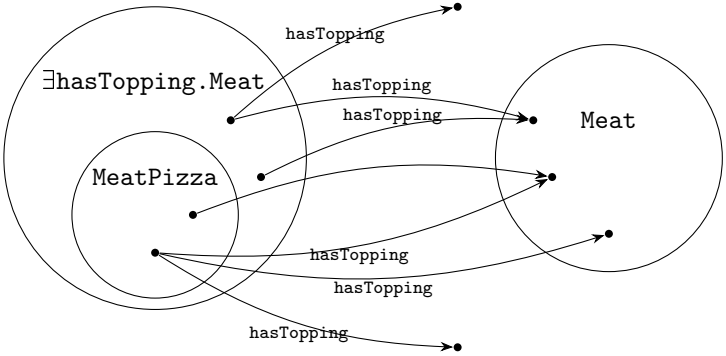


Existential Restrictions

- $A \sqsubseteq \exists R.C$: "$A$ is R-related to (at least) one C."
- $(\exists R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid$ there is a b where $\langle a, b\rangle \in R^{\mathcal{I}}$ and $b \in C^{\mathcal{I}}\}$

Existential Restrictions

- $A \sqsubseteq \exists R.C$: "$A$ is R-related to (at least) one C."
- $(\exists R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \text{there is a } b \text{ where } \langle a, b \rangle \in R^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\}$
- *e.g.*, `MeatPizza` $\sqsubseteq \exists$`hasTopping.Meat`



Existential Restrictions and `rdfs:domain`

Local scope for R:

- `Pizza` $\sqsubseteq \exists$`hasTopping.Topping`

Domain: (global scope for R)

- If R has the *domain* C: (`R rdfs:domain C`)
- then anything 'using' R is in C.
- Domain can also be expressed as: $\exists R.\top \sqsubseteq C$
- $\exists$`hasTopping`.$\top \sqsubseteq$ `Pizza`

Existential Restrictions and `rdfs:domain`

Local scope for R:

- `Pizza` $\sqsubseteq \exists$`hasTopping.Topping`

Domain: (global scope for R)

- If R has the *domain* C: (`R rdfs:domain C`)
- then anything 'using' R is in C.
- Domain can also be expressed as: $\exists R.\top \sqsubseteq C$
- $\exists$`hasTopping`$.\top \sqsubseteq$ `Pizza`
- Examples:
  - `:pizza1 :hasTopping :meat1`
  - `:ice-cream1 :hasTopping :choco-chips1`

Existential Restrictions and `rdfs:domain`

Local scope for R:

- `Pizza` $\sqsubseteq$ $\exists$`hasTopping.Topping`

Domain: (global scope for R)

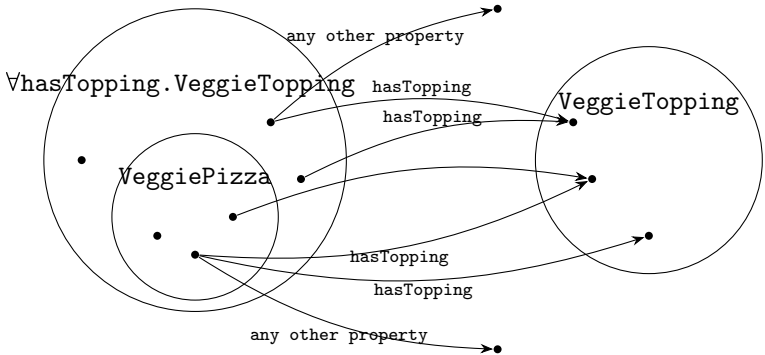
- If R has the *domain* C: (`R rdfs:domain C`)
- then anything 'using' R is in C.
- Domain can also be expressed as: $\exists R.\top \sqsubseteq C$
- ~~$\exists$`hasTopping`$.\top \sqsubseteq$ `Pizza`~~. $\exists$`hasTopping`$.\top \sqsubseteq$ `Food`
- Examples:
  - `:pizza1 :hasTopping :meat1`
  - `:ice-cream1 :hasTopping :choco-chips1`

