Semantic Web 0 (0) 1
IOS Press

A Knowledge Engineering Primer

Agnieszka Ławrynowicz [a,*], Jose Emilio Labra Gayo [b] and Mayank Kejriwal [c]
[a] *Faculty of Computing and Telecommunications, Poznan University of Technology, Poland*
E-mail: alawrynowicz@cs.put.poznan.pl
[b] *Dept. Computer Science, University of Oviedo, Spain*
E-mail: labra@uniovi.es
[c] *Information Sciences Institute, University of Southern California, CA, USA*
E-mail: kejriwal@isi.edu

Abstract. The aim of this primer is to introduce the subject of knowledge engineering in a concise but synthetic way to develop the reader's intuition about the area. The main knowledge organization systems are explained with examples. We also describe methodological aspects concerning knowledge engineering.

Keywords: knowledge engineering, knowledge base, semantic network, RDF, ontology, description logic, knowledge graph

1. Introduction

Knowledge can take different forms. We distinguish between declarative knowledge (knowing something) or procedural knowledge (knowing how, know-how), sensorimotor knowledge (riding a bicycle), and affective knowledge (deep understanding). The classic definition of *knowledge* derived from philosophy defines knowledge as a justified true belief. It can be said to occur in situations where we consider something to be objectively "true" or "stated". Another definition refers to what is "explicit knowledge" that is something that is known and can be written down [79].

Knowledge representation [18] is a (symbolic) encoding of statements (or facts). A mapping can be defined between the facts and their representation, which assigns to the facts the corresponding symbols in the representation. Knowledge representation in artificial intelligence refers to how data, information and knowledge are stored and processed in computer systems.

A *knowledge base, KB* is, in some simplification, a collection of facts representing entities, classes, attributes, and relationships, relevant generally or in a particular domain, which is prepared in a digital form. The above definition of a knowledge base may resemble a database description. How, then, does a knowledge base differ from a database? A good knowledge representation system, with which we represent a given knowledge base, in addition to the ability to represent the required forms of knowledge, should ensure the power and efficiency of reasoning and knowledge extraction. An important aspect of knowledge representation is the ability to perform inference that extends the represented knowledge with new knowledge. To perform inference, a type of software called a *reasoner* is used to derive new facts from a set of pre-existing, explicitly represented facts or axioms.

It is worth noting the trade-off between the expressivity of the knowledge representation language (that is, the variety and number of possibilities for representing knowledge in it) and the performance of reasoning engines. The more complex the language, and thus the more diverse forms of modelled knowledge, the more complex the

*Corresponding author. E-mail: alawrynowicz@cs.put.poznan.pl.

inference algorithms and the longer the inference time. In [13], Blumauer and Nagy classified popular knowledge organization systems. We extend this classification in Table 1 by additional forms of knowledge representation, showing selected classes of such systems of increasing complexity.

Table 1
Knowledge organization systems.

Knowledge organization system	**Building blocks**	**Examples**
Thesaurus	Synonyms, antonyms, broader and narrower concepts, associative relationships	blueberry = bilberry cold ≠ warm Blueberry juice *is related to* blueberry
Taxonomy	Hierarchical relationships	Blueberry `is-a` fruit
Semantic network	Any unary and binary relations	Maria Skłodowska-Curie `was born in the year` 1867 Parsley root `is part of` parsley
Frame	Attributes inherited by subclasses and instances	Country `has a capital.` Poland `has a capital` Warsaw.
Ontology	Classes, attributes, relationships, constraints	Parsley `is a subclass of the class` plants Carrot `has colour` orange Blueberry juice `is made from` blueberry `Every` tree `has at least one` root
Knowledge graph	Classes, attributes, relationships, constraints, links to other knowledge bases	Entity `Maria_Sklodowska_Curie is the same as` entity `wikidata:Q7186`

Issues of *knowledge acquisition*, including issues of the knowledge base construction process, are dealt with by the field of *knowledge engineering*. A knowledge engineer explores a domain, determines which concepts are relevant to that domain, and creates a formal representation of entities, relationships and constraints for that domain. He or she is often not a domain expert, and his or her role is to obtain knowledge from domain experts, among others.

Most knowledge representation systems proposed in artificial intelligence research are systems where knowledge is represented in symbolic form, easily readable by humans. The most important of these are:

- predicate calculus [77]),
- production rules [25],
- semantic networks [82],
- frames [72],
- ontologies [36, 38],
- knowledge graphs [44].

Most modern knowledge representation languages are declarative languages based on the concept of frames or first-order logic [62, 63]. Establishing a given language on the foundations of logic allows for the formalization and standardization of reasoning procedures, which in turn allows for constructing reasoning engines that operate on a given formalism. It is also worth mentioning that while knowledge structures themselves, such as knowledge graphs, are currently represented in symbolic form, in order to operate on them, sub-symbolic representations are also often created, such as *embeddings*.

Why is the issue of representation important at all? What makes one representation better than another in the context of artificial intelligence? Many information and knowledge processing tasks can be very easy or complex, depending on how they are represented. This general principle applies both in everyday life and in artificial intelligence. To illustrate it, let us take maps as an example. Old maps, such as those from the 16th century, were static and the localities on them were marked as a point rather than a region (polygon). The digital version of such maps, while continuing to mark localities as points, allows queries to be made to a historical-geographical information system about the location and attributes of localities over time and faster retrieval of information. Modern digital maps also have layers (buildings, roads, forests, etc.) and allow querying on various aspects of the terrain such as Points of Interest, etc. Depending on the representation and its expressivity, we can ask the model about different properties (only about the geographical coordinates or also about the type of Point of Interest some object may have etc.).

We have already mentioned that a good knowledge representation system should facilitate both the acquisition of knowledge and its use, including reasoning based on its representation. In general, a good representation of both knowledge and information facilitates the subsequent task on which the representation is operated and increases the efficiency or speed of its solution, such as being easier to process by machine learning models or making it easier to answer questions. And it is often in terms of the task that we choose the suitable representation.

For example, we want to build a machine learning model to recognize images of animals. In that case, a good data representation might be images in the form of raw pixels. The model will be able to learn from this form of input data and will be able to recognize different animals based on pixel patterns. If, on the other hand, we want to explore relationships between known scientists, a good option would be to create a graph in which the nodes are individuals, and the edges are labelled with the types of relationships between them (e.g., supervisor, co-author). Representing the data in this way makes finding connections between scientists and determining the degree of their proximity more simple. For example, we can easily find people who have published scientific articles together.

Other important aspects are the interpretability and reusability of a given knowledge model, including ease of modification and addition of new information. An example of a good knowledge representation in the context of medicine could be a knowledge graph containing information on various diseases, symptoms and treatments. For example, a knowledge graph might contain information about various diseases, such as diabetes or heart disease, and information about what the typical symptoms of these diseases are and what treatments are available. In this application, the knowledge graph makes it easy to find and interpret information about specific diseases and treatments, and to easily add new information.

An appropriate knowledge representation should facilitate reasoning. For example, in a health and nutrition ontology, one can define concepts such as: `disease`, `diet`, `nutrient`, and `drug`, as well as relationships between them, such as: `diet supporting the treatment of the disease`, `drug for the disease` or `effect of the drug on the absorption of the nutrient`, and constraints, such as the state of `hyperglycemia`, concerning blood sugar levels, is disjoint with the state of `hypoglycemia`. This representation of knowledge in an information system, where individual concepts, relationships, and constraints (axioms) are explicitly represented, should facilitate reasoning about recommended diets for people struggling with a particular disease. A diet recommendation system, for example, could include as a component an ontology about diabetes and use it to infer that diabetes is a disease that requires a special diet and that certain nutrients are particularly important to be included in the diet for people with diabetes and certain others should be restricted. As a result, the system can generate consistent and reasonable dietary recommendations for such people. It is precisely for the sake of facilitating inference and ensuring that the generated conclusions or recommendations are consistent and verifiable that many knowledge representation systems are based on logic.

2. Logical foundations

Logics are formal languages used to represent information in such a way that conclusions can be drawn. *Syntax* defines the statements (sentences) that can be formulated in a given language, knowledge representation structures. *Semantics* defines the *meaning* of statements, their *interpretation*, i.e., it determines the *truth* of statements in the world. Statements in logical form represent certain aspects of the world. The world, on the other hand, is the interpretation that gives meaning (semantics) to statements in logical form. Meaning in the logical sense is the relationship between statements in logical form and interpretations, i.e. possible worlds, including imagined ones.

Logicians typically think in terms of *model* theories. Models are formally structured worlds against which truthfulness can be evaluated. We say that m is a model of the statement α if α is true in m. By $M(\alpha)$ let us denote the set of all models of α. The *entailment (logical consequence)* means that one thing follows (logically) from another:

$$KB \models \alpha \tag{1}$$

The statement α is a logical consequence of the knowledge base KB if and only if it is true in all worlds where KB is true. $KB \models \alpha$ if and only if $M(KB) \subseteq M(\alpha)$. A sentence is *valid* if it is true in every model.

The definition of logical consequence can be applied to derive inferences, i.e. to perform *logical inference*. To understand the connection between the concept of logical consequence and logical inference, we can think of sentences that are logical consequences of the KB knowledge base, of which there may be many. Inference algorithms are used to find a subset of such statements as inferences (conclusions). We say that an inference algorithm i can *derive* a statement α from the KB knowledge base. An inference algorithm is *sound* if only valid sentences are provable in it. An inference algorithm is *complete* if every valid sentence is provable in it.

As described above, the semantics of first-order logic is typically defined using model theory. Statements can be assigned a logical value by defining the interpretation of the symbols of a given language belonging to the family of first-order logic. Interpretation $\mathcal{I}$=($\Delta^{\mathcal{I}}$, $\cdot^{\mathcal{I}}$) consists of a non-empty set $\Delta^{\mathcal{I}}$ (domain of interpretation) and interpretation function $\cdot^{\mathcal{I}}$. The interpretation function $\cdot^{\mathcal{I}}$ assigns elements from the set of the symbols to $\Delta^{\mathcal{I}}$.

3. Semantic networks

Semantic networks are a graphical notation for representing knowledge represented as a set of nodes (concepts) connected by labelled arcs that represent relationships between nodes. Figure 1 shows an example of a semantic network.

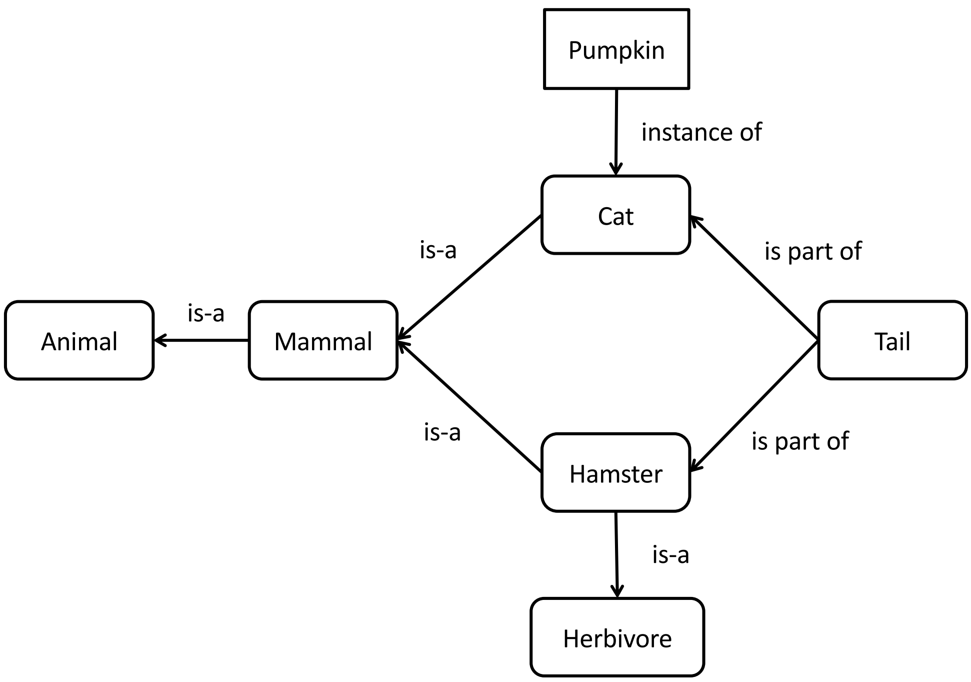

Fig. 1. An example of a semantic network.

