KRR: Ontologies

(for knowledge graphs)

Artificial Intelligence Challenge / Introduction to Artificial Intelligence

ICM 2A + M1 CPS²

15th March 2024

antoine.zimmermann@emse.fr

What we've done so far with KGs

- The kinds of KG we've seen or made so far are purely descriptive
- They describe knowledge of the specifics, the particulars, the instances
- But we need general concepts to describe them (e.g., EMSE is an *engineering school*) and general kinds of attributes, properties, relations (e.g., a mandate *starts* at some point)
- We need ways to identify what are the entities of interest that are *generic* and be able to assert *general knowledge* about them

Example of general knowledge

- All microblogging platforms are social networking platforms.
- A blog is a web platform where all blogposts have a date of publication.
- A message that is private is not public
- A direct message is a private message
- People do not have multiple birthdates
- A thread has a first message and each message has a set of responses
- Responses to a message are messages
- Etc.

Ontologies

- The term *ontology* comes from **philosophy**: it is *the study of being*
- Philosophical ontologists ask, and try to answer, questions like "*what exists?*" which can lead to attempts at classifying everything; then "*what is matter?*", "*what is time?*", then further down to "*what is cost?*" (or, put differently, "*what is **the nature** of these things?*")
- In **computer science**, ontologies are countable, and **an ontology** is *a formal and explicit specification of the concepts, entities, and relationships that exist within a particular domain or knowledge base*.
- A computer ontologists ask "*what are the **classes** that we need to identify for this domain of knowledge? What are the **relations** that may exist between the things of interest in the domain? What are the **axioms** that assert truth about certain kinds of things in the domain?*"

Basic ontological knowledge as graphs

Table 3.1: Definitions for sub-class, sub-property, domain and range

Feature	Definition	Condition	Example
SUB-CLASS	c —subc. of→ d	x —type→ c implies x —type→ d	City —subc. of→ Place
SUB-PROPERTY	p —subp. of→ q	x —p→ y implies x —q→ y	venue —subp. of→ location
DOMAIN	p —domain→ c	x —p→ y implies x —type→ c	venue —domain→ Event
RANGE	p —range→ c	x —p→ y implies y —type→ c	venue —range→ Venue

Ontologies and logics

- Logics are almost always used to formalise the axioms that hold true in a domain
- But ontologies are more than just logical theories
- Ontologies usually comprise:
  - A **logical theory**: formal axioms that are true of the domain
  - A **documentation**: definitions of what the terms in the theory are; what is their kind, e.g., a class vs. a relation; and what they are *intended* to describe (a natural language description)

What logic do we use for ontologies?

- Any logic can be used in principles, but a family of logics is very often used: *Description Logics* (DL)
- DLs can always be mapped to a subset of First Order Logic
- DLs only deal with *instances*, *classes*, and *binary relations*

  [+ *special treatment of datatypes and literals*]

  → Provide the means to define the nodes and edges of KGs
- DLs are "**modular**" (constructs of the language can be added or removed, that leads to more or less complex logics)
- DLs are the underlying logic of the *Web Ontology Language*, a standard that is the most used ontology language

Description Logics: the basics (1)

Syntax:

- Symbols are:
  - A set $\mathbf{I}$ of *instance names* (also called *individual names* or just *individuals*)
  - A set $\mathbf{C}$ of *class names* (also called *concept names*)
  - A set $\mathbf{R}$ of *relation names* (also called *role names* or *object properties*)
  - Symbols { '(' , ')' , ',' , '$\sqsubseteq$' }
- Formulas are:
  - If c and d are in $\mathbf{C}$, then $c \sqsubseteq d$ is a formula
  - If c is in $\mathbf{C}$ and a is in $\mathbf{I}$, then $c(a)$ is a formula
  - If r is in $\mathbf{R}$ and a and b are in $\mathbf{I}$, then $r(a,b)$ is a formula

DL syntax as a d.e.l.g.

