188.399 Einführung in Semantic Systems

Author:

Max Tiessler

2025-01-08

Contents

1 Introduction

2 What is an Ontology

Ontology is derived from the Greek words *ontos* (being) and *logos* (word), meaning the study or systematic explanation of existence.

In philosophy, ontology is a branch concerned with categorizing and explaining existence. Aristotle (400 BC) made early attempts to establish universal categories for classifying everything that exists.

2.1 Definition

According to the **Merriam-Webster dictionary**:

- A branch of metaphysics concerned with the nature and relations of being.
- A particular theory about the nature of being and kinds of existents.

In the context of knowledge representation, **Studer (1998)** defines ontology as seen in figure 1. This definition highlights several key aspects:

- **Formal**: Ontologies are machine-readable and interpretable.
- **Explicit**: Concepts, properties, functions, and axioms are clearly defined.
- **Shared**: Ontologies are agreed upon and shared by a community.
- **Conceptualization**: They provide an abstract model of some phenomena in the world.

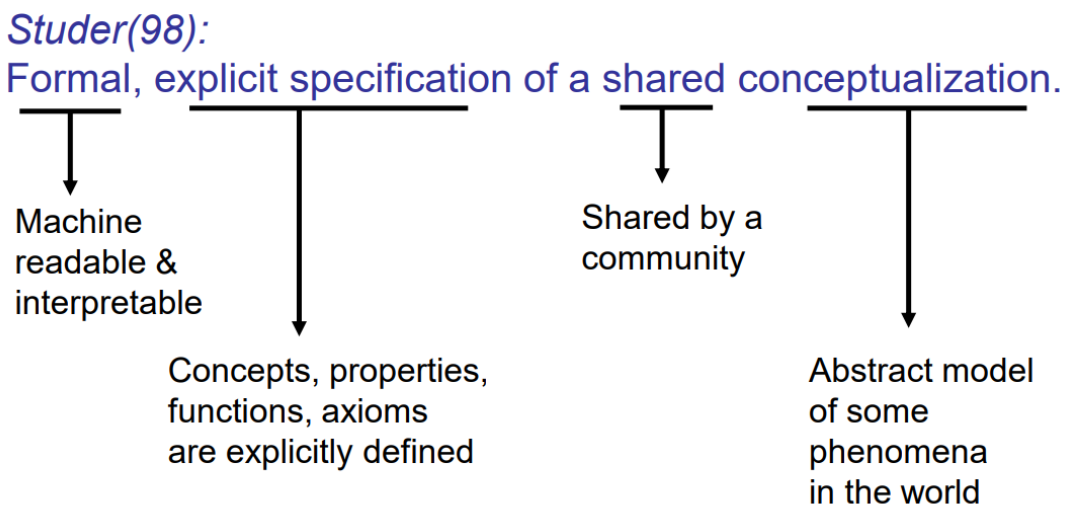

Figure 1: Ontology definition

2.2 Types and Categories

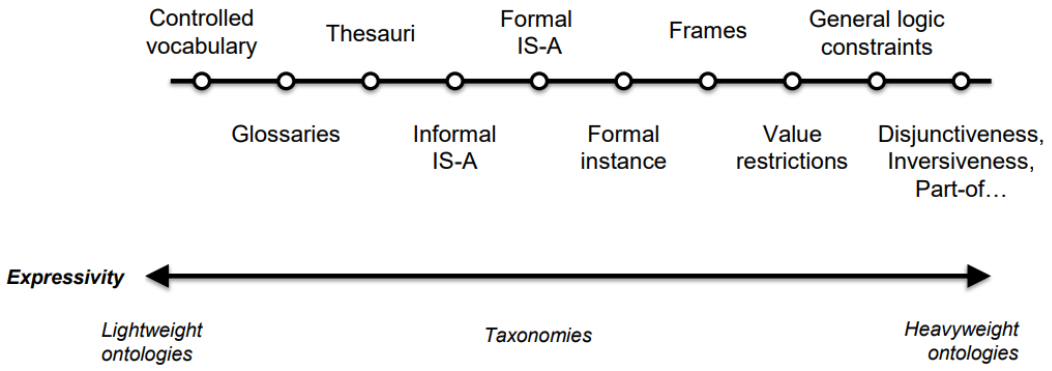

Figure 2: Ontology Types

2.2.1 Lightweight

Lightweight ontologies are simplified frameworks for organizing and defining knowledge. Key types include:

