Chapter 2

Large-Scale Knowledge Representation: Knowledge Graphs

As mentioned in Chapter 1, knowledge graphs are these huge sets of information and data that get combined. These kinds of knowledge representation are widely used in practice. One example we gave is the Linked Open Data Cloud.

In this chapter, we will look at what a knowledge graph is and how we can describe it with vocabularies, syntaxes, and semantics. Then, we will discuss how schemas, such as RDFS and OWL2, can help us reason on knowledge graphs and how we can define knowledge graphs with these two standards.

This chapter is based on the Knowledge Graph Survey [Hogan et al., 2021], slides by my former postdoc and now Professor at Aalborg University Daniele Dell'Aglio, as well as my own material.

2.1 Introduction and Motivation

In the "real" world, there are two ways to push "intelligence": Smart agents and smart data. **Smart agents** are agents that use sophisticated algorithms, such as natural language processing, machine learning, and deep learning, to process data. These agents generally take some question, deconstruct it into its parts, turn it into a query towards some data set, and then return the result of this query. Another way is to construct a data set that encodes precisely the answers we need, so that every question becomes a simple lookup. This is the **smart data** approach. This second approach, does, however, require to precompute answers to all possible questions in the universe.

Casting these alternatives as a dichotomy does not take into account that these approaches can be combined to a certain extent. For instance, precomputing answers to all possible questions is probably unfeasible. However, building an agent that can untangle all questions to provide an answer might also be unfeasible. It, hence, may make make more sense to combine these two approaches in some way.

Example 2.1 (Siri)**.** If we ask Siri if penguins fly, then it will probably return some article or other piece of knowledge which it thinks is relevant. Then it relies on the users' intelligence to determine if the found piece of information really answers the question. In other words, Siri often finds you a place where the answer is likely to be, instead of answering the question directly.

However, for other questions, Siri can directly provide an answer. When asking Siri what the capital of Switzerland is, it will correctly answer Bern.

2.1.1 The Semantic Web: Core Ideas

To provide more information that can be used to answer questions directly, Tim Berners-Lee suggested in 1998:[1]

> The Semantic Web is a web of data, in some ways like a global database.

There has been for a long time the idea and vision that we can take all the databases and combine them to create a single database that contains all knowledge of humanity. Having such a database would enable and/or simplify new automatic operations such as:

- **Search**: Manage synonyms and homonyms, consider the context of the query
- **Personalization**: Tailor the content of a web site based on agents' descriptions
- **Linking**: Dynamic decisions about the paths to follow to navigate web pages
- **Integration**: Collect information among different web sites and process it without mental copy-paste operations

[1] https://www.w3.org/DesignIssues/Semantic.html

To attain this goal, the key question now is how we can combine these different databases, as each one is potentially written in a different logic in terms of its database system. It took inspiration in the traditional Web: the Web is based on the technology of HTTP and HTML (Hypertext Markup Language), the former being a protocol for requesting pages, and the latter being a markup language for text. Importantly though, the content is organized according to the presentation of information (titles, tables, lists, highlights), rather than the actual structure of the information. Often, web pages are usually generated from structured data, e.g. databases or content management systems.

Example 2.2 (IMDb: Structured Data and Unstructured Presentation)**.** If we navigate to the IMDb page for the movie "Inception" as shown in Figure 2.1[a], then we will get some textual information about the movie. As we can see not all of this is free text, there is some structure in this text. The structure is often implied by typography. Specifically, one can find entities and their relations (as shown in Figure 2.1 on the right). For instance, the entity `:Nolan` is a `:director` of the entities `:Inception` and `:Interstellar`. Both of these are of `:type :Movie`.

[a] https://www.imdb.com/title/tt1375666

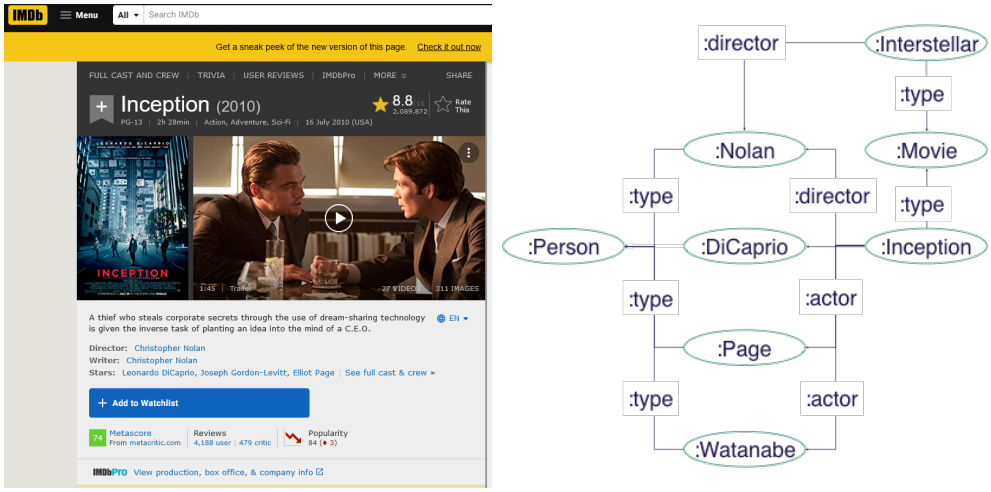


Figure 2.1: IMDb page of the movie "Inception" with the underlying typography. (source: Daniele Dell'Aglio)

How can we *use* these semi-structured data rather than "just" publishing them as formatted text? To that end, consider the example discussed in Example 2.3 and Figure 2.2

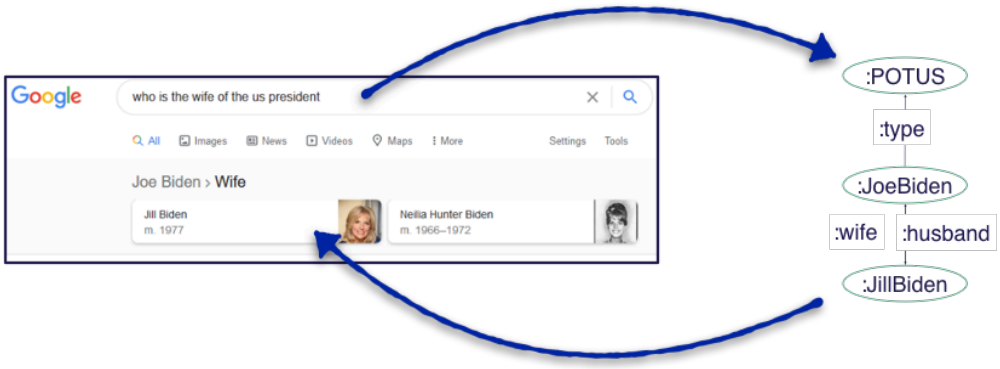


Figure 2.2: Google query accessing a knowledge graph. (Source: Daniele Dell'Aglio)

Example 2.3 (Using a Knowledge Graph for a Google Query.)**.** Google uses explicit knowledge for answering search queries. If we search for "who is the wife of the us president", as depicted in Figure 2.2, the search engine then can map the text "us president" to the entity `:POTUS` (acronym for 'President of the United States'). Then, we can access the graph and follow the relation `:type` to arrive at the entity `:JoeBiden`. We can then follow the relation `:wife` to get to the entity `:JillBiden`. Sure enough in the search query we see Jill Biden and also Joe Biden's first wife.

We can also use such knowledge representations in e-commerce (see Example 2.4 and Figure 2.3):

Example 2.4 (Filter Clothes Based on a Knowledge Graph)**.** Zalando uses a knowledge graph in order to filter search results to some kind of criteria without having to go over all products and classifying each one separately. Figure 2.3 presents an example graph that can categorizes materials like wool, silk, and leather and relates them to the entity `:AnimalBasedProduct` with the relation `:typeOf`. Then the graph specifies that these are not suitable for vegans. By now querying the graph for the some shirt, say `:ShirtX`, we can get the `:type` of each individual

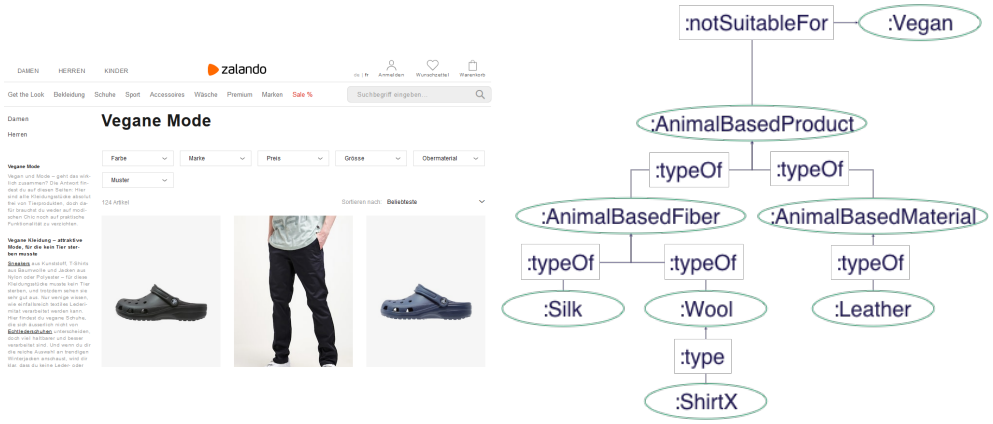


Figure 2.3: Zalando using a knowledge graph for filtering clothing to determine if it is vegan. (Source: Daniele Dell'Aglio)

materials and then we can determine if the shirt is vegan or not. We can now do that for every piece of clothing that is represented in this graph.