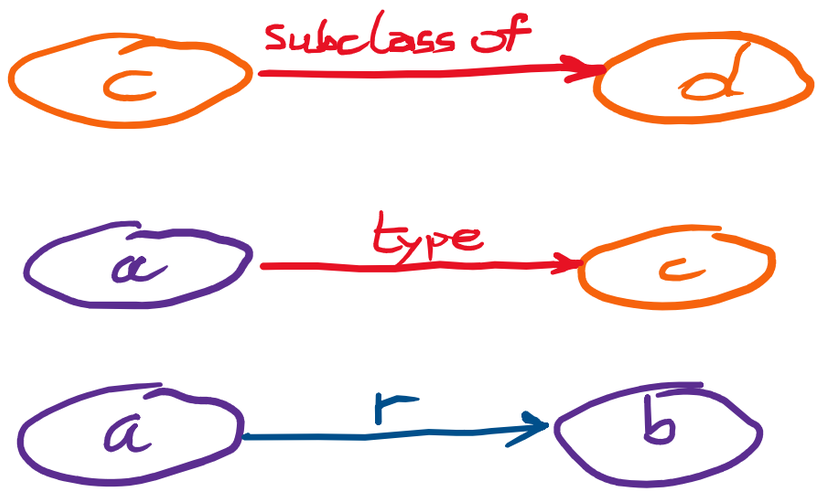
Formulas in DL syntax

$c \sqsubseteq d$

$c(a)$

$r(a,b)$

Directed **e**dge-**l**abelled **g**raph

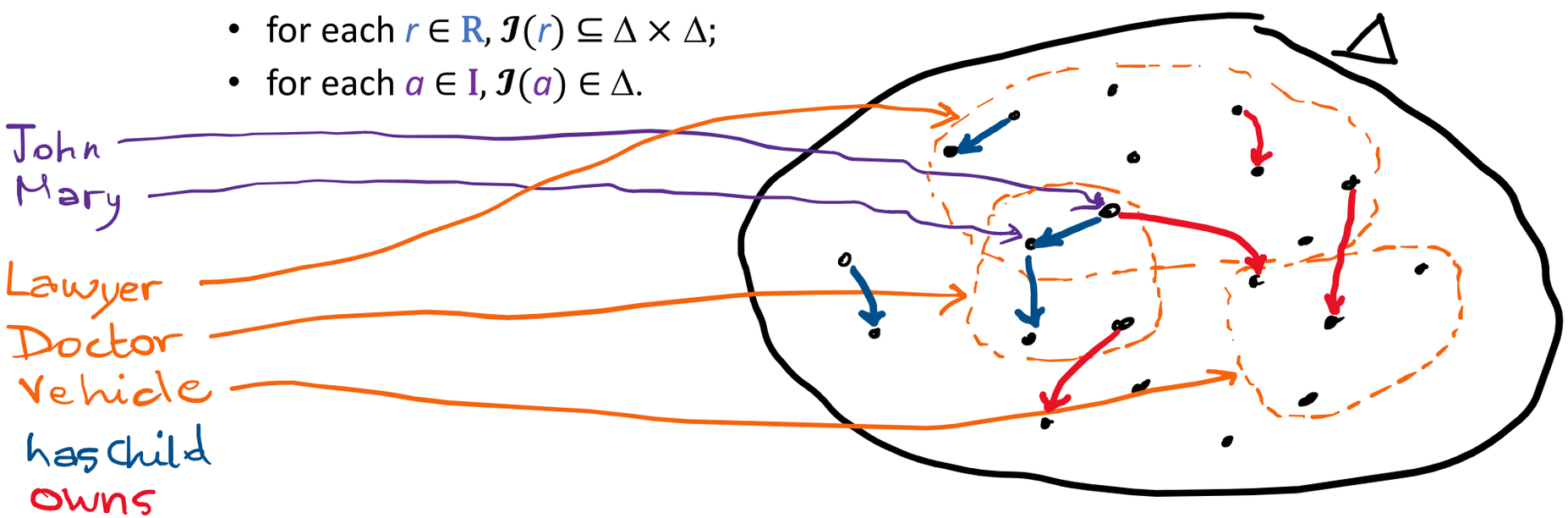


Description Logics: the basics (2)