- **Controlled Vocabulary**: A finite list of terms, often used in catalogs.
  - **Example**: A library catalog listing book titles, authors, and genres (e.g., "Fiction," "Science," "Biography").
- **Glossary**: A controlled vocabulary with informal definitions provided in natural language.
  - **Example**: A glossary of medical terms, such as:
    - "Hypertension: High blood pressure."
    - "Cardiology: The branch of medicine dealing with the heart."
- **Thesauri**: A controlled vocabulary where concepts are connected through various relations:
  - **Equivalency**: Synonym relations between concepts.
    - **Example**: "Automobile" and "Car."
  - **Hierarchies**: Subclass and superclass relationships.
    - **Example**: "Dog" is a subclass of "Animal."
  - **Homographs**: Handling of homonyms (terms with identical spellings but different meanings).
    - **Example**: "Bank" (a financial institution) vs. "Bank" (the side of a river).
  - **Associations**: Relations between similar or related concepts.
    - **Example**: "Wine" is related to "Grape."

A popular example of a lightweight ontology is the "Friend of a Friend" (FOAF) ontology, which defines relationships between people in social networks.

2.2.2 Taxonomies

Taxonomies provide a hierarchical system of grouping concepts or entities. The term originates from the Greek words *taxis* (order, arrangement) and *nomos* (law, science). Types of taxonomies include:

- **Informal IS-A Hierarchy**: An explicit hierarchy of classes where subclass relations are not strict. For example, the index of a library.
- **Formal IS-A Hierarchy**: An explicit hierarchy of classes with strict subclass relations.
- **Formal Instance**: A hierarchy that includes explicit class structures, strict subclass relations, and allows *instance-of* relations.

A taxonomy can be defined as a controlled vocabulary organized into a hierarchical structure. As seen in 3.

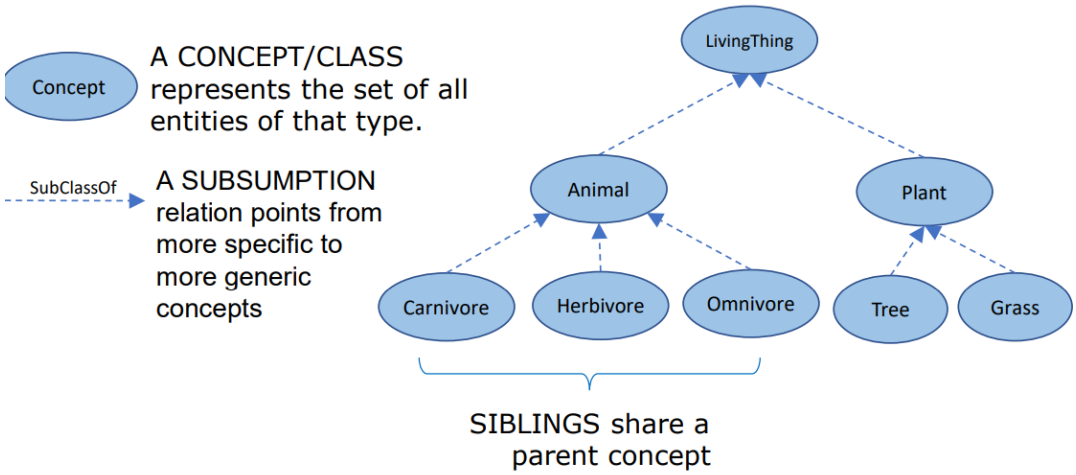

Figure 3: Taxonomy Example

Key concepts in formal taxonomies include:

- **SubClassOf**: A relation pointing from a more specific concept to a more generic concept.
- **Concept/Class**: Represents the set of all entities of a specific type.
- **Subsumption**: A specialization relation pointing from a specific concept to a generic one.