.

Semantic networks became part of artificial intelligence research in the 1960s. However, they had already been used in philosophy, psychology and linguistics before that.

The modern manifestation of semantic networks is the so-called *Semantic Web* [10]. The technologies of the Semantic Web, including the Resource Description Framework (RDF), RDF Schema (RDFS) and the Web Ontology Language (OWL), described in the following sections of this primer, are used to create and disseminate semantic networks and ontologies in a standardized form to allow global knowledge exchange on the World Wide Web. The father of the idea of the Semantic Web is Sir Tim Berners-Lee, who has also invented the World Wide Web.

Resource Description Framework, RDF (https://www.w3.org/TR/REC-rdf-syntax/) combines the concept of semantic networks with web technologies. The resources we want to describe can be, for example, specific people, locations, or abstract concepts.

Data in the RDF model is represented using the so-called "triple model", in which sentences are represented as triples consisting of a subject s, a predicate p and an object o, for example:

`Warsaw is_part_of Poland`

where `Warsaw` is the subject of the triple, `is_part_of` is the predicate, and `Poland` is the object. Such a triple can be visualized as a graph (Figure 2).[1]

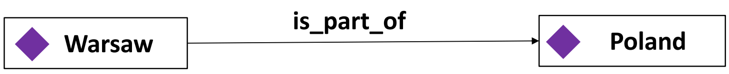

Fig. 2. An RDF triple represented as a graph.

Generally, there may be several localities with the same name, so the need arises to mark in a unique way what resource we are referring to. Web technologies come to the rescue, in particular *global identifiers (Uniform Resource Identifier, URI)*. Using URIs, we can easily create globally unique names in a decentralized manner – each domain name owner can create new URI references. URIs can also serve as a means of accessing information describing a given resource, much like, known from web technologies, an URL (each URL is also a special case of a URI). A URI may contain a part, called a fragment identifier, separated from the base part of the URI by the # symbol. For example, the role of the fragment identifier in the URI `http://example.edu#Warsaw` plays the string `Warsaw`. Since URI identifiers are usually long character strings, a simplified, abbreviated version called qnames was introduced. A URI expressed as a qname consists of two parts: the namespace and the identifier, separated by a colon. For instance, in `edu:Warsaw`, a qname identifier, which refers to the namespace, is `edu`, while `Warsaw` refers to the fragment identifier.

The basic elements of RDF are:

- resources, which are identified by URIs and correspond to nodes in the graph, e.g. `http://example.edu`,
- blank nodes, i.e. graph elements that are not given a label or URI identifier, and are often used to describe objects that do not have their own URI identifier or to build complex expressions that consist of multiple elements, when at the same time one does not want to create separate resources for each element. The strings representing blank nodes start with the characters `_:`, and software frameworks create them automatically,
- properties, identified by URIs, corresponding to arcs in the graph, and representing the binary relationships, e.g. `http://example.edu#is_part_of`,
- literals that represent specific data values e.g. `Warsaw`, `2022-05-26`.

Now let us formalize our knowledge of RDF graphs. Let us consider the pairwise disjoint sets $\mathbf{U}$, $\mathbf{B}$ and $\mathbf{L}$. They denote resources (URI references), blank nodes, and literals, respectively. *RDF triple* is a tuple $t = (s, p, o) \in (\mathbf{U} \cup \mathbf{B}) \times \mathbf{U} \times (\mathbf{U} \cup \mathbf{B} \cup \mathbf{L})$, where s is the subject, p is the predicate, and o is the object of the triple. *RDF graph* (or RDF dataset) $\mathcal{G}$ is a set of RDF triples.

Since we only deal with at most binary relations in an RDF graph, how can we represent relations that are inherently n-ary in such a graph? We can use the *reification* design pattern, illustrated in the following example.

[1]The graphical notation is inspired by the one used in a free, open-source ontology editor and framework Protégé (https://protege.stanford.edu)

Example 1 (Reification). Now let us look at the task of transforming a given table from a relational database into an RDF graph that reflects the meaning and relationships of the data in the table. For the purposes of our task, we will use Table 2, which contains shopping data.

Table 2

Data on purchases.

Buyer	Seller	Product	Number of pieces
Marcin Kowalski	Shop1	Natural yoghurt	5
Aleksandra Nowak	Shop2	Butter	2

In the case under consideration, the ’purchase’ relationship has more than one participant, being in this relationship. In addition, none of the table’s columns stands out as leading for the relationship (purchase). When we encounter such a situation, it is worth using the following design pattern, the so-called *reification*. We create an instance representing a relation with links to all instances that are in this relation. We will represent the n-ary relation in the RDF model by creating just such a new instance representing the relation between n specific instances.

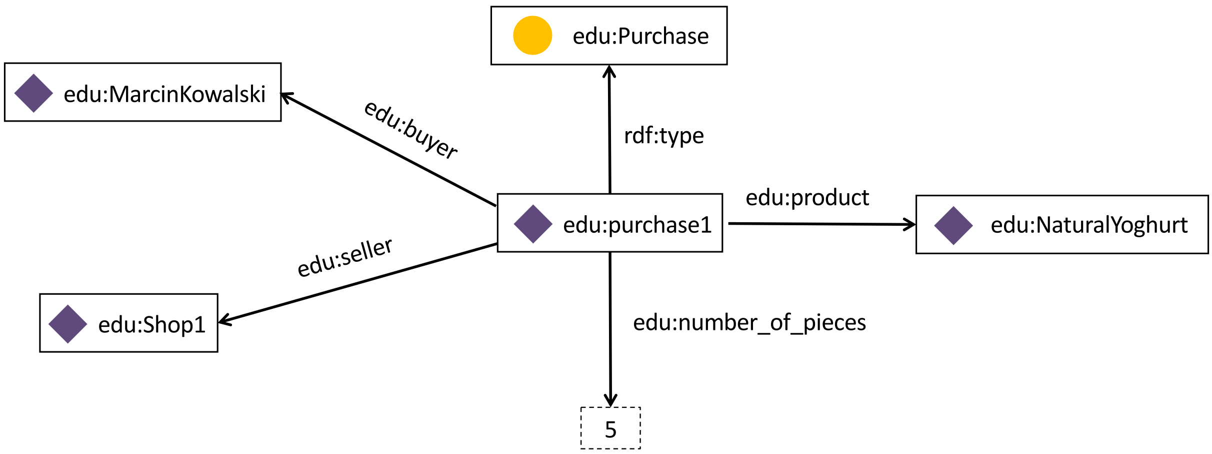

Fig. 3. RDF graph depicting n-ary purchase relationship.

This requires creating a total of n+1 triples: one to create the main instance of the relationship and one for each object that is in that relationship. Following the given pattern, and assuming that our namespace will be `http://example.edu`, the first row of the table can be visualized as an RDF graph as in the Figure 3. The set of triples corresponding to this graph is as follows:

```
edu:purchase1 rdf:type edu:Purchase
edu:purchase1 edu:product edu:NaturalYoghurt
edu:purchase1 edu:number_of_pieces "5"
edu:purchase1 edu:buyer edu:MarcinKowalski
edu:purchase1 edu:seller edu:Shop1
```

■

4. Frames

Frame is a complex data structure used in artificial intelligence to represent stereotypical situations or events. Frames, as a form of knowledge representation, are derived from semantic networks and are a specific version of them. A frame is a representation of an object or category, and allows to collect information about them. It has attributes and is in relationships with other objects or categories. This way of representing and organizing knowledge reflects the structure of the real world. Two types of frames can be distinguished:

- individual (represent a single object, such as a specific city),
- general (represent a category of objects, e.g., cities).

A single frame is a named list of *slots* that are filled with *facets*. Slots are individual pieces of information that make up a given frame, such as:

```
(frame-name
    <slot-name1 facet1>
    <slot-name2 facet2> ...)
```

A frame reflects previously accumulated experience of specific situations through defined and default values. General frames have a slot `IS-A`, which is filled in with the name of another general frame, such as:

```
(Carrots
  <:IS-A Vegetables>
  <:colour orange>  ...)
```

More specific frames inherit facets from more general frames. Individual frames have a slot `INSTANCE-OF`, which is filled with the name of the general frame, e.g.:

```
(warsaw
  <:INSTANCE-OF City>
  <:voivodeship mazowieckie>
  <:population 1 860 281>  ...)
```

Inference using a frame is done by:

- consistency checking when filling a slot with a value,
- inheritance of defined and default values (according to `IS-A`, `INSTANCE-OF`).

One of the ways in which frame-based knowledge representation manifests itself in modern systems is through the use of frame semantics, which is an approach to natural language processing that uses frame-based representations to understand or generate natural language. Frame semantics can be used in applications such as information extraction, machine translation and dialogue systems. An example of the practical use of this idea is FrameNet [8], a lexical database of English containing syntactically and semantically labelled examples of sentences from a corpus of texts. FrameNet consists of so-called semantic frames. A semantic frame is a description of the type of event, relation or entity and the units that constitute it. It consists of frame elements (frame roles) and lexical units, i.e., words that found in the text invoke the frame. In addition, sentences from the corpus annotated with frame elements are attached to the frame.

Example 2. Consider an example of a frame called `Cooking`, which represents the general concept of cooking and contains a number of slots that represent different aspects of cooking, such as a cook, ingredients, cooking method and cooking equipment. Some examples of lexical units (words or phrases that can invoke the frame) include `cooking`, `baking`, `grilling`, etc. Some examples of frame elements, specific slots in the frame that represent different aspects of a concept, are `Cook`, `Produced_food`, `Ingredients`, `Container`.

A sample sentence that fits into this example, where we can find the lexical unit associated with the frame `Cooking` might look like this:
`Today Maria is going to (bake)` [the lexical unit that invokes the frame] a `raspberry cake.`
In this sentence, we have an example of a lexical unit `bake` that triggers the frame `Cooking` as well as frame elements such as `Produced_food` (`raspberry cake`) and `Cook` (`Maria`). Some frame elements (e.g., `Container`) are not specified.
■

In a dialogue system, frame semantics can be used to help the system understand the user's intent. For example, if a user asks a question about the weather, the system can use frame semantics to identify the appropriate frame associated with the intent (e.g., `weather_forecast`) and fill in the appropriate fields associated with that intent (e.g., location, date) to generate a response.

Frame semantics is also used in Wikipedia, where it is used to represent relationships between different concepts and to provide context for articles. For example, a Wikipedia article about a particular city might include a frame (called an Infobox) that contains information about the city's location, population and other typical data. This allows readers to more easily access additional related information.

Knowledge representation in the form of frames and frame semantics are also used in robotics to help robots understand their environment and interact with it. In this context, frames represent various concepts and objects a robot may encounter, such as furniture or tools. In addition, the slots in each frame can contain information about the object's physical characteristics, such as size, shape and colour, as well as its function and use. Frame-based inference systems may allow robots to use knowledge about the environment to make decisions and take actions. For example, a robot can use its knowledge of objects in a room to navigate through it or to recognize and identify specific objects. Frame semantics can also be used in natural language processing, allowing robots to understand and respond to human commands and questions. For example, a robot can use frame semantics to understand the intent behind the command "pick up the red cup" by identifying the appropriate frame (e.g., `object_manipulation`) and filling in the appropriate fields (e.g., object to be manipulated, object's features).

Knowledge representation using frames played a significant role in developing early ontologies and tools for knowledge representation and manipulation. One example is the Protégé system, a popular software platform for building, editing and manipulating ontologies. In Protégé, a knowledge representation framework is used to represent concepts and their relationships in the form of frames, which consist of a collection of slots and fillers. Each frame represents a specific concept, and the slots represent properties or characteristics of that concept. Fillers are specific values that are assigned to each slot.