- Semantics
  - Interpretations are pairs $(\Delta, \mathcal{I})$ such that:
    - Δ is a non-empty set, called the **domain of interpretation** (or universe)
    - $\mathcal{I}$ is a function defined on the set $\mathbf{C} \cup \mathbf{R} \cup \mathbf{I}$ such that:
      - for each $c \in \mathbf{C}$, $\mathcal{I}(c) \subseteq \Delta$;
      - for each $r \in \mathbf{R}$, $\mathcal{I}(r) \subseteq \Delta \times \Delta$;
      - for each $a \in \mathbf{I}$, $\mathcal{I}(a) \in \Delta$.

Description Logics: the basics (2)

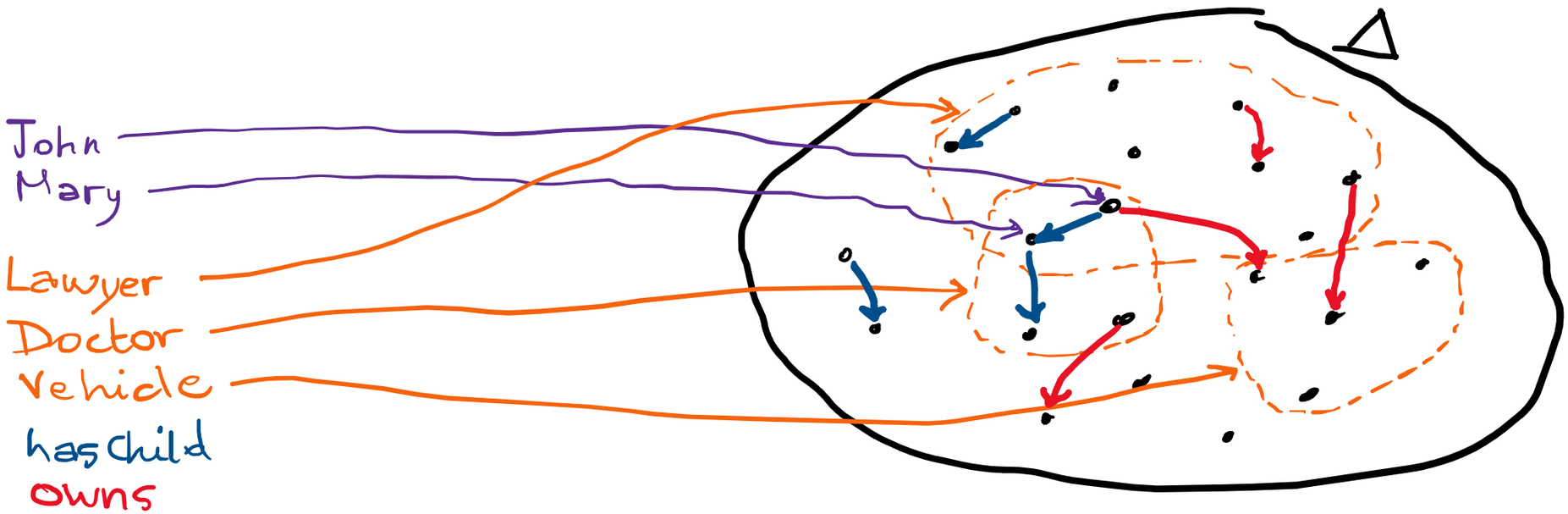
- Semantics
  - Interpretations are pairs $(\Delta, \mathcal{I})$ such that:
    - Δ is a non-empty set, called the **domain of interpretation** (or universe)
    - $\mathcal{I}$ is a function defined on the set $\mathbf{C} \cup \mathbf{R} \cup \mathbf{I}$ such that:
      - for each $c \in \mathbf{C}$, $\mathcal{I}(c) \subseteq \Delta$;
      - for each $r \in \mathbf{R}$, $\mathcal{I}(r) \subseteq \Delta \times \Delta$;
      - for each $a \in \mathbf{I}$, $\mathcal{I}(a) \in \Delta$.