Universal Restrictions

- $A \sqsubseteq \forall R.C$: A has R-relationships to C's only.
- $(\forall R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \text{for all } b, \text{ if } \langle a, b\rangle \in R^{\mathcal{I}} \text{ then } b \in C^{\mathcal{I}}\}$

Universal Restrictions

- $A \sqsubseteq \forall R.C$: A has R-relationships to C's only.
- $(\forall R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \text{for all } b, \text{ if } \langle a, b\rangle \in R^{\mathcal{I}} \text{ then } b \in C^{\mathcal{I}}\}$
- *e.g.*, $\texttt{VeggiePizza} \sqsubseteq \forall \texttt{hasTopping}.\texttt{VeggieTopping}$



Universal Restrictions and `rdfs:range`

Local scope for R:

- $\texttt{VeggiePizza} \sqsubseteq \forall \texttt{hasTopping}.\texttt{VeggieTopping}$.

Range: (global scope for R)

- If role R has the *range* C: (`R rdfs:range C`)
- then anything one can reach by R is in C,
- Range can also be expressed as $\top \sqsubseteq \forall R.C$.
- $\top \sqsubseteq \forall \texttt{hasTopping}.\texttt{VeggieTopping}$.

Universal Restrictions and `rdfs:range`

Local scope for R:

- `VeggiePizza` $\sqsubseteq \forall$`hasTopping.VeggieTopping`.

Range: (global scope for R)

- If role R has the *range* C: (`R rdfs:range C`)
- then anything one can reach by R is in C,
- Range can also be expressed as $\top \sqsubseteq \forall R.C$.
- $\top \sqsubseteq \forall$`hasTopping.VeggieTopping`.
- Example:
  - `:pizza1 :hasTopping :meat1` (is `meat1` a `VeggieTopping`?)

Universal Restrictions and `rdfs:range`

Local scope for R:

- `VeggiePizza` $\sqsubseteq \forall$`hasTopping.VeggieTopping`.

Range: (global scope for R)

- If role R has the *range* C: (`R rdfs:range C`)
- then anything one can reach by R is in C,
- Range can also be expressed as $\top \sqsubseteq \forall R.C$.
- ~~$\top \sqsubseteq \forall$`hasTopping.VeggieTopping`~~. $\top \sqsubseteq \forall$`hasTopping.PizzaTopping`.
- Example:
  - `:pizza1 :hasTopping :meat1`

Cardinality restrictions

- Restricts the number of relations a type of object can have.
- Syntax:
  - $\leq_n R.C$, $\geq_n R.C$, and $=_n R.C$.

Cardinality restrictions

- Restricts the number of relations a type of object can have.
- Syntax:
  - $\leq_n R.C$, $\geq_n R.C$, and $=_n R.C$.
- Axioms read:
  - $A \sqsubseteq \square_n R.C$: “An element of A is R-related to n number of C’s.”
    - $\leq$: *at most*
    - $\geq$: *at least*
    - $=$: *exactly*
- *e.g.*, $\texttt{SuperMeatPizza} \sqsubseteq \geq_5 \texttt{hasTopping.Meat}$

Modeling with OWL 2: RBox

Property axioms

- subsumption:
  $\texttt{hasBrother} \sqsubseteq \texttt{hasSibling}$
- equivalence:
  $\texttt{hasLocation} \equiv \texttt{locatedIn}$
- inverse roles (only object properties):
  $\texttt{hasParent} \equiv \texttt{hasChild}^{-1}$
- role chains (only object properties):
  $\texttt{hasParent} \circ \texttt{hasBrother} \sqsubseteq \texttt{hasUncle}$
  $(R \circ S)^{\mathcal{I}} = \{\langle a^{\mathcal{I}}, c^{\mathcal{I}} \rangle \mid \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}, \langle b^{\mathcal{I}}, c^{\mathcal{I}} \rangle \in S^{\mathcal{I}}\}$

Common characteristics for (object) properties

A relation R over the set $\Delta^{\mathcal{I}}$ ($R \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$) is