Another application of knowledge graphs we touched upon in Chapter 1 are rich snippets, such as the one depicted in Figure 1.10. Lastly, Facebook uses knowledge graphs to provide a set of very likely answers and responses for its chat bot, similar to how Siri uses it.

Side Note It is often assumed that the "only" thing that needs to be correct for a Web of Data to work is the technology. Obviously, this is wrong. Indeed, there are some arguments rooted in economics, why having one big database is likely to fail [Van Alstyne et al., 1995]. Whilst the social prerequisites are now researched and there are publications in outlets such as WebSci or the Web Conference, the economic aspect has been largely overlooked (see [Grubenmann et al., 2018] for a notable exception).

2.1.2 From the Web to the Semantic Web: A Quick Overview

In a first step this sub section provides a high-level overview to get an intuition of what the individual components do. In a second step we are going to dive into the theory of these components, answering questions like why they are the way they are and how they get structured logically and mathematically.

The Web of Data and Waterloo Let's consider some Wikipedia page, e.g., about the Battle of Waterloo, as depicted in Figure 2.4. This webpage clearly shows text but we can detect elements of structured data, such as a date, which is made up of a Weekday, the day, a month, and a year. Also, we can see a place, a person, and many more. All we need to do now, is annotate the text such that we know that "18" refers to a day, "Waterloo" is a place, "Napoleon" is a person, etc. Performing a large scale data exchanges requires

- an agreement on standard syntaxes to represent data and metadata,
- an agreement on vocabularies to express how the data is represented and structured, and
- publication of large amounts of data in these standard syntaxes and compliant to the agreed vocabularies.

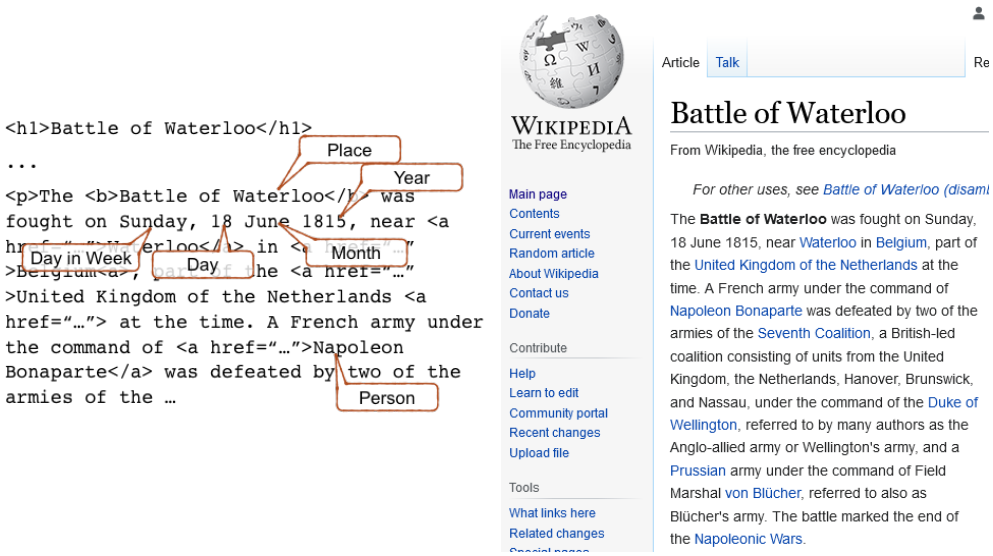


Figure 2.4: HTML and rendered Wikipedia page for the Battle of Waterloo.

Additionally, the Web is distributed with respect to three dimensions:

- Content
- Location
- Ownership

In other words, we have some content that is located on some server which is owned by someone. The Semantic Web inherits these core pillars, but given that we are dealing with data rather than text, we now need to ask ourselves how we deal with those. For instance, a standard database is located on someone's server and is, usually, self-contained: i.e., it assumes that it has all the information it needs and, therefore, does not link to any outside recourses. For this reason we need to go beyond this notion of standard databases and ask ourselves how we can deal with the ownership of this information if we want to access it from, say another database.

This is critical for the following reason: assume that person A publishes some information and our tool relies on linking to that information of person A. We now need to take into account that person A might at any point decide to take this information down (i.e., remove it from the Internet), so the question arises, just like in the Web: "How do you deal with HTTP 404 errors?"

The Semantic Web Technoloyg stack To achieve this functionality, the Semantic Web is built on a stack or Layer Cake of technologies.[2] This **Layer Cake of technologies** (see Figure 2.5) allows for the following:

- Consensus on small steps: people can agree on a set of layers (below of a certain level), but do not have to agree on higher layers
- Incremental adoption: From the bottom to the top
- Downward compatibility: agents are able to interpret the information at lower levels
- Upward compatibility: agents can (partially) use information at higher levels

This essentially means that each layer of technologies is detached from the other layers, where almost all of these technologies are W3C standards.

The specific set of technologies, as depicted in Figure 2.5, are the following: the character set used for encoding is UNICODE and URIs are used as identifiers—this is the Web after all. On top of that, they built a data interchange format called RDF, which relies on XML Syntax. To define taxonomies (i.e., a. kind of type and relationship hierarchy) over RDF, RDFS

[2]This follows the age-old computer science tradition of building layers of abstractions on layers of abstraction. Or as the quote attributed to David Wheeler says: "Any problem in computer science can be solved with another level of indirection" see https://en.wikipedia.org/wiki/Butler_Lampson#Quotes

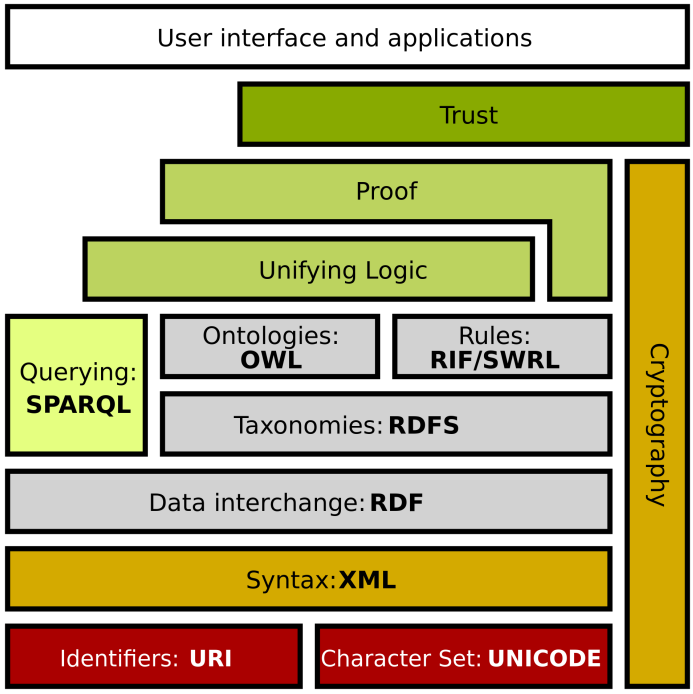
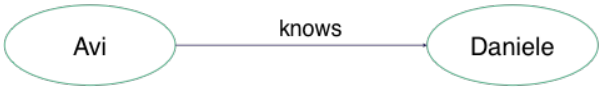


Figure 2.5: Abstract technology stack used for the Semantic Web. (Source: user:Marobi1 at Wikimedia Commons, https://commons.wikimedia.org/wiki/File:Semantic_web_stack.svg, "Semantic web stack", CC0_01.0)

and its extension OWL are used. To express more complex relationships, RIF/SWRL are typically used. The query language used for the Semantic Web is SPARQL, which is similar to the well-known query language SQL. It differs from SQL, as it operates over graphs rather than tables. Note that we will not discuss the remaining layers, as they are seldom used in today's applications. In the following we will highlight the most important standards and discuss why it makes sense to use them.