Description Logics: the basics (2)

- Semantics
  - Interpretations are pairs $(\Delta, \mathcal{I})$ such that:
    - Δ is a non-empty set, called the **domain of interpretation** (or universe)
    - $\mathcal{I}$ is a function defined on the set $\mathbf{C} \cup \mathbf{R} \cup \mathbf{I}$ such that:
      - for each $c \in \mathbf{C}$, $\mathcal{I}(c) \subseteq \Delta$;
      - for each $r \in \mathbf{R}$, $\mathcal{I}(r) \subseteq \Delta \times \Delta$;
      - for each $a \in \mathbf{I}$, $\mathcal{I}(a) \in \Delta$.
  - An interpretation $(\Delta, \mathcal{I})$ in DL satisfies:
    - $c \sqsubseteq d$ if and only if $\mathcal{I}(c) \subseteq \mathcal{I}(d)$;
    - $c(a)$ if and only if $\mathcal{I}(a) \in \mathcal{I}(c)$;
    - $r(a,b)$ if and only if $(\mathcal{I}(a), \mathcal{I}(b)) \in \mathcal{I}(r)$.

Description Logics: the basics (2)



This interpretation satisfies: Lawyer(John), Lawyer(Mary), Doctor(John), Doctor(Mary), and hasChild(John,Mary), Doctor $\sqsubseteq$ Doctor, Lawyer $\sqsubseteq$ Lawyer, Vehicle $\sqsubseteq$ Vehicle

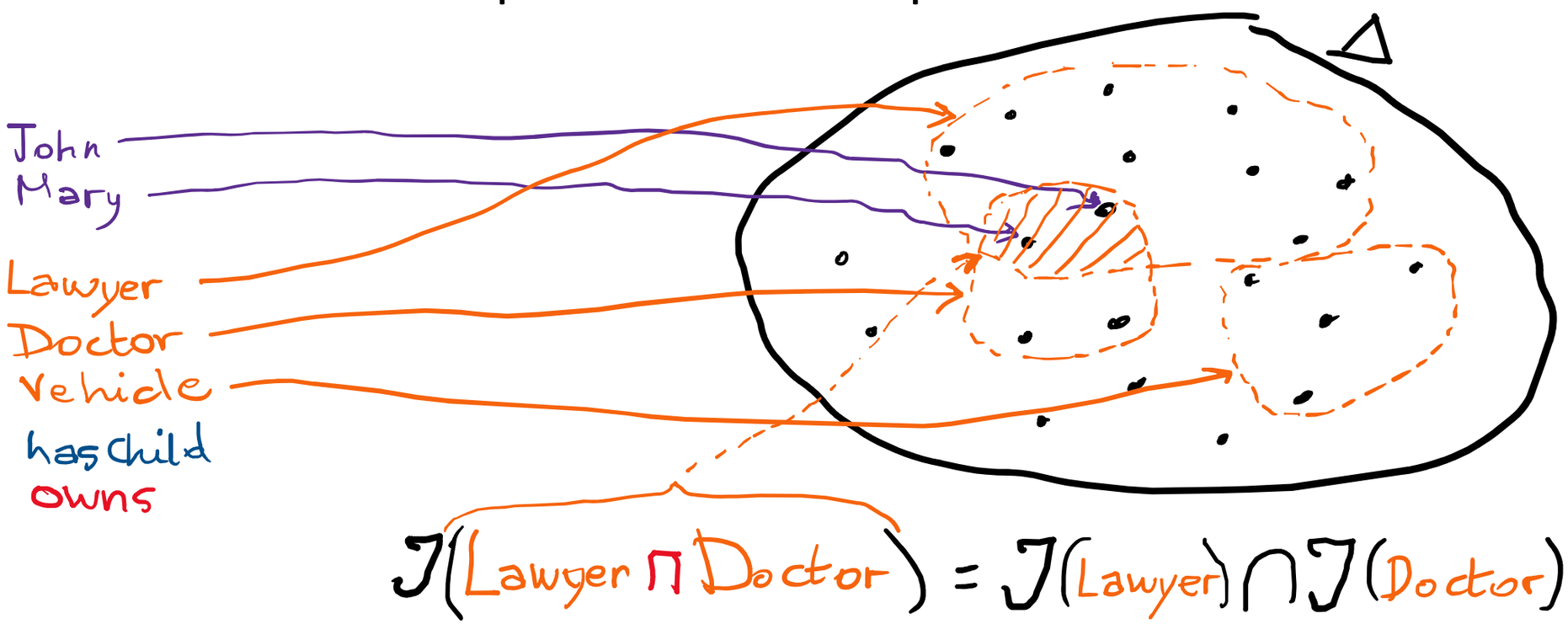
This interpretation **does not** satisfy: Vehicle(John), Vehicle(Mary), hasChild(Mary,John), owns(Mary,John), owns(John,Mary), Doctor $\sqsubseteq$ Lawyer, Lawyer $\sqsubseteq$ Doctor, Doctor $\sqsubseteq$ Vehicle, Vehicle $\sqsubseteq$ Doctor, Lawyer $\sqsubseteq$ Vehicle, Vehicle $\sqsubseteq$ Lawyer.