Subsumption/SubClass/Specialization Relation (in Formal IS-A Taxonomies) Semantics (Meaning): Subsumption in formal IS-A taxonomies ensures that if one class is a subclass of another, all instances of the subclass are also instances of the superclass. For example, if *Herbivore* is a subclass of *Animal*, all instances of *Herbivore* are also instances of *Animal*.

Common Mistake: Subsumption is often mistakenly used to represent other types of relations, such as *part-of* (meronymy), which denotes a part-to-whole relationship rather than a class hierarchy.

2.2.3 Heavyweight

Heavyweight ontologies provide a rigorous framework for defining and organizing knowledge with a strong emphasis on logic and formal specifications. Key characteristics include:

- **Rigorous Definition and Organization of Concepts**: Concepts and their relationships are precisely defined, ensuring clarity and reducing ambiguity.
- **Focus on Logic**: Heavyweight ontologies prioritize formally correct deductions and inferences, enabling reliable reasoning and decision-making.
- **Careful Formal Specification**: A formal specification ensures consistency and eliminates contradictions within the ontology.

Heavyweight ontologies are often used in applications requiring advanced reasoning, such as artificial intelligence, semantic web technologies, and complex domain modeling. They aim to create a shared, consistent understanding of knowledge that can be processed by both humans and machines.

Examples of Heavyweight Ontologies

- **Frames**: Frames provide structured representations for defining classes and their properties.
  - **Example**: In a medical ontology, the frame for "Disease" might include properties such as "Symptoms," "Causes," and "Treatment."
- **Value Restrictions**: These enforce constraints on the values that properties can take.
  - **Example**: In a wine ontology, "Wine color" might be restricted to values like "Red," "White," or "Rosé."
- **General Logic Constraints**: These involve the application of logical rules to ensure consistency across the ontology.
  - **Example**: If "Animal" is defined as disjoint from "Plant," no instance can belong to both classes simultaneously.
- **Disjunctiveness**: Ensures that certain concepts are mutually exclusive.
  - **Example**: "Male" and "Female" in a gender ontology might be disjoint concepts.
- **Inversiveness**: Defines inverse relationships between properties.
  - **Example**: If "Parent" is a property, its inverse would be "Child."
- **Part-of Relationships**: These express compositional relationships.
  - **Example**: "Engine" is a part of "Car."

2.3 Ontology

Ontology is a **taxonomy** extended with additional relations and further constraints, providing a more comprehensive framework for modeling knowledge. Relations within an ontology are directed, pointing from a **Domain** concept to a **Range** concept. These relations can also include:

- **Inverse Relations**: Relations that allow bidirectional reasoning. For example, if "Tom eats Jerry," the inverse relation could be "Jerry is eaten by Tom."
- **Disjoint Concepts**: Concepts that describe non-overlapping instance sets, ensuring that instances belong to distinct categories. For example, *Carnivore* and *Herbivore* may be disjoint concepts.

2.3.1 Mechanics

The mechanics of an ontology focus on the logical and structural rules governing the relationships and organization of concepts, as seen in figure 4:

- Establishing the **Domain** and **Range** of relations to clarify which concepts are linked.
- Defining and maintaining **consistency** among concepts, relations, and instances.
- Implementing **constraints**, such as disjointness or cardinality, to refine the ontology's structure and enforce rules.