Characteristic	**Semantics**	**Example**
Reflexive:	if $\langle a, a \rangle \in R$ for all $a \in \Delta^{\mathcal{I}}$	`part_of`
Irreflexive:	if $\langle a, a \rangle \notin R$ for all $a \in X$	`hasParent`
Symmetric:	if $\langle a, b \rangle \in R$ implies $\langle b, a \rangle \in R$	`hasSibling`
Asymmetric:	if $\langle a, b \rangle \in R$ implies $\langle b, a \rangle \notin R$	`memberOf/hasParent`
Transitive:	if $\langle a, b \rangle, \langle b, c \rangle \in R$ implies $\langle a, c \rangle \in R$	`locatedIn`
Functional:	if $\langle a, b \rangle, \langle a, c \rangle \in R$ implies $b = c$	`hasBiologicalMother`
Inverse functional:	if $\langle a, b \rangle, \langle c, b \rangle \in R$ implies $a = c$	`gaveBirthTo`

(*) Functional characteristics can also be applied to data properties.

Datatypes for data properties

- Many predefined datatypes are available in OWL:
  - all common XSD datatypes: `xsd:string`, `xsd:int`, ...
  - a few from RDF: `rdf:PlainLiteral`,
  - and a few of their own: `owl:real` and `owl:rational`.

Datatypes for data properties

- Many predefined datatypes are available in OWL:
  - all common XSD datatypes: `xsd:string`, `xsd:int`, ...
  - a few from RDF: `rdf:PlainLiteral`,
  - and a few of their own: `owl:real` and `owl:rational`.
- Datatypes may be restricted with *constraining facets*, borrowed from XML Schema.
  - `Teenager` is equivalent to:
    `Person` $\sqcap$ (∃`age.xsd:integer[>= 13, <= 19]`)

Modeling with OWL 2: ABox

ABox: Assertional axioms

Contains:

- A set of concept assertions $C(a)$ as in RDF
  DL: `Person(ernesto)`
  Triple `:ernesto rdf:type :Person`

ABox: Assertional axioms

Contains:

- A set of concept assertions $C(a)$ as in RDF
  DL: `Person(ernesto)`
  Triple `:ernesto rdf:type :Person`
- Role assertions $R(b, c)$ as in RDF
  DL: `teaches(ernesto, inm713-in3067)`
  Triple: `:ernesto :teaches :inm713-in3067`

ABox: Assertional axioms

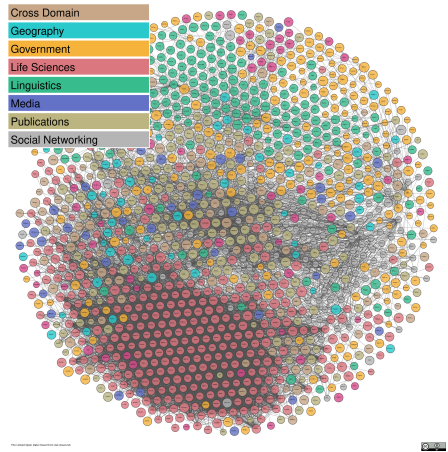
Contains:

- A set of concept assertions $C(a)$ as in RDF
  DL: `Person(ernesto)`
  Triple `:ernesto rdf:type :Person`
- Role assertions $R(b, c)$ as in RDF
  DL: `teaches(ernesto, inm713-in3067)`
  Triple: `:ernesto :teaches :inm713-in3067`
- Equality and non-equality between individuals
  - DL: `ernesto` $=$ `ejr`, `ernesto` $\neq$ `aidan`;
  - Triple (1) `:ernesto owl:sameAs :ejr` (2) `:ernesto owl:differentFrom :aidan`

Knowledge Graphs on the Web and beyond

Open KGs

- Linked Open Datasets:
  `https://lod-cloud.net/`
- Diverse domains: media, government, geography, tourism, life sciences, ecotoxicology, etc.
- General purpose KGs: DBpedia, Freebase, Wikidata, YAGO
  - `https://www.wikidata.org/`
  - `https://dbpedia.org/`



Domain specific KGs: Life Sciences

- **UMLS** (Unified Medical Language System) Metathesaurus
  - Integrates more than two hundred thesauri and ontologies
  - Contains more than 15 million concept names