The Resource Description Framework (RDF) One of the basic technologies used is the Resource Description Framework (RDF) is a way of interchanging data in a semi-structured way on the Web. It assumes that the basic structure of your data set is a graph, unlike relational databases that assume the basic data structure is a table. The reason is that a graph is a much more flexible data structure than a table (as it allows data to be in a non first normal form). I am unaware of a data structure in computer science that one cannot represented in the form of a graph with some extensions as we will see later.

With a graph we can markup any kind of information in a semi-structured way. In RDF, objects are nodes and relations between those objects are edges. We can now make an atomic statement "Avi knows Daniele" as follows:[3]



The beauty of this approach is we have not needed any relational schema for this.

Identifiers as URIs The second important technology we are going to look at are URIs. URIs are used as identifiers on the Web, which also allow to convey location if desired. A URI is used to identify entities or relationships. E.g. we can have the following URIs for the above example:[4]

We have now introduced identifiers and a relation. But note how we have also quietly introduced the standard vocabulary "FOAF"[5] in the URI of the relation. FOAF is designed to describe people and their relationships with each other.

Tim-Berners Lee 's Principles for Linked Data Publishing Tim Berners-Lee's vision of linked data is based on **four principles** [6]:

- "Use URIs as names for things.
- Use HTTP URIs so that people can look up those names, meaning people should be able to request it via HTTP.
- When someone looks up a URI, provide useful information, using the standards.
- Include links to other URIs, so that users can discover more things."

When all these conditions are met, and the data is open [7], we can publish data in a **5-star deployment scheme**[8] as depicted in Figure 2.6. This

[3]Figure by Daniele Dell'Aglio
[4]Figure by Daniele Dell'Aglio
[5]https://en.wikipedia.org/wiki/FOAF_(ontology)
[6]Linked Data https://www.w3.org/DesignIssues/LinkedData.html
[7]The Open Definition https://opendefinition.org/
[8]https://5stardata.info/en/

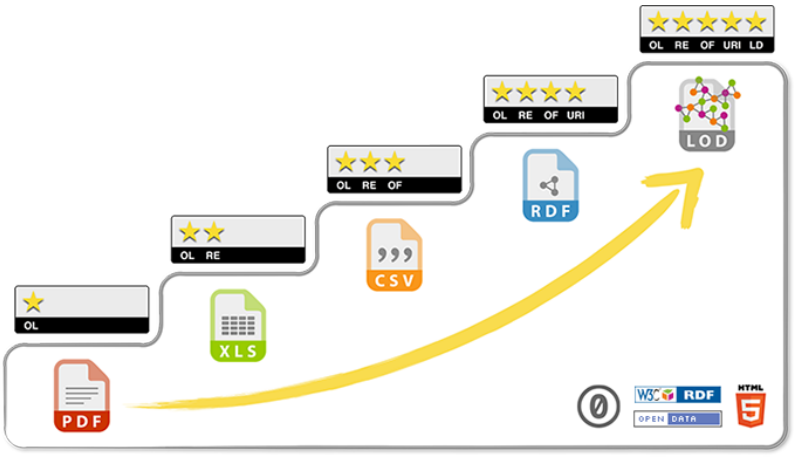


Figure 2.6: 5-Star deployment scheme for open data. (Source: https://5stardata.info/en/)

provides a rating on how beneficial the data is (i.e., the more stars you have, the more relatable your data becomes and the more processable it is in a large scale). The stars are given as follows:

1. **publish** your data
2. publish your data in a **structured format**
3. publish your data in an **open** structured format, rather than a proprietary formal like Microsoft Excel
4. publish your data in **RDF**
5. **link** your data to other data

Given these specifications, we can now assess, what standards Switzerland's Open Data site[9] conforms to. Additionally, thanks to these technologies, we can perform a data set search[10].

SPARQL The next important technology is the language **SPARQL** which is a query language designed to access the data. SPARQL is a query language akin SQL, but it is adapted to querying graphs (or graph fragments) rather than tables. We will look at SPARQL in a later section of this chapter (see Subsection 2.3.1 on page 43)

[9] https://opendata.swiss/en
[10] https://datasetsearch.research.google.com

2.2 Knowledge Graphs

Now that we have a high-level understanding of the Semantic Web and linked open data, we can introduce the concept of a **knowledge graph (KG)**. A knowledge graph is a graph, where nodes represent entities, and edges represent relationships between those entities. It is assumed to be a **directed edge-labelled graph**, unless mentioned otherwise. A KG differs from a graph database in that a KG is a database but also contains a vocabulary and a meaning to those vocabulary items.

A KG has certain characteristics according to [Paulheim, 2017]

- mainly describes real-world entities, their classes (or types) and their interrelations, organized in a graph
- defines possible classes and relations of entities in a schema
- allows for potentially interrelating arbitrary entities with each other
- covers various topical domains

In other words a KG requires real-world entities[11] (e.g., Avi or Daniele) and multiple domains (e.g., the relation knows is a relation between two entities of type Agent). These types/classes and relationships are often organized into taxonomies. Examples of KGs are: Wikidata, Google's KG, YAGO, DBPedia, etc.

Note that this is not a very precise definition. Indeed, all of the available definitions are a little complementary to each other. The general accepted connotation is that it includes entities, relationships between entities, and not just instance-based data, but also schema-based data (i.e., types/classes and their relationships) .

[Noy et al., 2019] define a KG from an industry and practical perspective as follows:

- "A knowledge graph describes *objects of interest* and *connections between them.*
- Knowledge graphs and similar structures usually provide a *shared substrate of knowledge within an organization*, allowing different products and applications to *use similar vocabulary* and to *reuse definitions and descriptions* that others create. Furthermore, they *usually provide a compact formal representation* that developers can use to *infer new facts and build up the knowledge.*"

[11] sometimes also called instances

Hence, the standard example for relational databases with students and classes and how they relate can also be put into a knowledge graph, where the main difference is that the data is represented as a graph, rather than tables. So, we could have such a graph for the University of Zurich and reason on this graph. However, other universities, such as the University of Bern, might also have graphs that represent the same thing, but their representation might differ. We could then relate our representation with their representation, creating a meta model that connects (and relates) the two. We can do this because the meta model, the model describing the different relationships is also a graph (i.e., we do not just relate entities directly, but also relate things that describe relationships between these entities).

Now let us define Knowledge Graphs formally:

Definition 2.1 (Directed Edge-Labelled Graph)**.** A directed edge-labelled graph, i.e., a tuple $G := (V, E, L)$, where $\mathbb{C}$ is a set of of constants, $V \subseteq \mathbb{C}$ is a set of nodes, $L \subseteq \mathbb{C}$ is a set of edge labels, and $E \subseteq V \times L \times V$ is a set of edges.

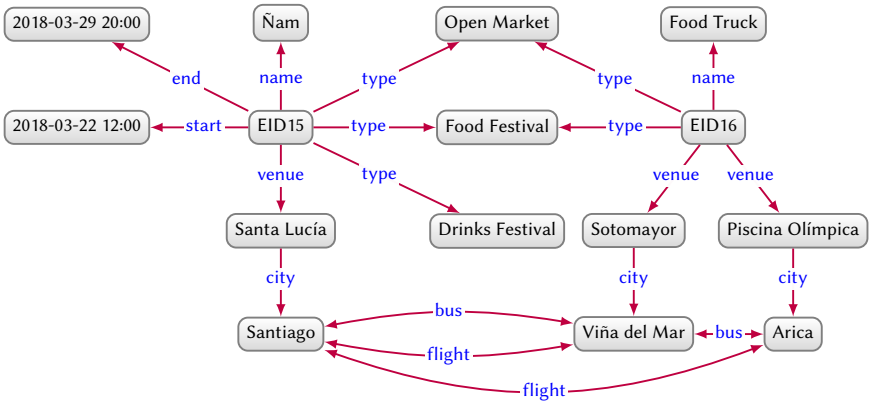


Figure 2.7: Directed edge-labelled graph describing events and their venues. (Source: [Hogan et al., 2021])

Example 2.5 (Directed Edge-Labelled Graph)**.** We are now going to look at what such a directed edge-labelled graph can look like. Consider the directed edge-labelled graph depicted in Figure 2.7, it shows the entity `EID15` which starts on the 22nd of March, 2018, ends on the 29th

of March, 2018, and has the name Ñam. `EID15` has `type Open Market`, `Food Festival`, and `Drinks Festival`.

As we know, we have a set of constants $\mathbb{C}$, which get used to describe the nodes as well as the edge types. So, we can see each edge annotated with `type` refers to the same constant.

If we now wanted to make this graph Web friendly, according to Tim Berners-Lee, we first need to describe the set of entities and relationships $\mathbb{C}$ via URIs (or, where appropriate, literals). Secondly, we need to agree on a syntax to publish this graph (e.g. using RDF).[a]

[a] One might be confused why RDF is called a syntax here. Indeed, as we will see in subsection 2.2.2, RDF is indeed a format for describing typed graphs, which can be serialized in a XML-style.