Using frame knowledge representation and frame semantics in ontologies and tools such as Protégé helped provide a structured and organized way to represent and manipulate knowledge, enabling users to manage and use large amounts of information more effectively.

5. Ontologies

The word *ontology* originated in philosophy. Ontology, as a philosophical discipline, was first studied by Aristotle. In his important philosophical work "Metaphysics" Aristotle defined what later became known as ontology, or the science of "being as being", which studies the nature of entities and their attributes. Ontology deals with describing and categorizing entities based on their structure and properties.

Ontologies in computer science are formal representations of concepts, relationships and constraints within a domain used to facilitate communication and understanding between multiple parties. In computer science, ontology is an engineering artefact, i.e., not the study of entities and their categorization but the concrete result of such categorization. Therefore, although from a philosophical perspective ontology can be seen as a particular system of categories responsible for a certain vision of the world, independent of its representation, in computer science ontology is dependent on the languages used to represent it. Several definitions of ontology have been proposed in this context. Ontology is defined by Gruber as a "formal, explicit specification of a shared conceptualization of a domain of interest" [36]. Formality is about making the ontology explicit for interpretation by machines. Clarity refers to making sure that all concepts and their interrelationships are clearly defined. Sharing refers to the ontology's capture of some consensus in modelling concepts, accepted by a community or several stakeholders, rather than an individual view. The domain of interest is established between the requirements of a specific application and the "unique truth". Another definition of ontology in computer science by Uschold states that it is "the representation, formalization and specification of important concepts and relationships within a given domain" [95]. This definition emphasizes the role of ontology in representing and specifying key concepts and relationships within a given domain, as well as formalizing these concepts and relationships.

Among the available ontologies, we can distinguish between foundational ontologies, i.e. ontologies that define the basic concepts and distinctions, the types of entities existing in the world and the relationships between them, such as objects fixed in time versus events or real or abstract objects. For example, the DOLCE [30] or BFO [5] ontology fall into this category.

On the other hand, domain-specific ontologies are designed to represent domain-specific concepts and relationships. Examples of such ontologies include biomedical ontologies, such as the SNOMED ontology, which is used to represent medical concepts and their relationships, the GO ontology, which is used to represent gene functions, or CHEBI, an ontology of biologically relevant chemical compounds.

To model ontologies, a number of representation languages have been proposed that provide various kinds of features. The most widespread have been languages based on the first-order logic since it is equipped with formal semantics and thus facilitates machines' interpretability of encoded knowledge.

5.1. RDFS language

Simple ontologies can be represented using *RDFS language (RDF Schema)* (https://www.w3.org/TR/rdf-schema/). RDFS belongs to the Semantic Web technology stack. It integrates with RDF by enriching data with its semantics formulated in the form of a data schema. Figure 4 illustrates an example of the semantic network, where RDF is the data layer and RDFS is the data schema layer.

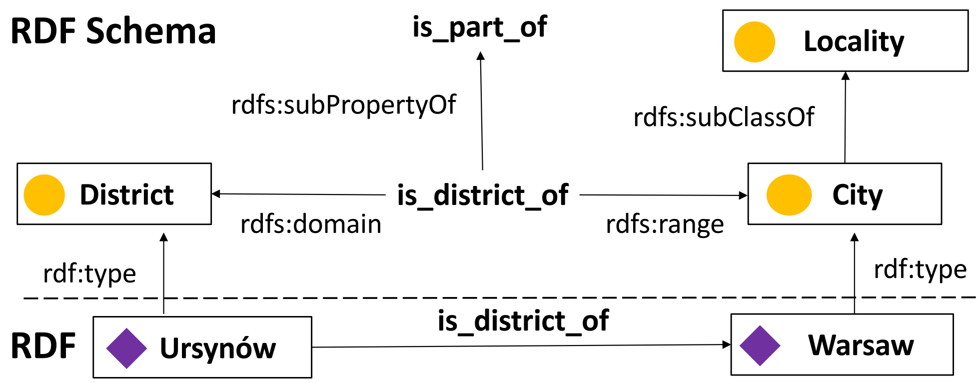

Fig. 4. RDFS-represented data schema layer embedded on RDF-represented data layer, together forming a single semantic network.

.

RDFS introduces a distinctive vocabulary to indicate to inference (reasoning) engines what conclusions they should derive from the facts being modelled. To model class hierarchies, RDFS includes the keyword `rdfs:subClassOf`, with which we can model that class C_1 is a subclass of class C_2. So, for example, we can model that class `City` is a subclass of class `Locality`. Continuing with the example, if then the knowledge base contains the following triple:

`Warsaw rdf:type City`

then we can infer (deduce) that

`Warsaw rdf:type Locality`

which allows us to model and infer the class hierarchy.

Similarly, we can use the property `rdfs:subPropertyOf` to model the fact that property P_1 is a subproperty of property P_2. For example, we can model that `is_district_of` is a subproperty of `is_part_of` and then given facts:

`Ursynów is_district_of Warsaw`

then we can infer that

`Ursynów is_part_of Warsaw.`

In addition, RDFS introduces a vocabulary for defining domain constraints (`rdfs:domain`) and range constraints (`rdfs:range`). When we denote the domain of property P_1 as class C_1 and there is a fact `entity1` P_1 `entity2` in the knowledge base, the inference will be that the instance `entity1` belongs to class C_1. Similarly, we can introduce a range constraint (`rdfs:range`) to limit the membership of a given class of objects (the third element) of the triple. For example, if there are the following facts in the knowledge base:

```
is_district_of rdfs:range City
Ursynów is_district_of Warsaw
```

then we can infer that

```
Warsaw rdf:type City
```

5.2. *OWL ontology modeling language*

The most popular, standard ontology modeling language is the *OWL (Web Ontology Language)* (https://www.w3.org/TR/owl-features/). Standardizing the knowledge representation language helps in creating tools for editing knowledge represented in the language and also in developing reasoning engines. OWL allows describing concepts (classes) in a formal, unambiguous way, based on set theory and logic. OWL ontologies are implementations of *description logic* [7], which is a subset of the first-order logic.

Knowledge bases represented using description logic typically consist of a terminological part, i.e. the schema of the knowledge base, and an assertional part, i.e. the data in the knowledge base.*The terminological part (terminological box, TBox)* contains the vocabulary used to describe the hierarchy of classes and relationships in the knowledge base. *The assertional part (assertional box, ABox)* contains statements about properties of instances.

OWL extends RDF and RDFS by providing additional vocabulary. OWL can be written using RDF syntax, where expressions containing OWL vocabulary are embedded in RDF documents and interpreted according to OWL semantics. Other common ways of writing OWL are turtle and the Manchester syntax. References to syntax formalized with description logic can also often be encountered.

The main elements that make up an ontology represented in OWL are:

- *entities* – classes, properties, individuals (instances) and any other elements of the modelled domain. A class is interpreted as a set, a property as a binary relation, and an individual as an element of a set;
- *expressions* – complex classes occurring in the modelled domain;
- *axioms* – assertions that are true in the modelled domain.

The ontology represented in OWL is a set of axioms.

Classes represent collections of individuals (instances). For example, by writing `City rdf:type owl:Class`, we can express that `City` is the class of cities, which includes instances such as `Warsaw`, `Poznan`, etc. The two special classes are: the universal class `owl:Thing`, which represents the set of all instances, is a superclass of all classes, and its interpretation is the entire domain under consideration, and the bottom class `owl:Nothing`, which represents the empty set of instances, i.e. constraints that are impossible to satisfy all together (its interpretation is the empty set).

Complex classes can be built from simpler classes using logical operators. This gives us a "conceptual Lego", where we construct complex classes from other (potentially complex) classes.

Let us denote (complex) classes by C, D, properties by R, S, and individuals (instances) by a, b. Examples of operators and their representation in description logic and turtle notations are shown in Table 3.

Because of the *Open World Assumption, OWA*, discussed later in this primer, it is worth noting the interpretation of some operators, particularly negation. Negation is interpreted in OWL as a complement, as illustrated in the Figure 5 on the left, referring to the expression `owl:complementOf Meat`.

In practice, inferring all imaginable non-meat objects, including those unknown and not described in the knowledge base is challenging. Therefore, to infer negation, it is often captured in the context of a superclass (e.g., `FoodProduct`), which describes a set of instances, some of which are, for example, meat and the rest are other

Table 3

Examples of logical operators that can be used in OWL to construct complex classes shown using description logic and turtle syntaxes.

Operator	Syntax (description logic)	Example (description logic)	Example (turtle)
universal class	$\top$	$\top$	`owl:Thing`
bottom class	$\bot$	$\bot$	`owl:Nothing`
negation	$(\neg C)$	(¬ `Meat`)	`owl:complementOf Meat`
intersection	$(C \sqcap D)$	(`Child` ⊓ `Man`)	`owl:intersectionOf (Child Man)`
union	$(C \sqcup D)$	(`Herbivore` ⊔ `Carnivore` ⊔ `Omnivore`)	`owl:unionOf (Herbivore Carnivore Omnivore)`
existential quantification	$(\exists R.C)$	(∃`eats.Meat`)	`owl:onProperty :eats ;` `owl:someValuesFrom :Meat`
value restriction	$(\forall R.C)$	(∀`eats.VegetarianProduct`)	`owl:onProperty :eats ;` `owl:allValuesFrom :VegetarianProduct`

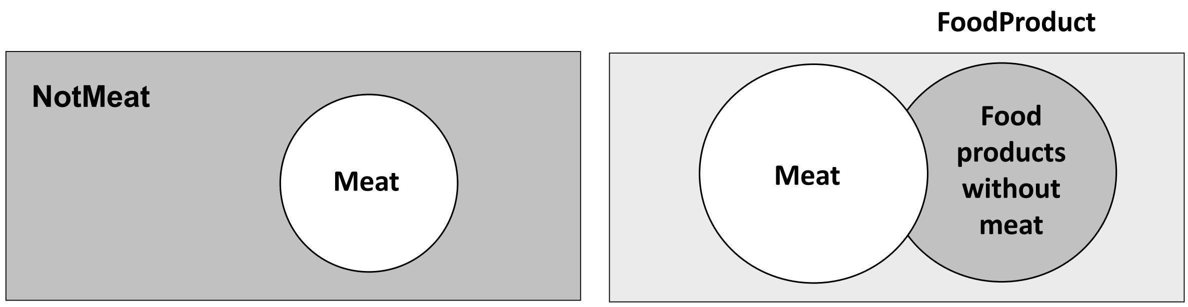

Fig. 5. Interpretation of negation as complement (left) and complement in the context of superclass (right).

instances belonging to the class `FoodProduct` as illustrated in the Figure 5 (right), where the area marked in light gray is the complement of a white and dark gray circle relative to the rectangle surrounding the whole. The class of things that vegetarians eat is in the area marked in dark gray.

Properties exist independently of classes. Properties in OWL are divided into:

– object properties that link a resource to another resource, e.g. the property `is_part_of` can link instances `Ursynów` and `Warsaw`,
– data properties that connect a resource to a literal, e.g. the `weight` property can associate a given product with its weight,
– annotation properties that link the resource to a note about it, e.g. `rdfs:label` links the resource to its label.

In addition, properties can have their own characteristics, of which we can distinguish:

– inverse property, e.g.:

```
:is_parent_of rdf:type owl:ObjectProperty ;
               owl:inverseOf :has_parent .
```

– functional property, e.g:

```
:has_father rdf:type owl:ObjectProperty ,
                 owl:FunctionalProperty .
```

– transitive property, e.g:

```
:is_part_of rdf:type owl:ObjectProperty ,
                      owl:TransitiveProperty ;
             rdfs:domain :Region ;
             rdfs:range :Region .
```

For classes, we can define three main types of axioms that define relationships between classes (also shown in Table 4):

- *subsumption*,
- *equivalence*,
- *disjointness*.

Table 4

The main axioms in the OWL language that define the relationships between classes presented by means of two syntaxes: the description logic syntax and the turtle syntax.