- **BioPortal**
  - Contains more than 1,000 ontologies
  - Represent a network of ontologies
  - More than 79 million mappings are available
  - Also includes user-submitted alignments

Enterprise KGs (i)

- Websearch (*e.g.*, Bing, Google),
- Commerce (*e.g.*, Airbnb, Amazon, eBay, Uber),
- Social networks (*e.g.*, Facebook, LinkedIn),
- Finance (*e.g.*, Accenture, Banca d'Italia, Bloomberg, Capital One)

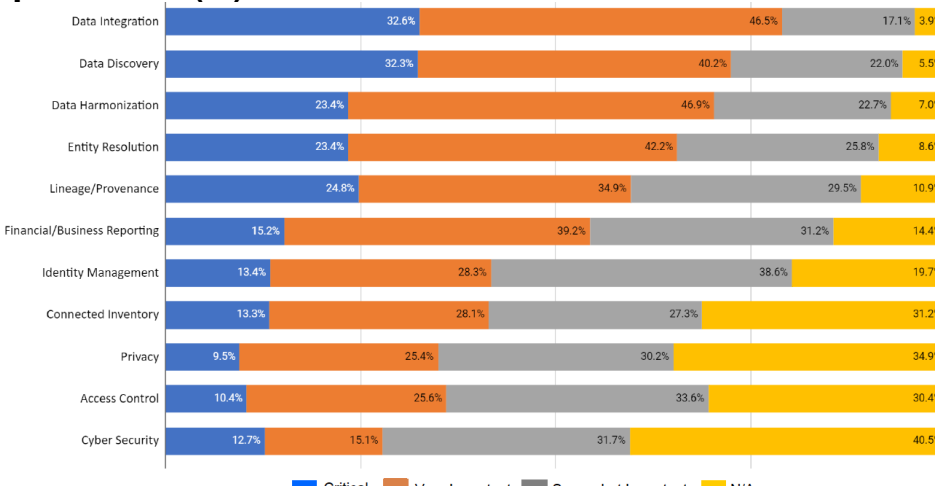
Aidan Hogan et al. Knowledge Graphs. Morgan & Claypool 2021.

Enterprise KGs (ii)

	Data model	Size of the graph	Development stage
Microsoft	The types of entities, relations, and attributes in the graph are defined in an ontology.	~2 billion primary entities, ~55 billion facts	Actively used in products
Google	Strongly typed entities, relations with domain and range inference	1 billion entities, 70 billion assertions	Actively used in products
Facebook	All of the attributes and relations are structured and strongly typed, and optionally indexed to enable efficient retrieval, search, and traversal.	~50 million primary entities, ~500 million assertions	Actively used in products
eBay	Entities and relation, well-structured and strongly typed	Expect around 100 million products, >1 billion triples	Early stages of development and deployment
IBM	Entities and relations with evidence information associated with them.	Various sizes. Proven on scales documents >100 million, relationships >5 billion, entities >100 million	Actively used in products and by clients

Natasha Noy et al. Industry-scale Knowledge Graphs: Lessons and Challenges. Communications of the ACM (2019).

Enterprise KGs (iii): Use cases



Enterprise Knowledge Graph Foundation: `https://www.ekgf.org`

How can we access the KGs?

- RDF dumps.
- From *SPARQL endpoints*:
  - Data kept in a *triple store*
  - Exposes data (in different formats)
    - with endpoint frontends, e.g.,
      `http://dbpedia.org/resource/London`,
      `https://dbpedia.org/page/City,_University_of_London` or
    - by direct SPARQL query endpoint: *e.g.*,
      `http://dbpedia.org/sparql`
      `https://query.wikidata.org/`
      `https://www.ebi.ac.uk/rdf/datasets/`

Acknowledgements

Acknowledgements

- Prof. Martin Giese and others (University of Oslo)
- INF4580 – Semantic technologies
- https://www.uio.no/studier/emner/matnat/ifi/INF4580/

Laboratory Session

Hands-on with RDF and OWL

- Create a small knowledge graph.
- Modelling with OWL.
- Pizza OWL tutorial (optional).