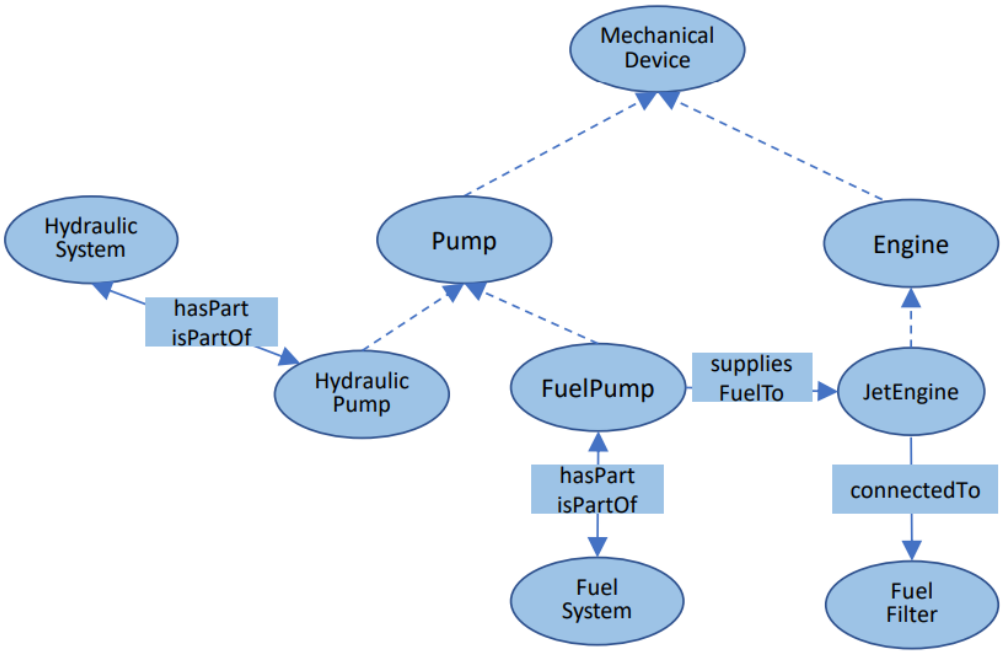

Figure 4: Ontology Mechanics

2.3.2 Instances/Entities

Instances or entities represent individual objects in the universe of discourse. They provide specific examples or metadata tied to the defined concepts in an ontology. This can be seen in 5.

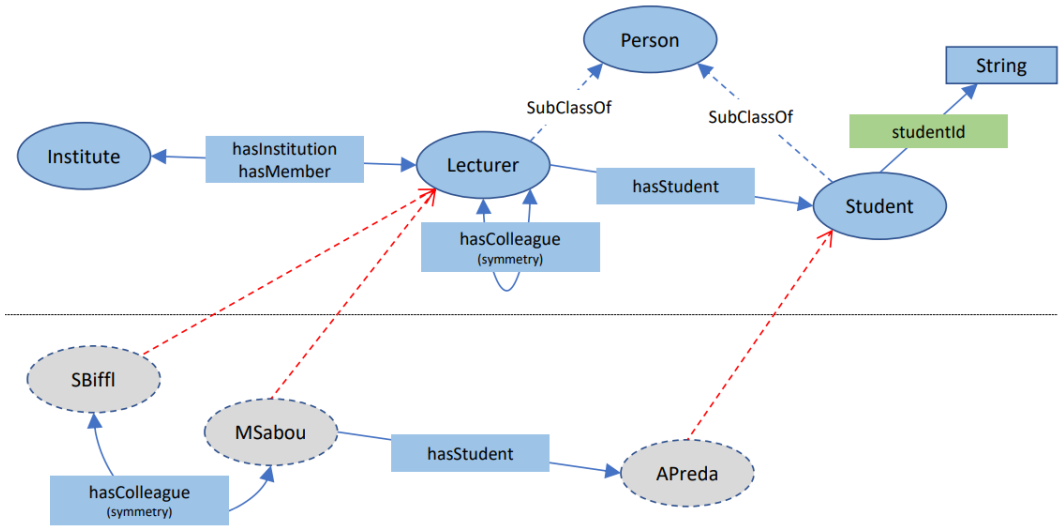

Figure 5: Ontology instances

Key aspects include:

- **Instance Representation**: An instance corresponds to a real-world entity and is associated with one or more concepts. For example:
  - *Tom is a Carnivore.*
  - *Jerry is an Animal.*
  - *Tom eats Jerry.*
- **Typing**: An instance is typed with respect to a concept, meaning that it is classified as being of a particular type. For example, "Tom" is an instance of the concept *Carnivore*.
- **Relations Between Instances**: Instances are connected through defined relations within the ontology, such as "eats" in the example above.