Axiom	Syntax (description logic)	Example (description logic)	Example (turtle)
subsumption	$C \sqsubseteq D$	Boy $\sqsubseteq$ Child	`Boy owl:subClassOf Child`
equivalence	$C \equiv D$	Country $\equiv$ State	`Country owl:equivalentClass State`
disjointness	$\mathsf{C} \sqcap \mathsf{D} \sqsubseteq \bot$	Herbivore $\sqcap$ Carnivore $\sqsubseteq \bot$	`Herbivore owl:disjointWith Carnivore`

Analogous axioms can be defined for properties, but they are much less frequently specified due to the *entity-centric* characteristics of the ontologies represented in OWL.

We introduce the subsumption relation using the keyword `owl:subClassOf`. By modelling the subsumption relation, we introduce the necessary conditions that instances of a class must satisfy. In this way, we model semantic class descriptions. One can interpret such a relation as a one-way implication. For example, we may wish to model that “all carnivores eat meat”:

```
:Carnivore rdf:type owl:Class ;
       rdfs:subClassOf [ rdf:type owl:Restriction ;
                         owl:onProperty :eats ;
                         owl:someValuesFrom :Meat
   ] .
```

The keyword `owl:equivalentClass` represents an equivalence relation. We model class definitions in this way. Such a relation can be interpreted as a two-way implication, which defines the necessary and sufficient conditions to consider an instance of a class as an instance of another class and vice versa. This means that equivalent classes share the same set of instances. For example, we may want to model that “every boy is a child and a man” and at the same time “everyone who is a child and a man is a boy”:

```
:Boy rdf:type owl:Class ;
     owl:equivalentClass [ rdf:type owl:Class ;
                  owl:intersectionOf ( :Child  :Man )
 ] .
```

As long as we do not explicitly introduce the *disjointness constraint*, classes can share instances. If we want that a given instance cannot belong to two given classes at the same time, we can introduce the axiom of class disjointness using the keyword `owl:disjointWith`. For example, given the following statements:

```
Herbivore owl:disjointWith Carnivore
     Pumpkin rdf:type Carnivore
```

we can infer that Pumpkin is not a herbivore. The introduction of disjointness constraints into ontologies is very important from the point of view of checking the consistency of an ontology or knowledge base.

At the level of individuals (i.e. instances of classes), there are two main types of axioms:

- assertions of individuals to classes, e.g.: `Warsaw rdf:type City`, and
- assertions of individuals to properties, e.g.: `Ursynów is_district_of Warsaw`.

The `owl:sameAs` property has an important role in reconciling entities that have different identifiers but are semantically equivalent to each other. In contrast, when wishing to express that given instance identifiers refer to different objects, we can use the property `owl:differentFrom` (modelling the relationship between two instances) or `owl:allDifferent` (modelling the relationship between instances from a set as pairwise disjoint).

5.2.1. Reasoning

The use of logic to model ontologies allows the use of inference engines. We can use inference engines, e.g. to check that all statements and definitions in the ontology are mutually consistent, to check which classes are in a superclass-subclass relationship (subsumption relationship) with each other automatically, and others. Inference can thus keep the hierarchy of classes in the ontology in the right order.

The basic inference tasks in OWL can be divided into schema („terminological" part of the ontology) inference tasks and instance („assertional" part of the ontology) inference tasks.

For the terminology (i.e. TBox) these are:

– checking whether a given (complex) class C is a subclass of another class D in a logical sense (subsumption test), i.e. whether C `owl:SubClassOf` D is a logical consequence of KB (the set of instances of class C is a subset of instances of class D in all KB models),
– checking whether a given (complex) class C is logically equivalent to another class D i.e. whether C `owl:EquivalentClass` D is a logical consequence of KB (the set of instances of class C is equal to the set of instances of class D in all KB models),
– checking whether a class is *satisfiable*, whether the set of constraints describing it is not contradictory, i.e. whether C `owl:EquivalentClass owl:Nothing` is not a logical consequence of KB (the set of instances of class C is not an empty set for some KB model).

The inference tasks typical of the assertional part, ABox, are:

– checking whether the ABox is *consistent*, i.e. whether it has a model, the so-called task of *consistency checking*,
– checking whether a given individual a is an instance of a given class C in the logical sense, the so-called task of *instance checking*,
– given an ABox and a class C, finding all individuals a such that the assertion a `rdf:type` of C is a logical consequence of the ABox, so-called *retrieval problem*,
– having an individual a and a set of classes, finding *most specific class* C from the set such that the assertion a `rdf:type` C is a logical consequence of ABox, so called *realization problem*.

Satisfiability and consistency tests can be used to determine the meaningfulness of the KB knowledge base. Subsumption tests are often used to automatically construct class hierarchies. Instance tests are queries designed to return individuals that satisfy the query conditions.

5.2.2. "Closed world assumption" versus "open world assumption"

Inference engines in OWL operate on certain assumptions, which sometimes differ from those made to query relational databases, for example. When dealing with a standard, centralised database, the so-called "closed-world assumption" is made, i.e. that we have complete knowledge of the instances and that missing information is negative information (negation-as-failure). However, by querying the knowledge base represented in the OWL language, we assume incomplete knowledge of instances. Any negation of a fact must be explicitly stated.

Example 3 (Assumption of "open world"). Consider an example illustrating how making a given assumption about the "closedness" or "openness" of the world affects inference. The Table 5 represents information from the knowledge base under development regarding food products and the allergens that may be present in their composition.

Table 5
Food products and the allergens typically found in them.

Product	Allergens
wheat flour	gluten
dark soy sauce	soya
sausages	soya, gluten
cream	milk
peanuts	nuts

Let us assume that we want to query the knowledge base on gluten-free products. Assuming a "closed world", the answers would be: `soy sauce`, `cream` and `peanuts`. However, in reality, both cream and soy sauce may contain gluten in their composition, only this information may not have been included in the knowledge base yet. In order to be sure of the results of inference under the assumption of an "open world", it would be necessary to include axioms explicitly stating that, for example, peanuts do not contain gluten into the knowledge base. ■

5.2.3. *Lack of unique names assumption*

Unlike most knowledge representation languages, OWL inference does not use the *Unique Names Assumption, UNA*, which is the assumption that distinct names denote distinct objects. In order to model knowledge in decentralised environments such as the Internet, it has been assumed that anyone can call anything by any name. Therefore, names cannot be assumed to be unique. However, different names do not necessarily mean different objects either. For example, two different names, `Madame Curie` and `Maria Sklodowska-Curie`, can refer to the same person. In addition, a person may be represented in some knowledge bases by alphanumeric identifiers, e.g. `wikidata:Q7186`.

Example 4 (Lack of assumption on uniqueness of names). The lack of assumption about uniqueness of names also affects the results of inference using functional properties. Suppose we have the following axioms in the knowledge base:

```
Ola has_father Jan
Ola has_father Marcin
has_father rdf:type owl:FunctionalProperty .
```

What conclusions will be derived from such a knowledge base? Will the inference engine show a contradiction? Well, due to the fact that `has_father` is the functional property and the lack of assumption of uniqueness of names, when the inference engine is run, a fact will be generated regarding the identity of the instances of `Jan` and `Marcin` i.e.: `Jan owl:sameAs Marcin`. ■

6. Knowledge graphs

A *knowledge graph* is a large, graphically structured knowledge base that represents facts in the form of relationships between entities. The basic building blocks of a knowledge graph are: entities, expressed through nodes in the graph, their properties (attributes) and the relationships connecting the nodes, expressed through edges in the graph. Entities may have (semantic) types, which is represented by the relation `is-a` between an entity and its type. It is also possible that some types, properties and relationships stored in the knowledge graph are structured in an ontology or data schema.

Example 5 (Knowledge graph). Figure 6 shows an example of a knowledge graph. The graph uses vocabulary from the RDF, RDFS and OWL namespaces, which are denoted in the figure by the prefixes `rdf:`, `rdfs:` and `owl:` respectively. The instance `Maria_Sklodowska_Curie` is linked to its semantically equivalent instance `wikidata:Q7186` (which has an alpha-numeric identifier) in the Wikidata knowledge base via the property `owl:sameAs`. The instance `wikidata:Q7186` belongs to the class `Person` and its type (class) is specified by using the property `rdf:type`. The class `Scientist` is related to the class `Researcher` through the property `rdfs:subClassOf`, meaning that it is a subclass of it, as well as through the property `owl:equivalentClass` to the class `wikidata:Q901`, meaning that they are semantically equivalent classes. In the figure we have examples of both object properties and datatype properties. An example of the first of these is the `discipline` property, which links two objects, among others: `Maria_Sklodowska_Curie` and `Chemistry`. An example of the second of these is the `birthdate`, which associates the `Maria_Sklodowska_Curie` object with a `1867-11-07` value that belongs to a specific data type, denoted via the `xsd:date` namespace as date. ■

More formal definitions specify a set of entities $\mathcal{E}$, a set of relations $\mathcal{R}$, and a knowledge graph as a directed multi-relational graph $\mathcal{G}$, representing facts or assertions as triples (s, p, o), consisting of a subject s, a predicate p, and an object o, where

$$\mathcal{G} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}, \tag{2}$$

$$(s, p, o) \subseteq \mathcal{G} \quad (3)$$

We can see a similarity with the already presented examples of semantic networks, the definition of RDF triples, or ontologies. How, then, does a knowledge graph differ from a graph composed of RDF triples or from an ontology? Not every knowledge graph uses RDF vocabulary and Semantic Web technologies, although many knowledge graphs are represented using these technologies. In a knowledge graph, an ontology is a kind of data schema that imposes semantics, meaning on the data, and usually has a shallow level of axiomatisation or is a small part of the knowledge graph. Knowledge graphs, on the other hand, are focused on data (instances) and the number of instances in a typical knowledge graph can be huge. It can be said that a knowledge graph is a kind of semantic network with added constraints. Due to this much larger scale, also the methods of knowledge acquisition or inference using the knowledge graph have to be adapted to this larger scale. One can observe a shift of attention from manual knowledge engineering methods, focusing on rule modelling and ontologies, to automatic or semi-automatic methods, often using data mining or machine learning or *crowdsourcing*[2]. Also, knowledge graph inference makes more use of graph data structure and a triple data model than complex logical inference and logical interpretation of data in the form of ontology axioms, and is often performed using statistical or neuro-symbolic methods [11] and learning sub-symbolic representations of knowledge graphs (knowledge graph embeddings [99]).

While KGs can be modeled using the Semantic Web stack (much of which is based on RDF), alternative approaches also exist and are becoming more popular in some communities [56]. One example is the *Wikidata* data model, which is used for modeling the popular Wikidata KG. Unlike RDF, this data model tends not to rely on formal URIs and is hence easier to design and store. Within the NLP community, such data models are also popular, with KGs often just represented as sets of triples, where each triple contains three strings, rather than URLs. Although less rigorous, such models offer ease of use and flexibility, which make them especially appropriate for domain modelers who are not well acquainted with the formalism of RDF. Finally, the *property graph* also offers

[2]Crowdsourcing refers to the practice of obtaining services or content by soliciting input from a large group of people, usually via the Internet. Crowdsourcing projects often involve breaking larger projects into micro-tasks, separate units of work that can be done independently and quickly, and outsourcing them to a range of collaborators rather than to a single person or organisation. The term is a combination of "crowd" and "outsourcing" and was coined in 2006 by Jeff Howe

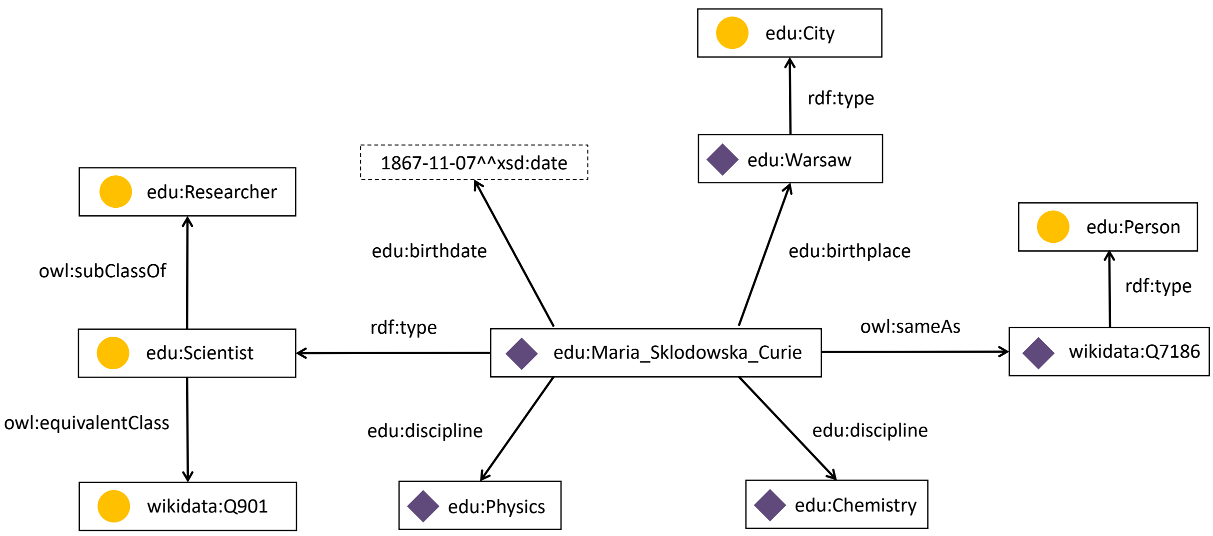

Fig. 6. Example of a knowledge graph that contains classes, instances and data of a specific type.