Description Logics: adding constructs

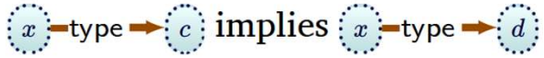
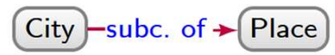
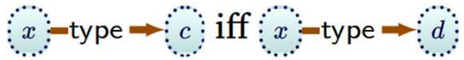
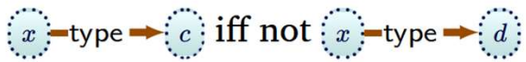
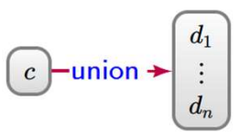
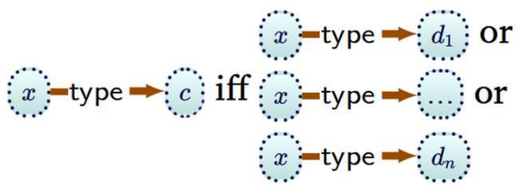
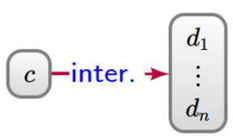
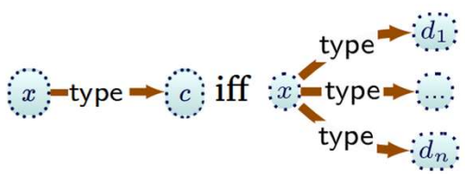
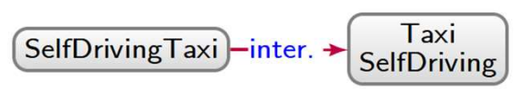
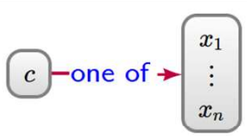
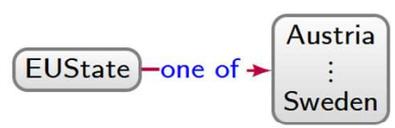
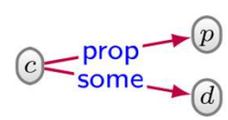
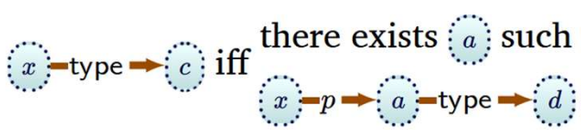
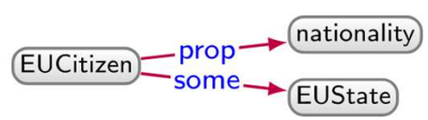
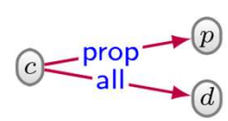
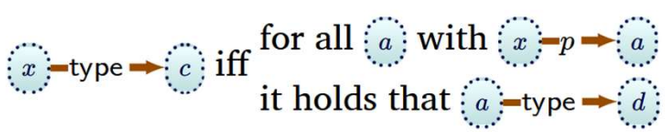
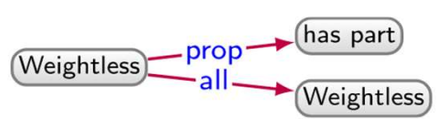
- In DL, we can define *complex classes* by combining names and operators (here is a sample):
  - The class of everything: $\top$
  - The empty class: $\bot$
  - The class of things that are lawyer and doctor: $\texttt{Lawyer} \sqcap \texttt{Doctor}$
  - The class of things that are either lawyer or doctor: $\texttt{Lawyer} \sqcup \texttt{Doctor}$
  - The class of things that have a child who is a doctor: $\exists \texttt{hasChild}.\texttt{Doctor}$
  - The class of things who only have doctor-children: $\forall \texttt{hasChild}.\texttt{Doctor}$
  - …
- Complex classes can be used in formulas:
  - $\texttt{Lawyer} \sqcap \texttt{Vehicle} \sqsubseteq \bot$ (the classes Lawyer and Vehicle are disjoint)
  - $\texttt{Parent} \sqsubseteq \exists \texttt{hasChild}.\top$ (parents have – at least – a child)
  - $\top \sqsubseteq \texttt{Good} \sqcup \texttt{Bad}$ (everything is either good or bad)