2.4 Description Logics

Description Logics (DL) are a family of formal knowledge representation languages used as the foundation for creating machine-readable ontologies. They are particularly suited for the Semantic Web and are based on a subset of First-Order Logics. DL provides a framework for representing and reasoning about knowledge in a structured and formal way.

2.4.1 Introduction

Description Logics aim to:

- Represent knowledge in a way that is both human-readable and machine-readable.
- Enable computers to reason with data and draw logical conclusions.
- Provide a formal basis for defining and organizing ontologies.

A formal, logic-based language in DL offers:

- **Syntax**: Defines which expressions are considered valid.
- **Semantics**: Provides the meaning of those expressions.
- **Calculus**: Defines how to compute and determine the meaning of expressions.

Semantic Web languages, such as OWL (Web Ontology Language), are built on Description Logics.

2.4.2 General DL Architecture

The general architecture of a Description Logic system (can be seen in 6), involves the following components:

- **Knowledge Base (KB)**: Composed of:
  - **TBox** (Terminological Knowledge): Contains definitions of concepts, attributes, and properties of a domain. For example:

    $$Writer \equiv Person \sqcap \exists author.Book$$

  - **ABox** (Assertional Knowledge): Contains information about specific individuals or entities. For example:

    $$Writer(GeorgeOrwell)$$

- **Inference Engine**: A component that reasons with the knowledge base to infer new information or check consistency.
- **Interface**: The mechanism for users or systems to interact with the ontology.

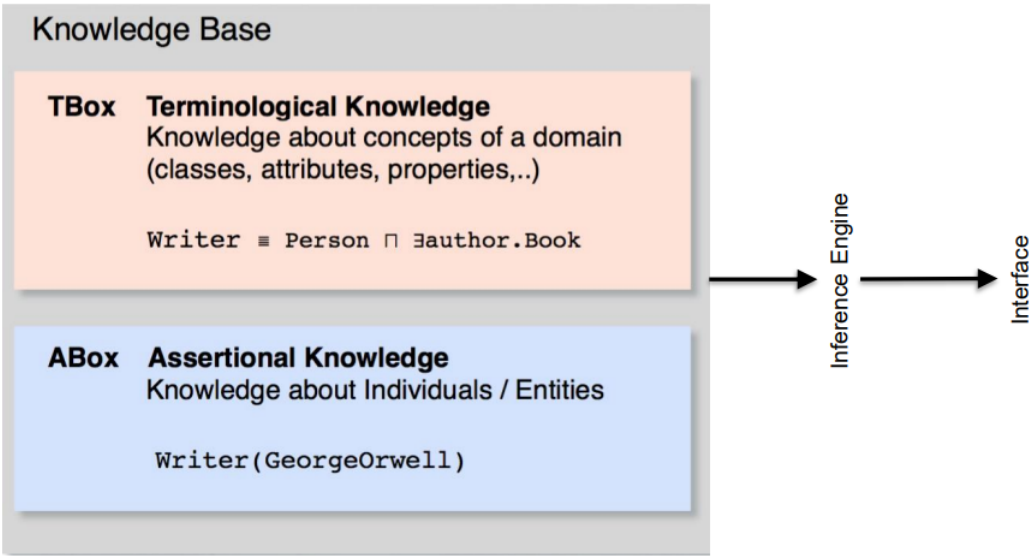

Figure 6: DL architecture

2.4.3 Notation / Syntax

The syntax of DL includes:

- **Atomic Types**:
  - Concept Names: $A, B, ...$
  - Special Concepts:
    - $\top$: Universal concept (all objects in the domain).
    - $\bot$: Bottom concept (empty set).
  - Role Names: $R, S, ...$
- **Constructors**:
  - Negation: $\neg C$
  - Conjunction: $C \sqcap D$
  - Disjunction: $C \sqcup D$
  - Existential Quantifier: $\exists R.C$
  - Universal Quantifier: $\forall R.C$

Examples:

- $Writer \equiv Person \sqcap \exists author.Book$
- $Novel \equiv Prose$
- $Novel \sqsubseteq Book$
- $Herbivore \sqsubseteq \neg\ Carnivore$
- $PetOwner \sqsubseteq Person \sqcap \exists hasPet.Animal$
- $Carnivore \sqsubseteq Animal \sqcap \forall eats.Animal$

2.4.4 Semantics

Description Logics rely on model-theoretic semantics, where an interpretation consists of:

- A **domain of discourse** (Δ): A collection of objects.
- **Interpretative Functions** (I): Functions mapping:
  - Classes (concepts) to sets of objects in the domain.
  - Properties (roles) to sets of pairs of objects.

In a DL, a class description characterizes the individuals that are members of that class, allowing precise reasoning and inference based on defined relationships and constraints.

It can be better seen in 7:

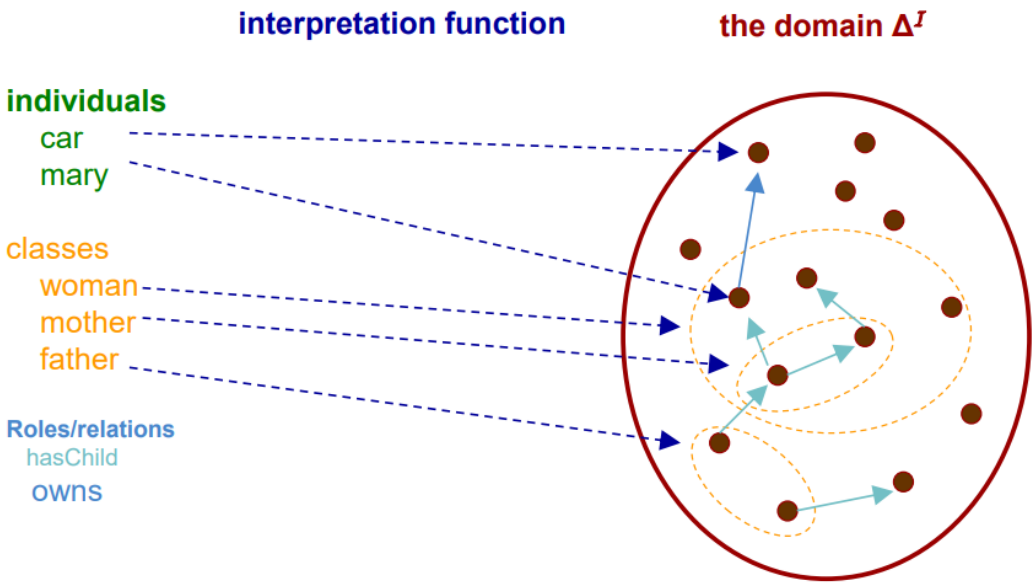

Figure 7: DL semantics

2.5 How to Build an Ontology

Building an ontology involves several steps, each addressing specific aspects of knowledge representation and organization. This process is iterative and not strictly linear.