.

a good data model for representing KGs. Property graphs are inspired by relational database management systems (RDBMS), which tends to represent its data using sets of tables. The property graph model represents properties (the edge labels in KGs) as columns in a table. Entities then become rows in the table. An advantage of this model is that, if the data is fairly regular, an RDBMS infrastructure (which tend to be very fast) can be used for modeling, querying and storing the graph. However, KGs are generally irregular, and only a few properties are defined for a given entity. Representing a graph as a table then becomes highly suboptimal, and also preempts the use of graph machine learning algorithms.

Depending on the availability of the knowledge graph, how it is built (within an organisation or through a community) the result can be an open or corporate knowledge graph. Open knowledge graphs are publicly available. Well known examples of such graphs are DBpedia [6], Wikidata [98], YAGO [93]. They cover many domains and offer multilingual lexicalisation. Open knowledge graphs can also address specific domains such as media, geography and others.

Corporate knowledge graphs are internal within a particular company and are aimed at commercial applications. Corporate knowledge graphs are used in industries such as web search, e-commerce, social networks, pharmaceuticals, finance and others. Typical applications of knowledge graphs include semantic search, question answering, intelligent assistants, innovation support in research and design (such as the design of new drugs).

6.1. Knowledge graph construction

The typical knowledge graph construction process starts with the acquisition of a corpus and ends with a graph ready for application. Typically, this process can be illustrated by two main phases: *knowledge extraction* and the construction of the knowledge graph (including its completion or refinement) as illustrated in Figure 7.

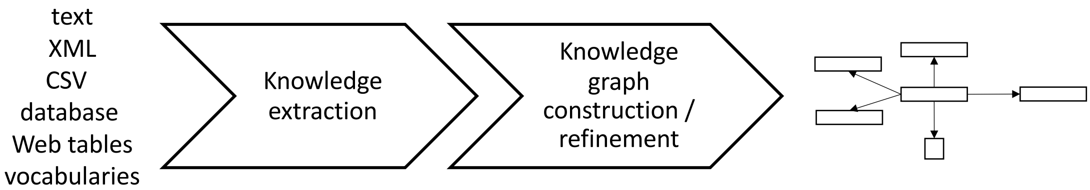

Fig. 7. Knowledge graph construction process.

The first stage involves extracting knowledge from both unstructured (e.g. text) and structured (e.g. relational databases) and semi-structured sources or by converting existing data (e.g. csv files). Natural language processing and information extraction methods can be used for this purpose. The task of knowledge extraction includes using existing knowledge resources (knowledge bases, ontologies) or generating a schema from source data. In the natural language processing (NLP) community, this task of knowledge extraction often goes by the name of *information extraction* (IE). While broad, IE comprises (at minimum) named entity recognition (NER) and relation extraction [20, 35, 52]. Considering the former, we note that it is not necessary for entities in text to always be named. Consider, for example, the sentence 'We do not know who stole the package, but witnesses claim that he drove off in a white van.' The person who stole the package is a non-abstract entity who obviously does exist in the real world, but is unnamed. However, we still have some information about the entity, such as its type ('Person') and what vehicle the entity drives. This example illustrates that extracting unnamed entities may also be of interest, but in most applications, they are rare compared to the occurrence of named entities in the text. Hence, a large focus of IE is on extracting named entities, using techniques that (over the last several decades) range from classic rule-based techniques, mostly popular in the 1990s and earlier, to more recent deep learning techniques based on transformer-based models and even generative AI. IE has also been explored in multiple domains, including 'usual' domains like news corpora to domains such as human trafficking [53, 57, 84].

While NER focuses on extracting named entities (which are the nodes in the KG), relation extraction focuses on extracting the edges. In the KG definition, edges are relationships between entities. Relationships of interest are pre-defined in the ontology. For example, in the sentence 'Michael was the CEO of Dell Corporation', 'Michael' and 'Dell Corporation' would be extracted as entities, while 'CEO_of' would be extracted as the relation between those two entities. This is assuming that 'CEO_of' is defined in the ontology. As the relation is quite specific, it may instead be the case that a more generic relation (e.g., 'employee_of' or 'executive_of') is defined instead. Hence, pattern matching and string matching algorithms are not as likely to work as the previous example suggests. Another element that makes relation extraction more challenging is that errors in NER may cascade into errors in relation extraction [78]. For instance, if 'Michael' or 'Dell Corporation' are not extracted by the underlying NER to begin with, then the relation extraction has little chance of deriving a relation between the two. Furthermore, inferring relations between two or more entities may require looking at multiple sources of information at the same time, which compounds its complexity. Even today, this problem of automatically detecting long-range dependencies between entities is far less studied and yields lower performance than the more typical problem of determining relations that can be contextualized fairly locally (within the span of a few sentences or a paragraph). Nevertheless, the problem continues to be considered as important within NLP and many papers continue to be published on it [91].

Once the initial information extraction steps are complete, the KG is still noisy and lacks context. This is especially the case when popular entities are present in the KG. For example, suppose that a city (such as 'Tokyo') is extracted as a node or named entity. In the data source itself, there may not be much information about Tokyo, but by 'linking' Tokyo to a canonical knowledge base like Wikipedia, we can acquire much more context and extra information about such entities. This can not only help with querying, but also help correct errors in the KG. The specific task we may therefore have to deal with at this stage is *entity linking* [85], which is the process of identifying and tagging mentions of specific objects in the text and linking them to the corresponding entities (their identifiers) in external knowledge bases, databases, or ontologies. For example, in the text `Jan Kowalski has seen Chicago`, an entity linking system could tag the entities `Jan Kowalski` and `Chicago` and link them to the corresponding pages in Wikipedia or identifiers in Wikidata. Sometimes there can be ambiguity as to which specific entity a mention should be linked to. The entity linking system has to deal with the problem of determining whether the text `Chicago` refers to a city, a musical or a movie. Similarly, it may not be certain whether the mention of "Jan Kowalski" refers to an athlete, a writer, and so on. In such situations, the system must take this ambiguity into account and help select the appropriate entity to link to the mention in the text.

Once created using knowledge extraction methods, the knowledge graph may contain a lot of noisy and incomplete data. The purpose of the *knowledge graph completion* task is precisely to fill in missing information and intelligently clean the data in the knowledge graph. This is usually solved by completing missing edges through link prediction, entity deduplication (eliminating repeated entities) and dealing with missing values.

Link prediction in knowledge graphs is the problem of predicting missing relationships between entities in a knowledge graph [83]. Missing relationships can be useful to complete the linking of missing information in a knowledge graph or to improve the accuracy of various applications, such as recommender systems or question answering. Historically (and even currently), link prediction was widely studied in the network science community, including prediction of friendship links and collaboration links in social networks. In KGs, the problem is more difficult because there are many different types of nodes and edges. Hence, various types of link prediction problems can be defined. An example is *triples classification*, which is the problem of determining whether a complete named link (including both the named relation and the two named entities that the edge is incident upon) is correct or not [26]. Unlike link prediction, triples classification can be used both for determining new links but also for removing incorrect links (or triples) that are the inevitable consequence of any KG construction architecture. Another related problem is *entity classification*, which is usually aimed at more accurately predicting the *type* of the entity. In a sense, this is also a link prediction problem, but with the link existing between an entity and an ontological concept. However, because the links have specialized :type semantics, special techniques can be applied to them, as a number of authors have demonstrated [22, 54, 55].

Besides link prediction, entity resolution (ER) or deduplication is another important sub-problem in KG completion. ER may be defined as the algorithmic problem of determining when two entities refer to the same underlying entity [32, 50]. The problem has been around for more than 50 years [33, 49], but there has been much progress, especially due to the advent of deep learning methods. ER is critical to knowledge graph completion as, without it, the same entity would get over-counted and knowledge within the graph would not be reliable. It has been explored in many domains [34, 51, 58, 61, 97], but still remains a difficult problem.

There are many different approaches to solving the problem of predicting relationships in knowledge graphs, including regression-based, classification-based and ranking-based approaches. All of these approaches require learning a model on a large dataset containing existing relationships in the knowledge graph, and then using this model to predict missing relationships.

Over the years, many solutions have been proposed for these problems, but a dominant paradigm today is that of learning knowledge graph embeddings. In this approach, entities and relationships in a knowledge graph are represented as vectors in a low-dimensional vector space, and the model learns these vectors from a large dataset containing existing relationships in the knowledge graph. Then, to predict a missing relationship between two entities, the model compares the vectors of these entities and predicts how probable is that they are connected by a given relationship.

One way in which KG embeddings work is through translation i.e., suppose that in a triple (h, r, t), the embedding for the head entity (h), tail entity (t), and relation (r) is represented as $\vec{h}$, $\vec{t}$ and $\vec{r}$, respectively. A translation-based embedding algorithm (such as TransE; see next section) learns these embeddings by optimizing the translation function $\vec{h} + \vec{r} = \vec{t}$. By encoding this function for each triple in the training dataset and optimizing it using a machine learning technique like stochastic gradient descent, embeddings that obey this relation approximately are derived. These embeddings can then be used in other machine learning applications, like link prediction and ER.

6.2. *Knowledge graph representation learning*

Representation learning involves embedding a data element, which can be, for example, a piece of text, an entity, a relation in a vector space. *Knowledge graph embedding* is performed using supervised machine learning on a large dataset of triples to project knowledge graph components onto a continuous and low-dimensional vector space. The aim of knowledge graph embedding is to capture the semantics of entities and relationships in the knowledge graph in a way that will facilitate use in a variety of tasks, be it tasks related to construction of the knowledge graph (such as its completion) or downstream tasks such as its use in recommender systems.

In particular, for any pair $s, o \subseteq \mathcal{E}$ and relation $p \in \mathcal{R}$, it can be determined whether the sentence (s, p, o) is true according to the data embeddings of the knowledge graph.

The knowledge graph embedding model consists of:

- a knowledge graph $\mathcal{G}$,
- a strategy for generating negative examples,
- an evaluation function of the triple $f(t)$,
- a loss function $\mathcal{L}$,
- a lookup layer,
- an optimization algorithm.

The architecture of such a solution, depicted by the authors of one of the popular libraries for learning knowledge graph representations [24], is shown in Figure 8.