Notice Sadly the the recording for a good part of the rest of this chapter is missing. We will attempt to write this part after having taught the class in the Fall of 2022.

Much of the content of this chapter can be found in [Hogan et al., 2021]. So please refer to that article.

2.2.1 Basic Definitions

2.2.2 Practical Considerations

2.2.3 Syntax Issues

2.3 Querying Knowledge Graphs

2.3.1 Practical Considerations: SPARQL

2.4 Schemas

So far we have seen how we can represent knowledge with graphs. This includes object and relations between them, but we never gave the relationships meaning beyond saying "`knows` is used in some other vocabulary". Now, we are looking at how to systematically deal with such elements that give us a vocabulary, which then allows us to describe the world.

We can achieve this by using a **schema**. Schemas are the languages of things that sit on top of the RDF in our tool stack (see Figure 2.5), these deal with taxonomies (RDFS), ontologies (OWL), and rules (RIF/SWRL).

Example 2.6 (Schema for the Battle of Waterloo)**.** Again we consider the Wikipedia article for the Battle of Waterloo, with the following text:

<p>The Battle of Waterloo was fought on Sunday, 18 June 1815, near Waterloo in <a href="...">Belgium<a>, part of the <a href="...">United Kingdom of the Netherlands <a href="...">> at the time. A French army under the command of >Napoleon Bonaparte was defeated by two of the armies of the ....

Lots of the words in here have a meaning and a relationship to each other (see Figure 2.8). Reading the HTML, we can infer certain entities and relationships. E.g., Waterloo, Belgium, and United Kingdom of the Netherlands are all places, which in some way are concrete entities. Then there are abstract classes such as a `Battle`, which happens `at` a `Place` and `is also` a `Historic Event`, which are relationships. The abstract classes and their relationships are what we call a schema.

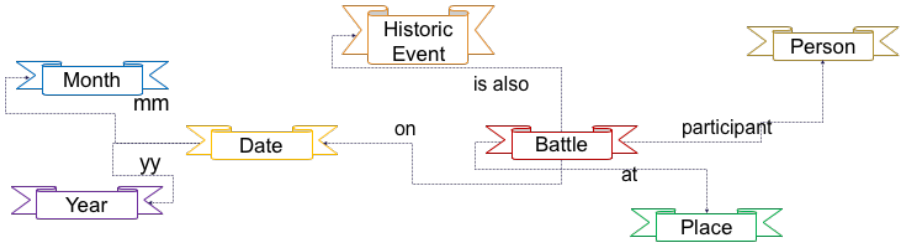


Figure 2.8: Example schema for the Wikipedia article on the Battle of Waterloo. The colors chosen correspond to the colors of the HTML in Example 2.6.

The question now is how we can model this domain and its relationships in a way that would adhere to the approaches that we have looked at for RDF and so forth. The way we do this is with **ontologies**. Note that the meaning of the word "ontology" in this context refers to its usage in computer science, logic, and AI. This should not be confused with the notion of an ontology in philosophy.

Definition 2.2 (Ontology)**.** An ontology is an explicit formal specification—or model—of the concepts in a domain (i.e., ontologies describe the

things, usually called entities, that exist in a domain and the relationships in such a domain, in a formal way, such that we can use some logical reasoning about them). An ontology consists of two parts: The terminology box and the assertion box — both of which contain statements that are part of the knowledge base.

Definition 2.3 (Terminology Box (TBox))**.** A terminology box (TBox) describes the domain model, i.e., its entity types and their relationships. It contains **axioms**.

A TBox is similar to classes and methods in object-oriented programming (OOP) or an Entity-relationship model in a DBMS. It makes general statements about groups of entities (or classes) and their relationships. For example, a TBox could contain a statement of the form: `Student is_a Person`. This essentially describes that entities of type `Student` are also entities of type `Person`. Said differently, the class `Student` is a subclass of the class `Person` So this is not about a specific individual but a more 'general' rule.

Definition 2.4 (Assertion Box (ABox))**.** An assertion box (ABox) contains facts that describe a portion of reality, as pertaining to the domain. I.e., an ABox contains **facts** or **assertations**.

For example, `Dale instance Student` means that the person with given name "Dale"[a] is a student. This assertion makes a statement about an individual named `Dale`.

[a] see https://en.wikipedia.org/wiki/Dale_(given_name) for more on the name.

Example 2.7 (Gene Ontology)**.** In order to clarify how we can put a schema to use, we will look at an example of a gene ontology (see Figure 2.9). We can build a huge construct that describes the entities and their relationships in a domain. We can see, for instance, certain parts of the TBox of the gene ontology, such as the `metabolic process` and the `small molecule metabolic process`, where the latter has an `is_a` relation to the former. We can see how these resemble classes from Object-Oriented Programming (OOP) and how their relationships are analogous to multiple inheritance.

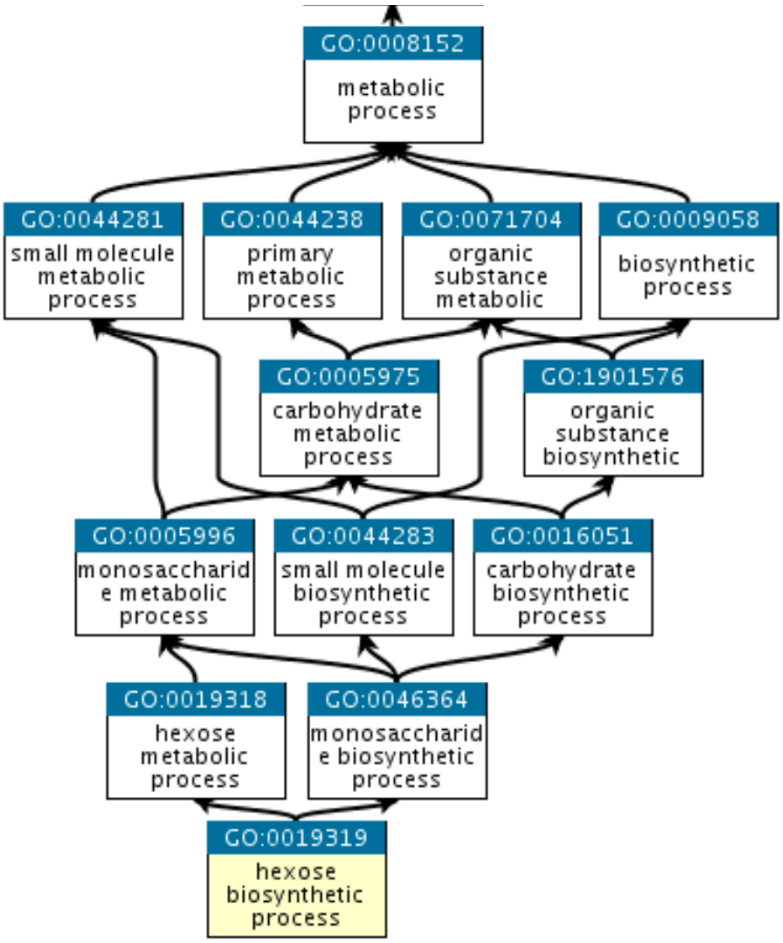


Figure 2.9: Gene ontology excerpt from http://geneontology.org/docs/ontology-documentation/.

Assume we want to store data that is within a domain. The first thing we would want to do is to check if there is already a model of that domain. By using the same model, the datasets can be merged very easily. One place to look up vocabularies is the website Linked Open Vocabularies[12], which contains over 750 vocabularies. Note that the use of the word vocabulary in this context can be seen as synonymous to the use of the word model, which is usually an ontology of some sort.

Figure 2.10 displays the homepage of the Linked Open Vocabularies initiative, which provides an index to some large vocabularies such as `dcterms`, which stands for "Dublin Core Terms" and is a vocabulary used to describe

[12] https://lov.linkeddata.es/dataset/lov/

documents. One of the first vocabularies is `FOAF` which stands for "Friend of a Friend" and is a vocabulary used to describe people and their relationships. `Schema.org` is a vocabulary initiative from Google, Microsoft, Yahoo, and Yandex for annotating web pages. Lastly, `CC` which is short for "Creative Commons" is a vocabulary that describes copyright licenses. There are many more than the ones mentioned here.

The beauty of these vocabularies being similar to classes, is that we can extend these if we want to specialize their meaning. For example, we could say that a professor at UZH is a subclass of a more generic professor and extend it with some special properties (such as the UZH-shortname). That is, we go from `uzh:professor` to `genericuni:professor` (subClassOf) and then extend the `uzh:professor` class with the property `uzh:shortname`.