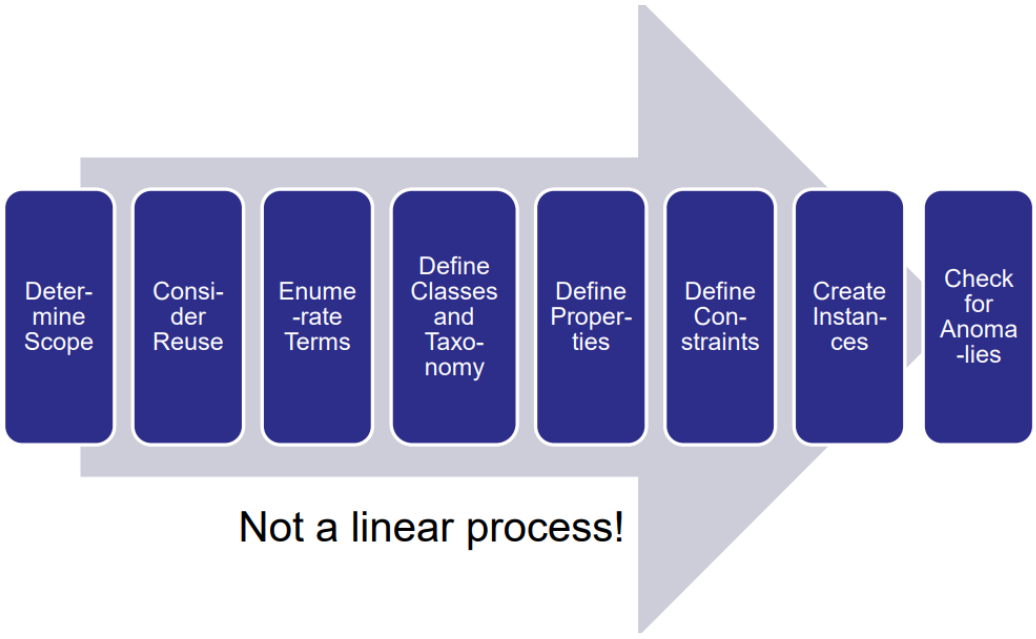

Figure 8: Ontology engineering process

2.5.1 1 - Determine Scope (Competency Questions)

The scope of the ontology should be determined by its intended use and anticipated extensions. Key questions to address include:

- What is the domain that the ontology will cover?
- For what purposes will the ontology be used?
- What types of questions should the ontology answer? (Competency Questions)
- Who will use and maintain the ontology?

Example Competency Questions for a Wine Recommender System:

- Which wine characteristics should I consider when choosing a wine?
- Is Bordeaux a red or white wine?
- Does Cabernet Sauvignon go well with seafood?
- What is the best choice of wine for grilled meat?

2.5.2 2 - Consider Reuse

Reusing existing ontologies saves effort, improves interoperability, and leverages validated ontologies. Sources for reusable ontologies include:

- Protégé Ontology Library (`http://protege.stanford.edu`)
- NCBO Bioportal (`http://bioontology.org`)
- Linked Open Vocabularies (`https://lov.linkeddata.es`)

2.5.3 3 - Enumerate Terms

Write down all relevant terms expected to appear in the ontology. This includes:

- **Nouns**: Basis for class names (e.g., wine, grape, winery).
- **Verbs or Verb Phrases**: Basis for property names (e.g., hasColor, madeFrom).

Examples for a Wine Ontology:

- Terms: wine, grape, winery, region, sugar content.
- Properties: hasColor, locatedIn, madeFrom.

2.5.4 4 - Define Classes and a Taxonomy

Classes represent concepts in the domain and are organized into a taxonomic hierarchy:

- A class is a collection of entities with similar properties (e.g., red wine, winery).
- Subclasses inherit properties from their superclasses.

Example:

- *Red Wine* is a subclass of *Wine*.
- Every instance of *Red Wine* is also an instance of *Wine*.

2.5.5 5 - Define Properties

Properties describe:

- **Attributes** of instances (e.g., color, sugar content).
- **Relationships** between classes (e.g., Wine *hasMaker* Winery).

Properties should:

- Be specified in the highest possible class in the hierarchy.
- Follow domain (parent class) and range (child class) rules.