A number of different approaches to training knowledge graph embeddings have been proposed, including translation-based approaches, tensor-based approaches and graph-based approaches [99]. For example, the TransE model, one of the first translation-based embedding methods to have been proposed, works on the simple principle that the combination of subject and relation vectors should ideally be equal to the object vector. TransE is able to learn composition, inverse and antisymmetry. Other Trans* algorithms build upon TransE by encoding more sophisticated notion of translation. For example, TransH represents relations as hyperplanes [102], rather than as simple vectors, to encode their properties in a more geometrically interesting manner. As is often the case with such learning-based algorithms, their effectiveness is judged by their empirical performance on real-world benchmarks, rather than on a theoretical basis. One problem with the more traditional translation-based algorithms is that they use local information more heavily and fail to capture global dependencies. Recently, however, the emergence of graph convolutional networks attempts to mitigate this issue by considering greater non-locality [108]. Other such algorithms rely on paths, rather than triples, in learning the embeddings, not dissimilar from random walk-based embedding algorithms that have become popular in the network science community.

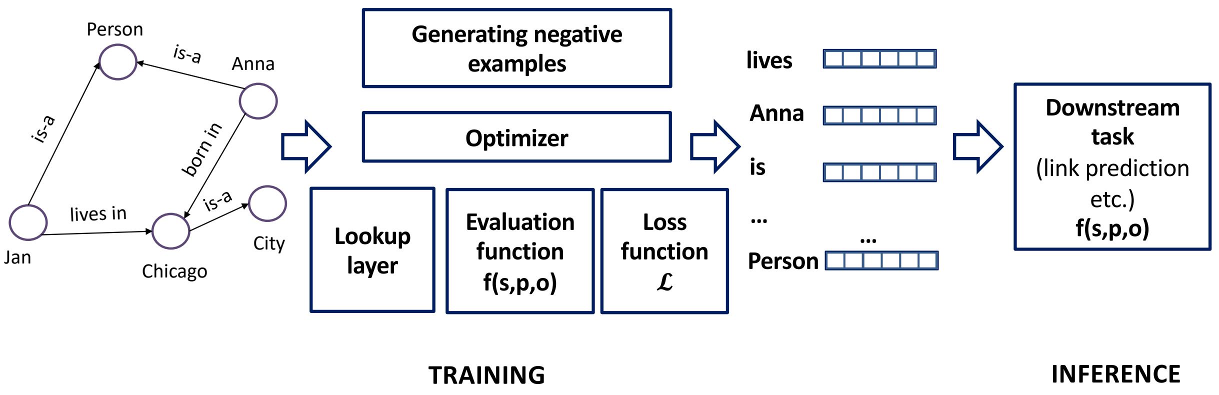

Fig. 8. Architecture of a system for creating knowledge graph embeddings.

6.3. Validation and quality of knowledge graphs

The graph model employed by Knowledge graphs promotes a flexible model, which allows users to add new entities and relationships in an easy way, with the illusion of a schema-less system. Although this flexibility can be beneficial to help the adoption of knowledge graphs by facilitating the collaborative addition of content, it can also be a problem when the quality of the data is important [86, 106].

Shape Expressions (ShEx) was proposed in 2014 as a concise and human-readable language to describe and validate RDF [81]. SHACL (Shapes Constraint Language) was later proposed as a W3C recommendation in 2017 [59]. Both ShEx and SHACL are based on the notion of shapes which declare constraints on the neighbourhood of a node and can be visualized in UML-like diagrams where a box represents a shape.

As an example, a schema for the knowledge graph in figure 6 is visualized in figure 9[3]. In this case, the schema defines three shapes: `:Researcher`, `:Discipline` and `:Place`. A node that conforms to the `:Researcher` shape must have a property `edu:birthDate` whose value must be of type `xsd:date` and another property `edu:birthPlace` whose value must conform to the shape `:Place`, and one or more properties `edu:discipline` whose nodes conform to the shape `:Discipline`, in this case it is a list of values `edu:Physics`, `edu:Chemistry`, etc.

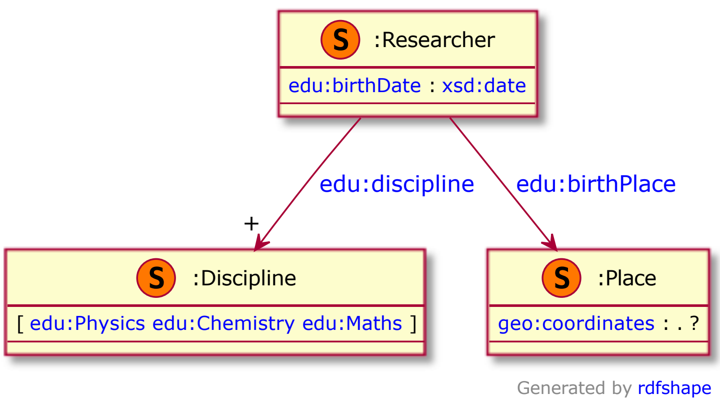

Fig. 9. Example of a shapes schema

In order to keep the flexible nature of Knowledge Graphs, the validation of nodes is not always mandatory and shapes are more descriptive than prescriptive. For example, Wikidata adopted ShEx in 2019 in the entity schemas

[3]The visualization has been automatically generated using rdfshape: https://rdfshape.weso.es/link/17225900064

namespace and there is a directory of shapes[4] which serve different purposes from documenting the expected content of entities to describe subsets of Wikidata [45].

A formal definition of ShEx was proposed in [15]. In the case of SHACL, the SHACL specification left recursion undefined, but a proposal that combined recursion and negation was presented in [23] with several other works proposing some semantic variants based on stable model semantics [3] or fixpoint semantics [14].

7. Knowledge engineering methodology

There are a number of methodological issues and best practices associated with knowledge engineering. Several of these are discussed below.

7.1. URIs and multilinguality

We can adopt two main naming conventions when specifying a URI for a resource [65]. One is to name the resource directly in its URI, i.e., insert the resource name in the URI string, such as `geo:Warsaw`. The advantages of descriptive URIs are simplicity, ease of interpretation and greater availability of tools that support such a format without problems.

The second convention is to create an alphanumeric resource identifier and put the name in the label using predefined annotation properties like `rdfs:label`, for example: `geo:locality1 rdfs:label "Warsaw"`. The advantages of this approach, called opaque URIs, are: facilitating multilinguality, avoiding the problems of formulating long names, which can be, for example, a concatenation of many words due to the addition of more and more detailed classes in their hierarchy (we can put any phrase in the label), ease and flexibility of changes, and preserving the stability of URI-based application builds in case the semantic meaning of a resource changes or drifts (it is then enough to change the label itself and not the identifier).

7.2. Ontology engineering methodologies

Knowledge engineering refers to the process of developing knowledge bases, ontologies and knowledge graphs. In particular, ontology engineering methodologies have been developed. Early methodologies followed the cascade model of ontology development, in which requirements and general conceptualization were established before ontology definition began. However, nowadays we often deal with large or constantly evolving ontologies, hence the subsequent methodologies that have been proposed promote more iterative and agile ways of building and maintaining ontologies [48].

Modern methodologies propose various best practices like reusing already existing resources and creating as a result a network of ontologies that import selected classes, properties or entire modules among themselves [90]. They often contain two common elements: ontology requirements and ontology design patterns [29].

A common way of expressing the requirements that an ontology must meet includes *competency questions*, which are natural language questions that an ontology or knowledge graph should be prepared to answer, i.e., at least contain the appropriate vocabulary.

Ontology Design Patterns define general ontology modeling patterns (modeling templates) that can be used as inspiration for modeling more specific phenomena. An example would be a pattern defining arbitrary events, which models, among other things, the spatial and temporal scope, the participants of the event, and is supplemented with competency questions and other documentation elements.

[4] https://www.wikidata.org/wiki/Wikidata:Database_reports/EntitySchema_directory

7.3. FAIR principles

FAIR Principles were originally proposed in the context of publishing scientific data [104], but are generally applicable to any situation where data is to be open, accessible, and published in a way that facilitates its reuse by external parties, with a particular emphasis on facilitating its processing in information systems. The FAIR acronym refers to four fundamental principles for data, metadata or both, each with specific purposes. Below is a brief discussion of each of the FAIR principles:

- *findable*: research results should be discoverable and easy to locate, using persistent identifiers (e.g., DOIs) and metadata that accurately and comprehensively describe the content.
- *accessible*: research results should be available to anyone who wants to access them, regardless of location or ability to pay. This can be achieved through open access publishing or other mechanisms that provide free and unrestricted access.
- *interoperable*: research results should be structured and formatted in a way that allows them to be easily integrated and linked with other data and research results. This requires the use of common standards and protocols.
- *reusable*: research results should be licensed in a way that allows them to be reused and built upon by others, subject to proper attribution and citation.

Adhering to FAIR can help make scientific research more transparent, reproducible and influential, and as a result can advance science and benefit society.

8. LLMs and Knowledge Engineering

As we described in Section 6.1, creating and populating knowledge bases requires knowledge extraction from different data formats. Traditionally, this requires a fixed data schema and the application of NLP methods in several steps, where each step can propagate errors to the next step. Petroni et al. [80] have shown that instead of the classical approach, *large language models (LLMs)* can be used as a source of knowledge. Such models, sometimes called *Foundation Models*, are deep neural networks scaled to billions of parameters on the task of predicting the next word on a large corpus and store the knowledge that was contained in the training data implicitly. They can generalise to new tasks without fine-tuning to answer questions structured as "fill-in-the-blank" cloze statements, such as: "`The colour of a carrot is [MASK]`".

This approach does not require manual engineering of the knowledge schema or human annotation of the data to extract relatively good quality knowledge. Narayan et al. [76] have shown how knowledge cleaning and integration tasks, including entity matching, can be performed by reformulating them as prompting tasks. For example, they examined the answer to the question "Are products A and B the same?" and the language model generated a string "Yes" or "No" as the answer. Other recent work that has considered using LLMs for entity matching and resolution include [66, 67, 75]. In a similar fashion, LLMs are also being increasingly considered or adapted for tasks that traditionally required significant investments in engineering natural language processing pipelines. Such tasks include information extraction and named entity recognition [73, 100], co-reference resolution [28, 70], and even knowledge graph identification [42, 69]. Much more recently, there have also been significant advancements on (previously difficult) tasks like text-to-SQL and text-to-Cypher, which would allow people without knowledge of formal querying languages to access knowledge graphs, with an LLM mediating the conversion of plain text to the formal query language [39, 46, 92, 107].

8.1. Prompt engineering

Prompt engineering (*in-context prompting*) [9] concerns methods of communicating with LLMs to get desired answers. The weights of the model stay unchanged. The models can be trained to learn a task in a few-shot manner (with minimal task description) or even in a zero-shot learning setting [19]. Prompt engineering is mostly an experimental field of study. The terminology is relatively simple, and introduced next. *Prompt* is a conditioning text before the test input. Zero-shot learning consists of simply providing a text to get the answer. *Demonstrations* is an instance of the prompt, which is a concatenation of the *k*-shot training data. Few-shot learning consists of presenting a set of demonstrations composed of input and envisaged output, for instance:

```
Lemon: yellow
Carrot: orange
Raspberry: raspberry red
Pear:
```

Pattern is a function mapping an input to the text. *Verbalizer* is a function mapping a label to the text.

Since the release of the ChatGPT interface, prompt engineering has become increasingly sophisticated, leading to development of entire families of methods such as *Chain-of-Thought* (CoT) prompting [27, 101, 103]. When prompted using CoT, the model is encouraged to think through a problem step by step. This method aims to improve the LLM's problem-solving and reasoning abilities by making its thought process more explicit and logical. By breaking down complex tasks into smaller, manageable steps, CoT is found to improve the accuracy and reliability of the model's responses, which can be especially useful in scenarios requiring multi-step reasoning, such as mathematical problem solving, decision making, and complex question answering [21].

While prompt engineering can make a difference in quality, constructing large-scale KGs at scale still requires the issue of cost to be properly addressed. Even though an individual prompt issued against a commercial LLM seems inexpensive, issuing tens of thousands of prompts can still ramp up costs substantially. Recently, some authors have been considering effective ways of balancing costs and quality (e.g., on tasks like entity matching) [60, 68, 87], but this is still an open area of research.