Interpretation of complex classes

- Given an interpretation of instance/class/relation *names*, we can define what the interpretation of complex classes are



Description Logics constructs and graphs

Feature	Axiom	Condition (for all x_*, y_*, z_*)	Example
SUB-CLASS			
EQUIVALENCE			
DISJOINT			
COMPLEMENT			
UNION			
INTERSECTION			
ENUMERATION			
SOME VALUES			
ALL VALUES			

Edited an ontology file with Protégé

- We will practice ontology editing using an ontology editor called Protégé
  - https://protege.stanford.edu/
  - Download the latest version (5.6.3)
  - You can skip the registration form if you want
  - It immediately works from an executable file (no install)

Ontology development methodology

- There exists several methodologies. We will roughly follow the “Ontology Development 101 methodology”:[1]
  - Step 1. Determine the domain and scope of the ontology
  - Step 2. Consider reusing existing ontologies
  - Step 3. Enumerate important terms in the ontology
  - Step 4. Define the classes and the class hierarchy
  - Step 5. Define the properties of classes – “slots”
  - Step 6. Define the facets of the slots
  - Step 7. Create instances
- **For a more agile process**, you can define a narrow scope first, and cycle back to step 1 to enhance your ontology. You will also probably go back and forth between the steps as you realise things are missing or incorrect

[1] https://protege.stanford.edu/publications/ontology_development/ontology101.pdf

Rules to keep in mind

1. *There is no one correct way to model a domain – there are always **viable alternatives**. The best solution almost always **depends on the application** that you have in mind and the extensions that you anticipate.*
2. *Ontology development is necessarily an **iterative** process.*
3. *Concepts in the ontology should be close to objects (physical or logical) and relationships **in your domain of interest**. These are most likely to be nouns (objects) or verbs (relationships) in sentences that describe your domain.*

Step 1: The domain and scope of the ontology

- What is the domain that the ontology will cover?
- For what we are going to use the ontology?
- For what types of questions the information in the ontology should provide answers?
- Who will use and maintain the ontology?