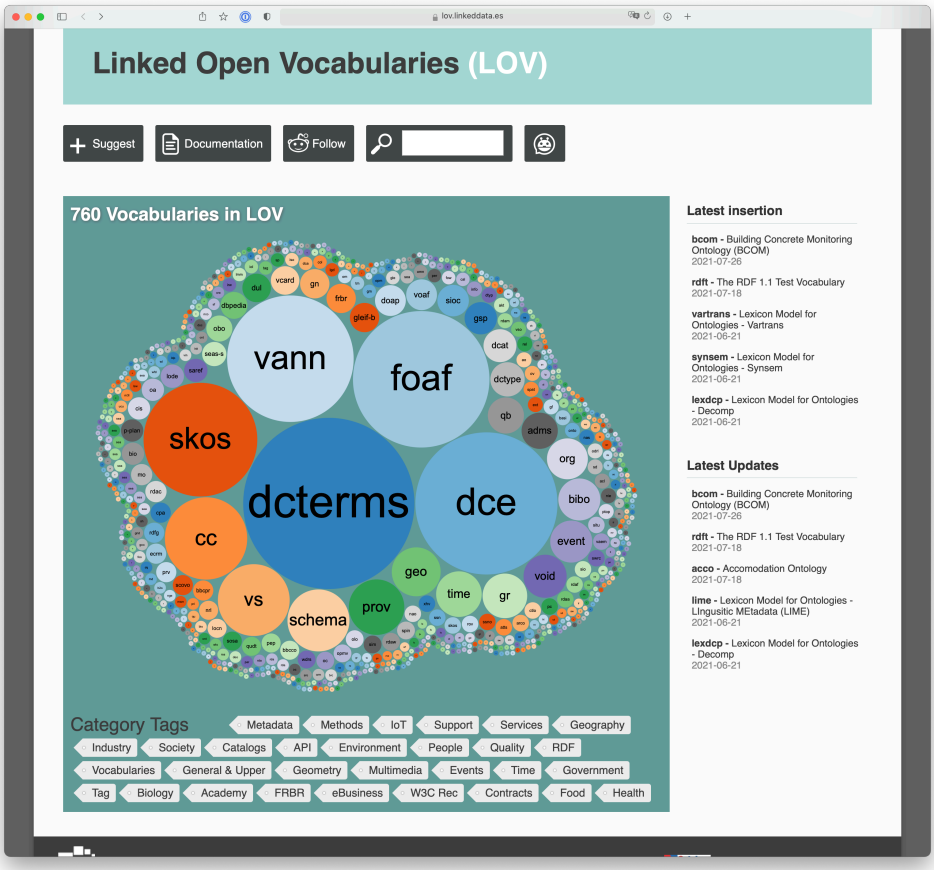


Figure 2.10: Web Page of the Linked Open Vacobularies (https://lov.linkeddata.es/dataset/lov/).

Given that we now know what ontologies can be used for, we need to define what is needed in an **ontology language** to build ontologies.

Definition 2.5 (Ontology Language)**.** An ontology language is a language that allows to express—or describe—an ontology. An ontology language requires the following elements:

- **A well-defined syntax**.
- **A formal semantics**, that describes the relationships between the elements in this well described syntax.

If we have meat the above two criteria, then we want the following three possible additional features:

- **Sufficient expressive power** for the domain that gets modeled. For example, we may want to be able to express complicated matters, such as "a human being usually has two legs." While this sounds easy to do, this requires a cardinality constraint, which is not necessarily easy to achieve in basic logic.
- **Convenience of expression**, which might result in a trade-off with respect to the sufficient expressive power.
- **Efficient reasoning support** by an engine that will allow us to reason or deduce information. For example, if we have a `Person` at the university that `takes Courses`, then we can infer that this person is a `Student`, because these are the only `Person`'s at universities that can take courses at a university. So we want an engine, which can pick that up.

In the remainder, we will look at two often used ontology languages: RDF Schema and OWL2. **RDF Schema (RDFS)**[13] is the simpler one of the two and it was introduced in the year 2000. It complements the RDF model with some basic concepts (e.g. classes and subclasses). It uses the namespace `rdfs`.[14] In Section 2.5.1 we will explore many relations that are part of this namespace in more detail. **OWL2**[15] has much more expressive power than RDFS and extends it with additional features, such as inverse properties. It

[13] https://www.w3.org/TR/rdf-schema/
[14] http://www.w3.org/2000/01/rdf-schema#
[15] https://www.w3.org/TR/owl2-overview/

was proposed as a W3C[16] standard and has five sub-languages with varying expressive power and, thus, varying complexity of reasoning. OWL2 was introduced in 2002 and uses the namespace `owl`.[17]

Definition 2.6 (Well-defined Language)**.** Well-defined means that the language can express everything in an unambiguous manner.

A **well-defined** syntax is necessary to let a machine process the information, just like in programming languages. RDF, RDFS, and OWL2 all have well-defined syntaxes. It turns out that one can express RDFS and OWL2 in RDF, so RDF is a model that we use for both, describing entities and their relationships.

Definition 2.7 (Formal Semantics)**.** A formal semantics describes the meaning of a statement precisely (i.e., it does not rely on subjective intuitions and does not allow different interpretations). Both, RDFS and OWL2 have well-defined formal semantics.

Statements or sentences that are written with a language with a well-defined syntax and a formal semantics can be interpreted. This interpretation—or inference via reasoning—can be performed by a human or a program.

Definition 2.8 (Knowledge Representation (KR))**.** A knowledge representation (KR) according [Davis et al., 1993] is most fundamentally

> "a surrogate, a substitute for the thing itself, used to enable an entity to determine consequences by thinking rather than acting', i.e., by reasoning about the world rather than taking action in it".
>
> "It is a set of ontological commitments, i.e., an answer to the question: In what terms should I—or the program—think about the world?"
>
> "It is a *fragmentary theory of intelligent reasoning*, expressed in terms of three components:

[16] https://www.w3.org
[17] http://www.w3.org/2002/07/owl#

i the representation's fundamental conception of intelligent reasoning,

ii the set of inferences the representation sanctions, and

iii the set of inferences it recommends."

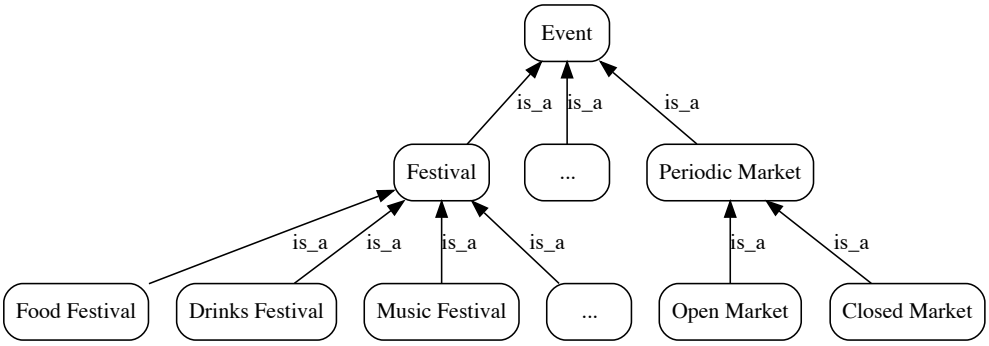
A knowledge representation is, thus, not the thing (i.e, the actual object in the world) itself, but a representation thereof. It is used to enable an agent to think about that thing. So if we `professors`, `students`, and `lectures` are classes with instances such as `Abraham Bernstein`, `Dale Doe`, and `Advanced Topics in AI` in a knowledge representation, then those instances are not the actual professor, student, or lecture themself, but an abstract representation we have in the computer.

We can use KR as a medium for pragmatic and efficient computation in the computational environment in which thinking is accomplished. In contrast to OOP however we don't "program" it (i.e., we don't execute the KR), but we use it as a model that then is used by programs to compute on them. Therefore, the focus is not on the execution (e.g., the methods) but on the knowledge and the modeling thereof. So, we can also use KR as a medium of human expression, i.e., a language in which we say things about the world. Therefore, KR is a part of both philosophy and AI.

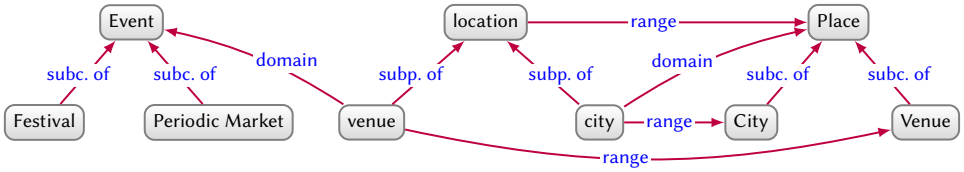
2.5 Knowledge Graph Schemas

One of the benefits of modeling data as graphs—versus, for example, the relational model—is the option to forgo or postpone the definition of a schema. However, when modeling data as graphs, schemata can be used to describe a high-level structure and/or semantics that the graph follows or should follow. As we discovered with RDFS and OWL, which can be expressed as RDF, which is essentially a graph representation, the structure (or schema) of a graph model can also be expressed as a graph.