8.2. *Commonsense knowledge*

Some datasets and knowledge graphs are designed to capture commonsense knowledge, such as the knowledge of physical phenomena, that humans acquire during their lives as part of their interaction with the environment, or knowledge required in scientific computing. For example, ConceptNet [89] is a knowledge graph that connects structured knowledge to natural language, bridging the gap between formal knowledge representation and natural language. It connects words and phrases with labeled edges. It gathers knowledge from the resources developed by experts, crowd-sourcing, and games with a purpose. Further linking this graph with text embeddings helps to solve tasks such as SAT-style analogies more efficiently than with resources based primarily on formalised knowledge structure.

Benchmarks designed to evaluate commonsense reasoning often come with the task of question answering. For instance, PIQA ("Physical Interaction – Question Answering") [12] is a dataset related to best achieving a goal in everyday tasks such as crafting, baking, or manipulating objects using everyday materials. The user has to choose one of the two answers concerning how to achieve the goal best. One of the answers is the right one. For instance, if one asks how to eat soup, the correct answer is to use a spoon rather than a fork. Answering such questions requires knowledge that may not be represented explicitly as, e.g., attributes of some object (as factual knowledge) and also might depend on context. Consider, for instance, querying an LLM on the functions (roles) of some ingredients in a dish (implicit and contextual knowledge):

```
sugar [DISH] cake [FUNCTION] sweetener
baking powder [DISH] cake [FUNCTION] leavening agent
egg yolk [DISH] Hollandaise sauce [FUNCTION] emulsifier
yeast [DISH] bread [FUNCTION]
```

Answering such queries may require the model to understand food technological processes. Combining domain knowledge and common sense knowledge is an important area of research, and one where LLMs and knowledge graphs may be able to play a synergistic role. LLMs may be able to provide good common sense knowledge because of their ingestion of massive data sources from the Web, and the natural language nature of much of human common sense relevant to KG applications. However, LLMs are also prone to problems like *hallucinations* [4, 31]. These are especially problematic in high-stakes applications, and may not be infrequent, as the recent publication of the HALO ontology indicates [74]. There is some evidence that an appropriate use of knowledge graphs and engineering may help in reducing hallucinations in LLMs [1, 37], although much more research on this topic is likely forthcoming at the time of writing.

9. Further reading

A more detailed description of the RDF model can be found in Heath and Bizer's book "Linked Data: Evolving the Web into a Global Data Space" [40].

A concise introduction to description logic can be found in the article by Krötzsch et al. [64]. A comprehensive textbook on description logic is by Baader et al. [7]. Issues related to description logics are also discussed in a book by Hitzler et al. "Foundations of Semantic Web Technologies" [41], on the theoretical foundations of the technologies.

The issues of ontology modelling in OWL from the side of engineering, good practices, design patterns are extensively presented in the books "Semantic Web for the Working Ontologist: Effective Modeling for Linked Data, RDFS, and OWL" by Allemang, Hendler and Gandon [2] and "Demystifying OWL for the Enterprise" by Uschold [94]. Knowledge engineering issues as a whole (theoretical foundations of ontology representation languages and good modelling practices) are covered in the publicly available textbook "An Introduction to Ontology Engineering" by Keet [47].

Knowledge graphs constitute a relatively new, active and interdisciplinary area of artificial intelligence that emerged around 2012, drawing from areas such as natural language processing, data mining and the Semantic Web. There are relatively recent textbooks, including "Knowledge Graphs: Fundamentals, Techniques, and Applications" by Kejriwal, Knoblock and Szekely [56] and "Knowledge Graphs" by Hogan et al. [43].

References

[1] G. Agrawal, T. Kumarage, Z. Alghami and H. Liu, Can knowledge graphs reduce hallucinations in llms?: A survey, *arXiv preprint arXiv:2311.07914* (2023).

[2] D. Allemang, J. Hendler and F. Gandon, *Semantic Web for the Working Ontologist: Effective Modeling for Linked Data, RDFS, and OWL*, Association for Computing Machinery, 2020.

[3] M. Andresel, J. Corman, M. Ortiz, J.L. Reutter, O. Savkovic and M. Simkus, Stable Model Semantics for Recursive SHACL, in: *Proceedings of The Web Conference 2020*, WWW '20, ACM, 2020. doi:10.1145/3366423.3380229.

[4] K. Andriopoulos and J. Pouwelse, Augmenting LLMs with Knowledge: A survey on hallucination prevention, *arXiv preprint arXiv:2309.16459* (2023).

[5] R. Arp, B. Smith and A.D. Spear, *Building Ontologies with Basic Formal Ontology*, The MIT Press, 2015. ISBN 0262527812.

[6] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak and Z.G. Ives, DBpedia: A Nucleus for a Web of Open Data, in: *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, K. Aberer, K. Choi, N.F. Noy, D. Allemang, K. Lee, L.J.B. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber and P. Cudré-Mauroux, eds, Lecture Notes in Computer Science, Vol. 4825, Springer, 2007, pp. 722–735. doi:10.1007/978-3-540-76298-0_52.

[7] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi and P.F. Patel-Schneider (eds), *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press, 2003. ISBN 0-521-78176-0.

[8] C.F. Baker, C.J. Fillmore and J.B. Lowe, The Berkeley FrameNet Project, in: *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 1*, ACL '98/COLING '98, Association for Computational Linguistics, USA, 1998, pp. 86–90–. doi:10.3115/980845.980860.

[9] I. Beltagy, A. Cohan, R.L.L. IV, S. Min and S. Singh, Zero- and Few-Shot NLP with Pretrained Language Models, in: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics, ACL 2022 - Tutorial Abstracts, Dublin, Ireland, May 22-27, 2022*, L. Benotti, N. Okazaki, Y. Scherrer and M. Zampieri, eds, Association for Computational Linguistics, 2022, pp. 32–37. doi:10.18653/v1/2022.acl-tutorials.6.

[10] T. Berners-Lee, J. Hendler and O. Lassila, The Semantic Web, *Scientific American* **284**(5) (2001), 34–43.

[11] T.R. Besold, A.S. d'Avila Garcez, S. Bader, H. Bowman, P.M. Domingos, P. Hitzler, K. Kühnberger, L.C. Lamb, P.M.V. Lima, L. de Penning, G. Pinkas, H. Poon and G. Zaverucha, Neural-Symbolic Learning and Reasoning: A Survey and Interpretation, in: *Neuro-Symbolic Artificial Intelligence: The State of the Art*, P. Hitzler and M.K. Sarker, eds, Frontiers in Artificial Intelligence and Applications, Vol. 342, IOS Press, 2021, pp. 1–51. doi:10.3233/FAIA210348.

[12] Y. Bisk, R. Zellers, J. Gao, Y. Choi et al., PIQA: Reasoning about physical commonsense in natural language, in: *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34, 2020, pp. 7432–7439.

[13] A. Blumauer and H. Nagy, *The Knowledge Graph Cookbook*, edition mono/monochrom, 2020.

[14] B. Bogaerts and M. Jakubowski, Fixpoint Semantics for Recursive SHACL, in: *Proceedings 37th International Conference on Logic Programming (Technical Communications), ICLP Technical Communications 2021, Porto (virtual event), 20-27th September 2021*, A. Formisano, Y.A. Liu, B. Bogaerts, A. Brik, V. Dahl, C. Dodaro, P. Fodor, G.L. Pozzato, J. Vennekens and N. Zhou, eds, EPTCS, Vol. 345, 2021, pp. 41–47. doi:10.4204/EPTCS.345.14.

[15] I. Boneva, J.E. Labra Gayo and E.G. Prud'hommeaux, Semantics and Validation of Shapes Schemas for RDF, pp. 104–120.

[16] R. Brachman and H. Levesque, *Readings in Knowledge Representation*. ISBN 9780934613019.

[17] R. Brachman, R.J.B.H.J. Levesque, H. Levesque and M. Pagnucco, *Knowledge Representation and Reasoning*. ISBN 9781558609327.

[18] R. Brachman and H. Levesque, *Knowledge Representation and Reasoning*, The Morgan Kaufmann Series in Artificial Intelligence, Morgan Kaufmann, Amsterdam, 2004. ISBN 978-1-55860-932-7. http://www.sciencedirect.com/science/book/9781558609327.

[19] T.B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D.M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever and D. Amodei, Language Models are Few-Shot Learners, *CoRR* **abs/2005.14165** (2020). https://arxiv.org/abs/2005.14165.

[20] C.-H. Chang, M. Kayed, M.R. Girgis and K.F. Shaalan, A survey of web information extraction systems, *IEEE transactions on knowledge and data engineering* **18**(10) (2006), 1411–1428.

[21] Z. Chu, J. Chen, Q. Chen, W. Yu, T. He, H. Wang, W. Peng, M. Liu, B. Qin and T. Liu, A survey of chain of thought reasoning: Advances, frontiers and future, *arXiv preprint arXiv:2309.15402* (2023).

[22] M. Collins and Y. Singer, Unsupervised models for named entity classification, in: *1999 Joint SIGDAT conference on empirical methods in natural language processing and very large corpora*, 1999.

[23] J. Corman, J.L. Reutter and O. Savković, *Semantics and Validation of Recursive SHACL*, in: *The Semantic Web – ISWC 2018*, Springer International Publishing, 2018, pp. 318–336. ISSN 1611-3349. ISBN 9783030006716. doi:10.1007/978-3-030-00671-6$_1$9.

[24] L. Costabello, S. Pai, C.L. Van, R. McGrath, N. McCarthy and P. Tabacof, AmpliGraph: a Library for Representation Learning on Knowledge Graphs, 2019. doi:10.5281/zenodo.2595043.

[25] R. Davis, B. Buchanan and E. Shortliffe, Production rules as a representation for a knowledge-based consultation program, *Artificial intelligence* **8**(1) (1977), 15–45.

[26] P. Do and T.H. Phan, Developing a BERT based triple classification model using knowledge graph embedding for question answering system, *Applied Intelligence* **52**(1) (2022), 636–651.

[27] G. Feng, B. Zhang, Y. Gu, H. Ye, D. He and L. Wang, Towards revealing the mystery behind chain of thought: a theoretical perspective, *Advances in Neural Information Processing Systems* **36** (2024).

[28] Y. Gan, M. Poesio and J. Yu, Assessing the Capabilities of Large Language Models in Coreference: An Evaluation, in: *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, 2024, pp. 1645–1665.

[29] A. Gangemi, Ontology design patterns for semantic web content, in: *The Semantic Web–ISWC 2005: 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005. Proceedings 4*, Springer, 2005, pp. 262–276.

[30] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari and L. Schneider, *Sweetening Ontologies with DOLCE*, in: *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web: 13th International Conference, EKAW 2002 Sigüenza, Spain, October 1–4, 2002 Proceedings*, A. Gómez-Pérez and V.R. Benjamins, eds, Springer, Berlin, Heidelberg, 2002, pp. 166–181. ISBN 978-3-540-45810-4. doi:10.1007/3-540-45810-7_18. http://link.springer.de/link/service/series/0558/bibs/2473/24730166.htm.

[31] Z. Gekhman, G. Yona, R. Aharoni, M. Eyal, A. Feder, R. Reichart and J. Herzig, Does Fine-Tuning LLMs on New Knowledge Encourage Hallucinations?, *arXiv preprint arXiv:2405.05904* (2024).

[32] L. Getoor and A. Machanavajjhala, Entity resolution: theory, practice & open challenges, *Proceedings of the VLDB Endowment* **5**(12) (2012), 2018–2019.

[33] L. Getoor and A. Machanavajjhala, Entity resolution for big data, in: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 1527–1527.

[34] M. Gheini and M. Kejriwal, Unsupervised Product Entity Resolution using Graph Representation Learning., in: *eCOM@ SIGIR*, 2019.

[35] R. Grishman, Information extraction, *IEEE Intelligent Systems* **30**(5) (2015), 8–15.

[36] T. Gruber, A translation approach to portable ontology specifications, *Knowledge Acquisition* (1993), 199–220.