2.5.6 6 - Define Constraints

Constraints refine the ontology by adding logical rules:

- **Cardinality**: Specifies the number of values a property can have (e.g., each wine has exactly one maker).
- **Existential Restrictions**: Ensures that at least one property exists with a specified value (e.g., wines are located in regions).
- **Universal Restrictions**: Ensures all values of a property are of a certain type (e.g., wines can only be made in wineries).
- **Property Characteristics**: Includes symmetry, transitivity, inverse properties, and functional values.

Examples:

- Wine $\sqsubseteq \geq$1madeFrom.Grape
- Wine $\sqsubseteq \leq$1hasMaker

2.5.7 7 - Create Instances

Instances represent individual entities in the domain:

- The class becomes the direct type of the instance.
- Instances inherit properties and constraints from their class hierarchy.
- Property values assigned to instances must conform to constraints.

Example for a Wine Ontology:

- *Chateau Margaux* is an instance of *Wine*.
- *Chateau Margaux hasMaker Margaux Winery.*

2.5.8 8 - Check Anomalies

Use the formal semantics of the ontology to:

- Validate the model:

  - Check for consistency.
  - Ensure satisfiability of concepts and properties.
- Derive additional knowledge:
  - Automatically classify instances.
  - Infer new relationships or properties.

2.6 Reasoning

Reasoning is one of the key functionalities provided by ontologies due to their formal grounding in logic. It enables the inference of new knowledge based on already declared information. Reasoning is performed using specialized software called reasoners or inference engines.

2.6.1 Capabilities of Reasoning with Ontologies

- **Consistency Checking**: Ensures that the ontology model is free from logical contradictions.
- **Class Inference**: Discovers relationships between classes that were not explicitly declared.
- **Instance Inference**: Determines class membership of instances based on defined relationships and constraints.

2.6.2 Consistency Checking

Consistency checking ensures that the ontology does not contain logical errors:

- Example:
  - Declared: Animal $\sqsubseteq \neg$Plant
  - Declared: Plant(myAlgae)
  - Inferred: Animal(myAlgae)
  - Result: An inconsistency because myAlgae cannot be both an Animal and a Plant.

2.6.3 Satisfiability Checking

Satisfiability checking identifies concepts or classes that cannot possibly have any instances:

- Example:
  - Declared: Algae $\sqsubseteq$ Plant, Animal $\sqsubseteq \neg$Plant, Algae $\sqsubseteq$ Animal
  - Result: The class Algae is unsatisfiable because it cannot belong to both Animal and Plant simultaneously.

2.6.4 Class Inference

Class inference discovers new relationships among classes based on given knowledge:

- Example:

  - Declared: Carnivore $\sqsubseteq$ Animal $\sqcap \forall$eats.Animal
  - Declared: Cat $\sqsubseteq$ Animal $\sqcap \forall$eats.Mouse, Mouse $\sqsubseteq$ Animal
  - Inferred: Cat $\sqsubseteq$ Carnivore

2.6.5 Instance Inference

Instance inference identifies class membership for specific entities:

- Example:
  - Declared: hasPet(Person, Animal), PetOwner $\sqsubseteq$ Person $\sqcap \exists$hasPet.Animal
  - Declared: Person(Pete), Animal(Spike), hasPet(Pete, Spike)
  - Inferred: PetOwner(Pete)

2.6.6 Class + Instance Inference

Combining class and instance inference allows the discovery of both new classes and instance memberships:

- Example:
  - Declared:
    * hasPet(Person, Animal)
    * PetOwner $\sqsubseteq$ Person $\sqcap \exists$hasPet.Animal
    * PetLover $\sqsubseteq$ Person$\sqcap >$ 3hasPet.Animal
  - Declared:
    * Person(Pete), Animal(Tom), Animal(Jerry), Animal(Spike)
    * hasPet(Pete, Spike), hasPet(Pete, Tom), hasPet(Pete, Jerry)
    * $\{Spike\} \sqsubseteq \neg\{Tom, Jerry\}$
  - Inferred:
    * PetOwner(Pete)
    * PetLover(Pete)

Reasoning enables powerful, logic-based conclusions to be drawn from the ontology, enriching both its usability and reliability.