In the following, we first discuss the general intuition behind **graph schemata** whilst introducing then basic building blocks of RDF. We then discuss some of the **basic assumptions for schemas** that need to be made when modeling. We continue with an **intuitive introduction of OWL**, which introduces a number of constructs that go beyond RDFS. We close with a discussion of **Description Logic** the formalism underlying OWL.



(a) Class hierarchy for the Festival/Market domain



(b) RDFS-inspired schema of the Festival/Market domain.

Figure 2.11: Two example schema graphs for the knowledge graph from Figure 2.7. (Source: [Hogan et al., 2021])

2.5.1 Semantic Schema for Knowledge Engineering

The intuition behind a semantic schema is **building a "class hierarchy" of things**. Recall Figure 2.7, which describes representations of some assertions, like `EID15` has `type Food Festival`. Figure 2.11 depicts two example schema graphs describing this domain. For example, we see in the first graph (Figure 2.11a) that a `Festival` is a subclass of an `Event` and a `Closed Market` is a `subclass` of a `Periodic Market`. These subclass relationships are indicated using the `is_a` arrow (indicating the relationship) in the graph. However, not all relations in a schema need to be an `is_a` relation. In the second example graph (Figure 2.11b) we see that a `venue` has a `domain Event`, while a `Venue` is a `subclass of (subc. of)` a `Place`. Hence, the `Venue` describes a place, where events can take place, whilst the identifier `venue` (note lowercase rather than uppercase) denotes a relationship between a `Event` (the `domain` of the relation), and a `Venue`, (the `range` of the relation). In other words, `venue` relates an event to the place where it will take place. Additionally, we see that the relation `city` is a `subproperty of` (`subp. of`) `location`. These are two examples related to a TBox, as they describe the terminology.

Feature	Definition	Condition	Example
Subclass	c —subc. of→ d	x —type→ c implies x —type→ d	City —subc. of→ Place
Subproperty	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

Figure 2.12: Core feature definitions for sub-class, sub-property, domain, and range features in semantic schemata (incl. RDFS) from [Hogan et al., 2021].

We can build a semantic schema with RDFS by using its predefined features of its semantic schema. These **RDFS features**—or descriptors—are shown in Figure 2.12. For instance, the **subclass** feature, `c subc. of d` means that its condition, `x type c` implies `x type d` holds. Mathematically,

$$\texttt{c subc. of d} \equiv \forall x \in c \implies x \in d \tag{2.1}$$

The feature **subproperty** `p subp. of q` means that if `x` has the property `p` in `y`, then also `x` has the property `q` in `y`. In other words, if `venue subp. of location` holds, then we know that for each relation `x venue y`, we know that also `x location y` holds.

For properties we can define the features **domain** and **range**. We say that `p` has the `domain c` if and only if some entity `x` having the relation `p` to `y` implies that `x` has the `type c`. This essentially tells us that the relation `p` is located in the `domain` of `c`. The feature range follows analogously, but for `y` instead of `x`. Note that this is analogous to the domain and the range of functions. These two allow us to type relationships.

Example 2.8 (Ontologies Intuition)**.** Recall Figure 2.7. Even though we do not have the relation `EID15 location Santiago` explicitly in the knowledge graph, we can infer it as follows using the second graph schema from Figure 2.11.

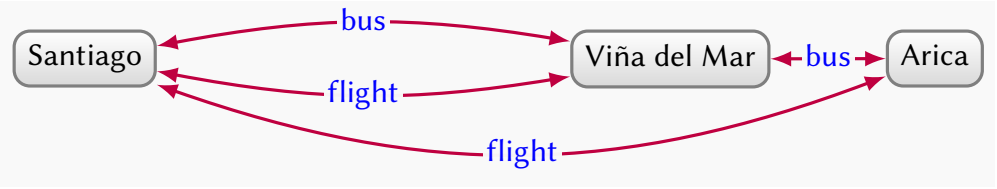
We know from the knowledge graph that `EID15 venue Santa Lucia` and `Santa Lucia city Santiago`. We also know from the schema that `venue subp. of location`. By the definition of the subproperty feature we know that `x venue y` implies `x location y`. Now substituting `EID15` for `x` and `Santa Lucia` for `y` gives us `EID15 location Santa Lucia`.

Similarly, we could infer that cities with flights should have airports nearby, even though this is also not explicitly contained in the KG.

Also, using the class hierarchy from Figure 2.11 we can infer certain properties about `Food Festival` and `Drinks Festival`, as the graph makes it clear that they are both a subclass of `Festival`.

2.5.2 Basic Assumptions for Schemas

Let us now consider some interesting questions and what inference we can make depending on our assumptions about the knowledge graph. Consider the following schema (again, from [Hogan et al., 2021]):



We can now ask the question "Is there a `flight` between `Arica` and `Viña del Mar`?" As we discussed previously, the way we reason about this question depends on our assumptions about knowledge:

- **Closed World Assumption (CWA)**, which assumes that all knowledge is in the KG.
- **Open World Assumption (OWA)**, which assumes that possibly not every piece of knowledge is in the KG.

If we assume the CWA, which is what is usually implicitly assumed when querying a traditional (relational) database, then we come to the conclusion that there is no flight, because between `Viña del Mar` and `Arica`, as we only have a bus connection expressed as a statement. However, this is not a very realistic assumption in the context of the Web. So, assuming the OWA, we only see that we do not have any evidence in our KG. Hence, we cannot say if there is or there is no such flight. Underlying is the difficulty of proving the non-existence of something, here a flight.

Another interesting question is "How many `flights` are there to `Santiago`?" The answer to this question depends on whether we assume if an item in the knowledge graph has a unique name or not:

- **Unique Name Assumption (UNA)**, which assumes that every entity has a unique name (i.e., a unique identifier for an entity).

- **Non Unique Name Assumption (NUNA)**, which assumes that there might be multiple names for the same entity.

Under the UNA we would argue that we have two flights. Under the NUNA we can only come to the conclusion that there is at least one flight, because we don't know if `Viña del Mar` and `Arica` are two names for the same entity and thus if the two flights in the KG are indeed the same flight. On the Web we should never assume that everything has one name only. For instance, there are multiple web pages describing "Abraham Bernstein," but some refer to him by the given name "Abraham" and some by the given name "Avi," which is short for Abraham. These web pages, thus, use two different names for the same person.[18]

2.5.3 OWL: Going beyond RDFS

RDFS provides a basis for schema definitions. But the language is quite limited. We will, therefore, now discuss OWL with its extensions, which allows more refined domain modeling. Note that this subsection aims at conveying an intuitive introduction to OWL, before the next subsection (2.5.4) conveys the formal underpinnings.

Ontology Features for Individuals/Entities

Now we introduce the rules that "apply" to the world we are trying to model. Figure 2.13 introduces the ontology features for individuals. This means that these are all features that can be used to reason about entities—or items/instances. The easiest way to understand are **assertions** is to assume that state facts about the entities. Assertions have the form `x` has the relation `y` to `z`, e.g., `Chile` has the `capital` `Santiago`. Alternatively, they can express **negations**, for which we need the `type` to be defined as `Neg` and—like with standard assertions—a subject (`sub`), `x`, the predicate (`pred`), `y`, and the object (`obj`), `z`. Additionally, we can also have **same as** and a **different from** relations. For example, `Abraham Bernstein` is `different from` `Abraham Lincoln` but `Abraham Bernstein` is `same as` the person described as `Prof. Abraham Bernstein, PhD.` on the UZH web page.

Ontology Features for Properties

Figure 2.14 depicts ontology features for property axioms. These are features that allow us to reason about properties (or relationships between en-

[18] Obviously, there are multiple Abraham Bernstein's. So, the problem is even worse, as the same name is also used for multiple people. But that is another problem.