Decide what the **competency questions** are: questions that a knowledge graph using your ontology should be able to answer

E.g., *can I get this product delivered to my home?*

If the ontology does not define anything related to delivery, this question cannot be answered. If it provides the cost of delivery but does not say the geographic range of the delivery service, we cannot answer the question

Step 2: Consider reusing existing ontologies

- Several Web standards are ontologies:
  - W3C Organization Ontology (https://www.w3.org/TR/vocab-org/)
  - W3C Semantic Sensor Network Ontology (https://www.w3.org/TR/vocab-ssn/)
  - W3C PROV Ontology (https://www.w3.org/TR/prov-o/)
  - W3C Data Cube vocabulary (https://www.w3.org/TR/vocab-data-cube/)
  - W3C Data Catalogue vocabulary (https://www.w3.org/TR/vocab-dcat-2/)
  - ETSI Smart Applications REFerence Ontologies (https://saref.etsi.org/)
- Some ontologies are supported by companies or organisations:
  - Schema.org (consortium of companies) (https://schema.org/)
  - GoodRelations (an ontology for e-commerce, supported by several e-commerce platforms) (https://www.heppnetz.de/projects/goodrelations/)
  - GS1 Web vocabulary (mainly product description, maintained by non-profit GS1, the organisation that standardises barcodes) (https://www.gs1.org/voc/)
  - …

Step 2 (cont'd): ontology repositories or indexes

- Linked Open Vocabulary (https://lov.linkeddata.es/)
- Open Biological and Biomedical Ontology Foundry (https://obofoundry.org/)
- BioPortal (https://bioportal.bioontology.org/)
- AgroPortal (https://agroportal.lirmm.fr/)
- Ontology Design Patterns (http://ontologydesignpatterns.org/)
- …

Step 3. Enumerate important terms in the ontology

- You can start by listing everything that comes to your mind: brainstorm together
- There can be useful concepts that do not correspond to a single word
- Then, discuss which terms should be classes, which ones are relations, which ones are out of scope or irrelevant
- Normally, it is unlikely that an ontology of a domain contain instances (the instances are normally for specific applications)

Step 4. Define the classes and the class hierarchy

- You can have a top-down or bottom-up approach:
  - **Top-down**: what are the most generic classes we want to cover (e.g. Product), then how these are divided into the main subclasses (e.g., Food, Cloth, etc.), and so on
  - **Bottom-up**: what is the finest granularity before the individuals that correspond to a class (e.g. AMD Ryzen 3 1200 AF Wraith Stealth Edition (3.1 GHz / 3.4 GHz). Then generalize (e.g., AMD Ryzen 3 processors). You may have multiple inheritance (e.g., 3.1 GHz processors)
- For all terms identified in Step 3, you can think of what are more general classes, and what are more specialised ones

Step 5. Define the properties of classes – “slots”

- What are the general characteristics of all the instances of a class? Sometimes, something that we associate with a class intuitively may in fact be better as a characteristic of something else (e.g., being “out-of-stock” is not a characteristic of a product *per se*)
- Sometimes, properties can also be defined and described independently of a class

Step 6. Define the facets of the slots

- This is where the important “axioms” are defined.
- For each property that applies to a type, one can define the type of the values of the property
- Properties may be transitive, symmetric, irreflexive, etc.
- There may be cardinality restrictions (e.g., the same offer has only 1 price)
- Sometimes, the property itself fixes the type of the subjects and objects used with it (*domain* and *range*)

Step 7. Create instances

- Normally, the instances are not part of the ontology
- The creation of instances serve 2 purposes:
  - As a way to exemplify how the model can be instantiated (documentation)
  - To provide the data that the application(s) need(s)
- Instances are often produced programmatically, either from users' input, or from other existing data (e.g., from a relational database, text documents, web pages, open data)
- Sometimes at this stage, one realises that are missing or incorrect things in the model