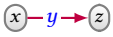
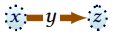
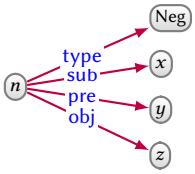
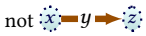
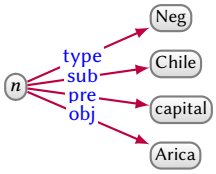
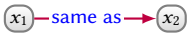
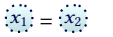
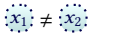
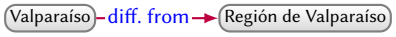
Feature	Axiom	Condition	Example
Assertion			
Negation			
Same As			
Different From			

Figure 2.13: Ontology features for individuals. (Source: [Hogan et al., 2021])

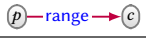
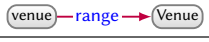
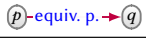
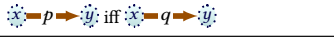
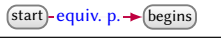
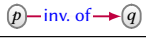
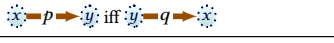
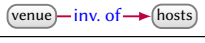
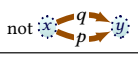
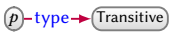
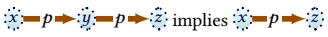
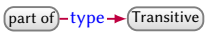
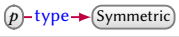
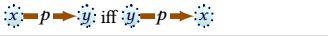
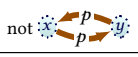
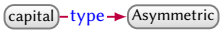
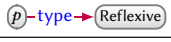
Feature	Axiom	Condition (for all x_*, y_*, z_*)	Example
Subproperty			
Domain			
Range			
Equivalence			
Inverse			
Disjoint			
Transitive			
Symmetric			
Asymmetric			
Reflexive			
Irreflexive			
Functional			
Inv. Functional			
Key			
Chain			

Figure 2.14: Ontology features for property axioms. (Source: [Hogan et al., 2021])

tities), as opposed to the entities themselves. These features—or properties of properties—allow us to describe how the properties relate to each other.

The first three features were already explained with respect to Figure 2.12. Two properties `p` and `q` are **equivalent** if and only if it holds that: `x` has the relationship `p` with `y` if and only if `x` has the relationship `q` with `y`. A property `p` is the **inverse** property of `q` if and only if it holds that: Item `x` has the relationship `p` with `y` if and only if `y` has the relationship `q` with `x`. Note that this is very similar to the property of equivalence, with the difference that the second relationship is inverted (or reversed). E.g. Barak Obama is the husband of Michelle Obama, then `is the wife of` is the reverse property of `is the husband of`, as we can say, Michelle Obama is the wife of Barak Obama. Two properties `p` and `q` are **disjoint** if `x` has either relationship `p` or `q` with `y`, but not both at the same time.

The following features concern a single property `p`. A property `p` can be **transitive**, **symmetric**, **asymmetric**, **reflexive**, or **irreflexive** which should be self-explanatory. We say `p` is **functional** if and only if it holds that: Item `x` has a relationship `p` to items `y_1` and `y_2` implies that `y_1` and `y_2` are the same. We say that `p` is **inverse functional** if and only if it holds that: if `x_1` and `x_2` both have relationship `p` with `y` then $\texttt{x\_1} = \texttt{x\_2}$. Note that theses is similar to definitions in math about injective (etc.) functional properties. Please refer to [Hogan et al., 2021] for a full discussion of these features.

Ontology Features for Classes

Analogous to the features of entities and properties, we can also define features for classes (see Figure 2.15). These are very intuitive if you are familiar with OOP. We have, for example, a feature called **subClass**, which we discussed in the context of Figure 2.12. Two classes can also be **equivalent**, **disjoint**, or a **compliment** of each other. A class can also be a **union**, **intersection**, or **ennumeration** of other classes. More features defined in OWL can be found in Figure 2.15.

Rule-based Interpretation

The features described above can also be interpreted as horn rules. Figure 2.16 depicts such an interpretation. Here, each rule has a body and a head, where the body implies ($\Longrightarrow$) the head. Having put these features in the form of a rule, we can now use a standard rule engine to reason about them.

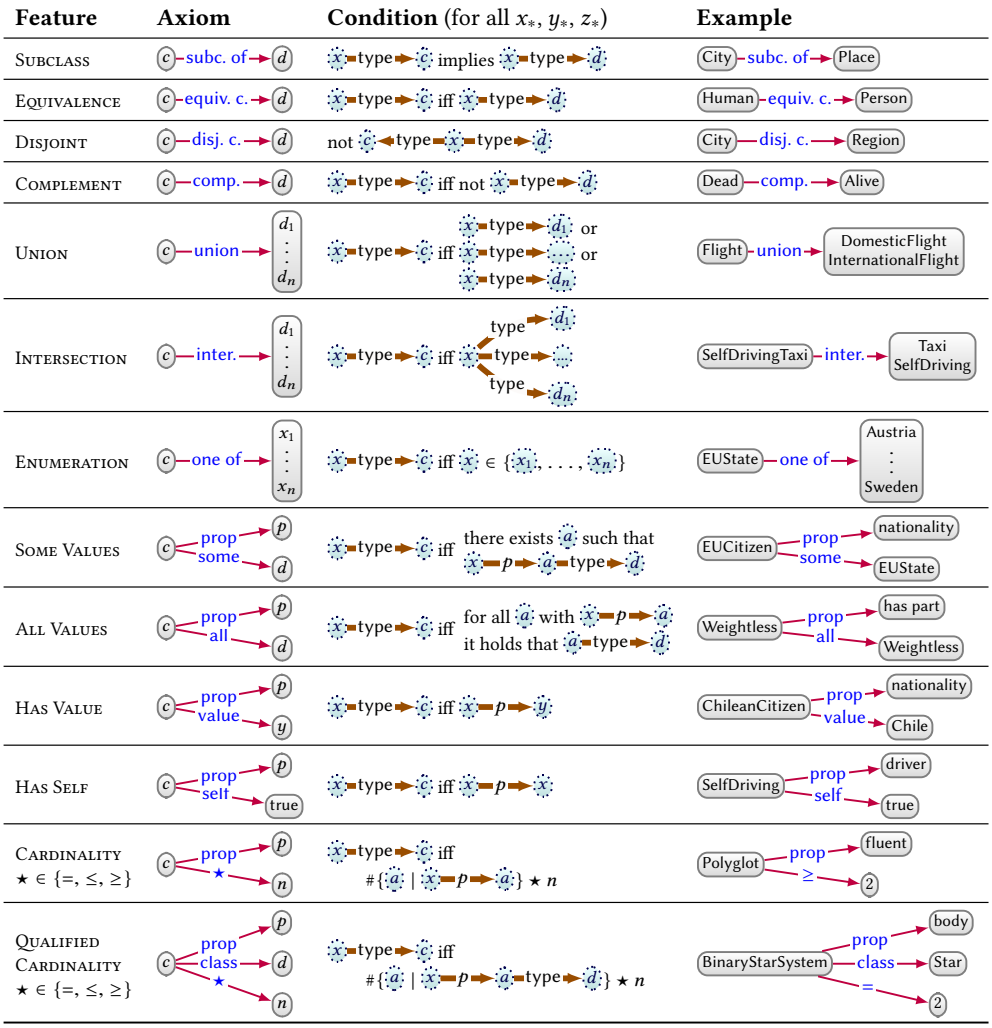


Figure 2.15: Ontology features for class axioms and definitions. (Source: [Hogan et al., 2021])

2.5.4 Description Logics: The Formal Underpinnings of OWL

Now that we have an intuitive grasp of the capabilities of OWL, we can look at the formal underpinnings for OWL. Specifically, we will look at a subset of first-order logic, called **description logic (DL)**. The advantage of using such a well-defined subpart of first-order logic is that reasoning becomes bound, i.e., we can reason within certain time and space constraints, which is a strong argument for using them for huge graphs and knowledge bases.

Feature	Body	⇒	Head
Subclass (I)	?x –type→ ?c –subc. of→ ?d	⇒	?x –type→ ?d
Subclass (II)	?c –subc. of→ ?d –subc. of→ ?e	⇒	?c –subc. of→ ?e
Subproperty (I)	?x –?p→ ?y, ?p –subp. of→ ?q	⇒	?x –?q→ ?y
Subproperty (II)	?p –subp. of→ ?q –subp. of→ ?r	⇒	?p –subp. of→ ?r
Domain	?x –?p→ ?y, ?p –domain→ ?c	⇒	?x –type→ ?c
Range	?x –?p→ ?y, ?p –range→ ?c	⇒	?y –type→ ?c

Figure 2.16: Example rules for subclass, subproperty, domain, and range features. (Source: [Hogan et al., 2021])

OWL and OWL2 are both based on various DL dialects (i.e., subsets of logic that have certain capabilities adhering to DL principles).

Definition 2.9 (Description Logic Knowledge Base)**.** A DL knowledge base K is defined as a tuple (A, T, R), where

- A is the **ABox**: A set of assertional axioms,
- T is the **TBox**: A set of class (aka concept/terminological) axioms, and
- R is the **RBox**: A set of relation (aka property/role) axioms.

Definition 2.10 (DL Interpretation)**.** A DL interpretation I is defined as a pair $(\Delta^I, \cdot^I)$, where

- Δ^I is the **interpretation domain**, and
- $\cdot^I$ is the **interpretation function**.

The interpretation domain is a set of individuals, i.e., the instances in OOP speak. The interpretation function accepts a definition of either an individual a, a class C, or a relation R, mapping them, respectively, to

an element of the domain ($a^I \in \Delta^I$), a subset of the domain ($C^I \subseteq \Delta^I$), or a set of pairs from the domain ($R^I \subseteq \Delta^I \times \Delta^I$) that will give us the assertions thereof.

Table 7. Description Logic semantics

Name	**Syntax**	**Semantics** ($\cdot^I$)
	Class Definitions	
Atomic Class	A	A^I (a subset of Δ^I)
Top Class	$\top$	Δ^I
Bottom Class	$\bot$	$\emptyset$
Class Negation	$\neg C$	$\Delta^I \setminus C^I$
Class Intersection	$C \sqcap D$	$C^I \cap D^I$
Class Union	$C \sqcup D$	$C^I \cup D^I$
Nominal	$\{a_1, ..., a_n\}$	$\{a_1^I, ..., a_n^I\}$
Existential Restriction	$\exists R.C$	$\{x \mid \exists y : (x, y) \in R^I \text{ and } y \in C^I\}$
Universal Restriction	$\forall R.C$	$\{x \mid \forall y : (x, y) \in R^I \text{ implies } y \in C^I\}$
Self Restriction	$\exists R.\text{Self}$	$\{x \mid (x, x) \in R^I\}$
Number Restriction	$\star\, n\, R$ (where $\star \in \{\geq, \leq, =\}$)	$\{x \mid \#\{y : (x, y) \in R^I\} \star n\}$
Qualified Number Restriction	$\star\, n\, R.C$ (where $\star \in \{\geq, \leq, =\}$)	$\{x \mid \#\{y : (x, y) \in R^I \text{ and } y \in C^I\} \star n\}$

Figure 2.17: Description logic semantics: Part 1. (Source: [Hogan et al., 2021])

Figure 2.17 summarizes the syntax and semantics of the main elements usually employed a description logic. For instance, we have an **atomic class** A, a **top class** $\top$, which is the class that contains all classes, the **bottom class** $\bot$, which is the class that contains no classes. Again, this can be interpreted as OOP classes, where the top class would be the upmost class in a hierarchy. Furthermore, we have **class negation** $\neg$, **class intersection** $\sqcap$, and **class union** $\sqcup$. So, for a class intersection we could for example say that the intersection of all the people at university and all the people who play soccer is the set of people at the university, who play soccer.

We also have two quantifiers, namely the **existential restriction** (or existential quantifier), and the **universal restriction** (or universal quantifier). For example, we can consider the set of all professors A, the set of all people at IfI[19] I and the relationship `isAdvisedBy`, as in x`isAdvisedBy`y. Now we need to consider that for the universal quantifier we have to note the relationship and then the class. So, we would write $\forall$`isAdvisedBy`.P describes the set of all people advised by IfI professors, where $P = A \sqcap I$ is the set of all professors at IfI. Note, and here it becomes a bit confusing at first, that describing and naming a set is the same as describing and naming a class of individuals.

[19]The department of informatics. In German Institut für Informatik (IfI).

Class Axioms (T-Box)		
Class Inclusion	$C \sqsubseteq D$	$C^I \subseteq D^I$
Relation Definitions		
Relation	R	R^I (a subset of $\Delta^I \times \Delta^I$)
Inverse Relation	R^-	$\{(y, x) \mid (x, y) \in R^I\}$
Universal Relation	U	$\Delta^I \times \Delta^I$
Relation Axioms (R-Box)		
Relation Inclusion	$R \sqsubseteq S$	$R^I \subseteq S^I$
Complex Relation Inclusion	$R_1 \circ \ldots \circ R_n \sqsubseteq S$	$R_1^I \circ \ldots \circ R_n^I \subseteq S^I$
Transitive Relations	Trans(R)	$R^I \circ R^I \subseteq R^I$
Functional Relations	Func(R)	$\{(x, y), (x, z)\} \subseteq R^I$ implies $y = z$
Reflexive Relations	Ref(R)	for all $x \in \Delta^I : (x, x) \in R^I$
Irreflexive Relations	Irref(R)	for all $x \in \Delta^I : (x, x) \notin R^I$
Symmetric Relations	Sym(R)	$R^I = (R^-)^I$
Asymmetric Relations	Asym(R)	$R^I \cap (R^-)^I = \emptyset$
Disjoint Relations	Disj(R, S)	$R^I \cap S^I = \emptyset$
Assertional Definitions		
Individual	a	a^I (an element of Δ^I)
Assertional Axioms (A-Box)		
Relation Assertion	$R(a, b)$	$(a^I, b^I) \in R^I$
Negative Relation Assertion	$\neg R(a, b)$	$(a^I, b^I) \notin R^I$
Class Assertion	$C(a)$	$a^I \in C^I$
Equality	$a = b$	$a^I = b^I$
Inequality	$a \neq b$	$a^I \neq b^I$

Figure 2.18: DL Rules for OWL2 classes, relations, and individuals: Part 2. (Source: [Hogan et al., 2021])

In addition, we have cardinality constraints, such as the **number restriction** and the **qualified number restriction**. Using the number restriction we can, for instance, say ≥ 5`isAdvisedBy`.P, which describes the class (or set) that contains all the entities (i.e., IfI members) that are being advised by IfI professors, who advise at least 5 indivduals. Hence, we put a cardinality constraint on the relationship. We can now use these operators to define new classes.

Similarly, there are DL rules that underlie the rest of the OWL ontology features for individuals, properties, and classes discussed above. These can be found in Figure 2.18.

Example 2.9 (Description Logic vs. First-order Logic)**.** We now will look at a short example to help clarify the distinction between description logic and first-order logic. Assume that we have two sets `Actor` and

`Artist`, then we can relate them as follows using description logic:

$$\texttt{Actor} \sqsubseteq \texttt{Artist}$$

In first-order logic we can express this as

$$\forall x(\texttt{Actor}(x) \rightarrow \texttt{Artist}(x)),$$

I.e., for every x where $\texttt{Actor}(x)$ is true, $\texttt{Artist}(x)$ is also true. This means that we have a category of objects that are artists and a category of things that are actors, and we say that an actor is a kind of artist.

Example 2.10 (Description Logic vs. First-order Logic 2)**.** To look at a slightly more complex example, we can describe with description logic the set of all US governors that are also actors, which are a subset of either bodybuilders or are not Austrian:

$$\texttt{Actor} \sqcap \texttt{USGovernor} \sqsubseteq \texttt{Bodybuilder} \sqcup \neg\texttt{Austrian}$$

In first-order logic we would write this as:

$$\forall x\,(\texttt{Actor}(x) \wedge \texttt{USGovernor}(x) \rightarrow \texttt{Bodybuilder}(x) \vee \neg\texttt{Austrian}(x))$$

Example 2.11 (Description Logic vs. First-order Logic with Relationships)**.** We now look at an example which includes relationships. This is the point, where it starts to become counter-intuitive for some. Let us say that we want to express the statement that people, who have received at least one award have a proud leader. In description logic we can express this as:

$$\exists\texttt{hasReceived}.\texttt{Award} \sqsubseteq \forall\texttt{hasLeader}.\texttt{Happy}$$

Essentially what this tells us is that entities, that have at least one (i.e., it $\exists$) relationship of the type `hasReceived` to an `Award`—in other words people, who have received at least received one reward—are a subset of all the people (i.e., $\forall$), who have a leader (i.e., a relationship `hasLeader`) to entities that belong to the category of `Happy` (i.e., happy people).

We can dissect this it as follows:

$$\underbrace{\exists \underbrace{\texttt{hasReceived.Award}}_{\text{people who have received awards}}}_{\text{people, who have at least one } \texttt{hasReceived.Award} \text{ relationship}} \sqsubseteq \underbrace{\forall \texttt{hasLeader}. \underbrace{\texttt{Happy}}_{\text{all happy people}}}_{\text{all subordinates of happy people}}$$

In first-order logic we can express this as:

$$\forall x \left(\exists y \left(\texttt{hasReceived}(x, y) \land \texttt{Award}(y)\right) \rightarrow \forall z \left(\texttt{hasLeader}(x, z) \rightarrow \texttt{Happy}(z)\right)\right)$$

Example 2.12 (Description Logic with Cardinality Constraint)**.** We can express the set of polygamists (i.e., people who are married to more than one person) using description logic as follows:

$$\texttt{Polygamist} \sqsubseteq \ >_1 \ \texttt{Married}.\top$$

Recall that top, $\top$, is the set of all classes, here people. Then $\texttt{Married}.\top$ is the set of all people that are married to someone of the set of all people, i.e., it is the set of all married people. By adding the cardinality constraint, we take only the people from the set of all married people, who are in a married relationship with more than 1 person.