[37] X. Guan, Y. Liu, H. Lin, Y. Lu, B. He, X. Han and L. Sun, Mitigating large language model hallucinations via autonomous knowledge graph-based retrofitting, in: *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38, 2024, pp. 18126–18134.

[38] N. Guarino, Formal Ontology and Information Systems, in: *Proceedings of Formal Ontology in Information System*, IOS Press, 1998, pp. 3–15.

[39] A. Guo, X. Li, G. Xiao, Z. Tan and X. Zhao, Spcql: A semantic parsing dataset for converting natural language into cypher, in: *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 2022, pp. 3973–3977.

[40] T. Heath and C. Bizer, *Linked Data: Evolving the Web into a Global Data Space*, Morgan & Claypool, 2011.

[41] P. Hitzler, M. Krötzsch and S. Rudolph, *Foundations of Semantic Web Technologies*, Chapman and Hall/CRC Press, 2010.

[42] M. Hofer, J. Frey and E. Rahm, Towards self-configuring knowledge graph construction pipelines using llms-a case study with rml, in: *Fifth International Workshop on Knowledge Graph Construction@ ESWC2024*, 2024.

[43] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutiérrez, S. Kirrane, J.E. Labra Gayo, R. Navigli, S. Neumaier, A.-C. Ngonga Ngomo, A. Polleres, S.M. Rashid, A. Rula, L. Schmelzeisen, J.F. Sequeda, S. Staab and A. Zimmermann, *Knowledge Graphs*, Synthesis Lectures on Data, Semantics, and Knowledge Vol. 22, Springer, 2021. ISBN 9783031007903. doi:10.2200/S01125ED1V01Y202109DSK022. https://kgbook.org/.

[44] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutierrez, S. Kirrane, J.E.L. Gayo, R. Navigli, S. Neumaier, A.N. Ngomo, A. Polleres, S.M. Rashid, A. Rula, L. Schmelzeisen, J.F. Sequeda, S. Staab and A. Zimmermann, Knowledge Graphs, *ACM Comput. Surv.* **54**(4) (2022), 71:1–71:37. doi:10.1145/3447772.

[45] S.A. Hosseini Beghaeiraveri, J.E. Labra Gayo, A. Waagmeester, A. Ammar, C. Gonzalez, D. Slenter, S. Ul-Hasan, E. Willighagen, F. McNeill and A.J.G. Gray, Wikidata subsetting: Approaches, tools, and evaluation, *Semantic Web* (2023), 1–27. doi:10.3233/sw-233491.

[46] G. Katsogiannis-Meimarakis and G. Koutrika, A survey on deep learning approaches for text-to-SQL, *The VLDB Journal* **32**(4) (2023), 905–936.

[47] C.M. Keet, *An Introduction to Ontology Engineering*, College Publications, 2018.

[48] C.M. Keet and A. Lawrynowicz, Test-Driven Development of Ontologies, in: *The Semantic Web. Latest Advances and New Domains - 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29 - June 2, 2016, Proceedings*, H. Sack, E. Blomqvist, M. d'Aquin, C. Ghidini, S.P. Ponzetto and C. Lange, eds, Lecture Notes in Computer Science, Vol. 9678, Springer, 2016, pp. 642–657. doi:10.1007/978-3-319-34129-3_39.

[49] M. Kejriwal, Populating entity name systems for big data integration, in: *The Semantic Web–ISWC 2014: 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part II 13*, Springer, 2014, pp. 521–528.

[50] M. Kejriwal, Entity resolution in a big data framework, in: *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 29, 2015.

[51] M. Kejriwal, Named Entity Resolution in Personal Knowledge Graphs, *arXiv preprint arXiv:2307.12173* (2023).

[52] M. Kejriwal and M. Kejriwal, Information Extraction, *Domain-Specific Knowledge Graph Construction* (2019), 9–31.

[53] M. Kejriwal and P. Szekely, Information extraction in illicit web domains, in: *Proceedings of the 26th international conference on world wide web*, 2017, pp. 997–1006.

[54] M. Kejriwal and P. Szekely, Scalable generation of type embeddings using the abox, *Open Journal of Semantic Web (OJSW)* **4**(1) (2017), 20–34.

[55] M. Kejriwal and P. Szekely, Supervised typing of big graphs using semantic embeddings, in: *Proceedings of the international workshop on semantic big data*, 2017, pp. 1–6.

[56] M. Kejriwal, C. Knoblock and P. Szekely, *Knowledge Graphs: Fundamentals, Techniques, and Applications*, The MIT Press, 2021.

[57] M. Kejriwal, R. Shao and P. Szekely, Expert-guided entity extraction using expressive rules, in: *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval*, 2019, pp. 1353–1356.

[58] M. Kejriwal, J. Peng, H. Zhang and P. Szekely, Structured event entity resolution in humanitarian domains, in: *The Semantic Web–ISWC 2018: 17th International Semantic Web Conference, Monterey, CA, USA, October 8–12, 2018, Proceedings, Part I 17*, Springer, 2018, pp. 233–249.

[59] H. Knublauch and D. Kontokostas, Shapes Constraint Language (SHACL), W3C Recommendation 20 July 2017, W3C Recommendation, 2017. https://www.w3.org/TR/2017/REC-shacl-20170720/.

[60] V.K. Kommineni, B. König-Ries and S. Samuel, From human experts to machines: An LLM supported approach to ontology and knowledge graph construction, *arXiv preprint arXiv:2403.08345* (2024).

[61] P. Kouki, J. Pujara, C. Marcum, L. Koehly and L. Getoor, Collective entity resolution in familial networks, in: *2017 IEEE International Conference on Data Mining (ICDM)*, IEEE, 2017, pp. 227–236.

[62] R. Kowalski, *Computational Logic and Human Thinking: How to be Artificially Intelligent*, 1st edn, Cambridge University Press, USA, 2011. ISBN 0521123364.

[63] R. Kowalski, *Logic for Problem Solving, Revisited*, Computer science essentials, Books on Demand, 2014. ISBN 9783837036299. https://books.google.fr/books?id=6vh1BQAAQBAJ.

[64] M. Krotzsch, F. Simancik and I. Horrocks, Description Logics, *IEEE Intelligent Systems* (2014), 12–19.

[65] J.E. Labra Gayo, D. Kontokostas and S. Auer, Multilingual linked data patterns, *Semantic Web* **6**(4) (2015), 319–337. doi:10.3233/sw-140136.

[66] H. Li, S. Li, F. Hao, C.J. Zhang, Y. Song and L. Chen, BoostER: Leveraging Large Language Models for Enhancing Entity Resolution, in: *Companion Proceedings of the ACM on Web Conference 2024*, 2024, pp. 1043–1046.

[67] H. Li, L. Feng, S. Li, F. Hao, C.J. Zhang, Y. Song and L. Chen, On leveraging large language models for enhancing entity resolution, *arXiv preprint arXiv:2401.03426* (2024).

[68] Z. Li, L. Deng, H. Liu, Q. Liu and J. Du, UniOQA: A Unified Framework for Knowledge Graph Question Answering with Large Language Models, *arXiv preprint arXiv:2406.02110* (2024).

[69] R. Luo, T. Gu, H. Li, J. Li, Z. Lin, J. Li and Y. Yang, Chain of history: Learning and forecasting with llms for temporal knowledge graph completion, *arXiv preprint arXiv:2401.06072* (2024).

[70] Q. Min, Q. Guo, X. Hu, S. Huang, Z. Zhang and Y. Zhang, Synergetic Event Understanding: A Collaborative Approach to Cross-Document Event Coreference Resolution with Large Language Models, *arXiv preprint arXiv:2406.02148* (2024).

[71] M. Minsky and J. Lee, *Society Of Mind*. ISBN 9780671657130.

[72] M. Minsky, A Framework for Representing Knowledge, Technical Report, USA, 1974.

[73] M. Monajatipoor, J. Yang, J. Stremmel, M. Emami, F. Mohaghegh, M. Rouhsedaghat and K.-W. Chang, LLMs in Biomedicine: A study on clinical Named Entity Recognition, *arXiv preprint arXiv:2404.07376* (2024).

[74] N. Nananukul and M. Kejriwal, HALO: an ontology for representing and categorizing hallucinations in large language models, in: *Disruptive Technologies in Information Sciences VIII*, Vol. 13058, SPIE, 2024, pp. 86–100.
[75] N. Nananukul, K. Sisaengsuwanchai and M. Kejriwal, Cost-Efficient Prompt Engineering for Unsupervised Entity Resolution (2024).
[76] A. Narayan, I. Chami, L.J. Orr and C. Ré, Can Foundation Models Wrangle Your Data?, *Proc. VLDB Endow.* **16**(4) (2022), 738–746. https://www.vldb.org/pvldb/vol16/p738-narayan.pdf.
[77] A. Newell and H. Simon, The logic theory machine–A complex information processing system, *IRE Transactions on information theory* **2**(3) (1956), 61–79.
[78] T.H. Nguyen and R. Grishman, Relation extraction: Perspective from convolutional neural networks, in: *Proceedings of the 1st workshop on vector space modeling for natural language processing*, 2015, pp. 39–48.
[79] I. Nonaka and H. Takeuchi, *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*, Oxford University Press, 1995.
[80] F. Petroni, T. Rocktäschel, S. Riedel, P.S.H. Lewis, A. Bakhtin, Y. Wu and A.H. Miller, Language Models as Knowledge Bases?, in: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, K. Inui, J. Jiang, V. Ng and X. Wan, eds, Association for Computational Linguistics, 2019, pp. 2463–2473. doi:10.18653/v1/D19-1250.
[81] E. Prud'hommeaux, J.E. Labra Gayo and H. Solbrig, Shape Expressions: An RDF Validation and Transformation Language, pp. 32–40. doi:10.1145/2660517.2660523.
[82] R.H. Richens, Preprogramming for mechanical translation., *Mech. Transl. Comput. Linguistics* **3**(1) (1956), 20–25.
[83] A. Rossi, D. Firmani, P. Merialdo and T. Teofili, Explaining link prediction systems based on knowledge graph embeddings, in: *Proceedings of the 2022 international conference on management of data*, 2022, pp. 2062–2075.
[84] S. Sarawagi et al., Information extraction, *Foundations and Trends® in Databases* **1**(3) (2008), 261–377.
[85] Ö. Sevgili, A. Shelmanov, M. Arkhipov, A. Panchenko and C. Biemann, Neural entity linking: A survey of models based on deep learning, *Semantic Web* **13**(3) (2022), 527–570.
[86] K. Shenoy, F. Ilievski, D. Garijo, D. Schwabe and P. Szekely, A Study of the Quality of Wikidata, *Journal of Web Semantics* **72** (2022), 100679. doi:https://doi.org/10.1016/j.websem.2021.100679.
[87] K. Sisaengsuwanchai, N. Nananukul and M. Kejriwal, How does prompt engineering affect ChatGPT performance on unsupervised entity resolution?, *arXiv preprint arXiv:2310.06174* (2023).
[88] J.F. Sowa, *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. ISBN 9780534949655.
[89] R. Speer, J. Chin and C. Havasi, ConceptNet 5.5: An Open Multilingual Graph of General Knowledge, in: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, S. Singh and S. Markovitch, eds, AAAI Press, 2017, pp. 4444–4451. http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14972.
[90] M.C. Suárez-Figueroa, A. Gómez-Pérez, E. Motta and A. Gangemi, Introduction: Ontology Engineering in a Networked World, in: *Ontology Engineering in a Networked World*, M.C. Suárez-Figueroa, A. Gómez-Pérez, E. Motta and A. Gangemi, eds, Springer, 2012, pp. 1–6. doi:10.1007/978-3-642-24794-1_1.
[91] H. Sun and R. Grishman, Lexicalized Dependency Paths Based Supervised Learning for Relation Extraction., *Computer Systems Science & Engineering* **43**(3) (2022).
[92] R. Sun, S.Ö. Arik, A. Muzio, L. Miculicich, S. Gundabathula, P. Yin, H. Dai, H. Nakhost, R. Sinha, Z. Wang et al., SQL-PaLM: Improved Large Language Model Adaptation for Text-to-SQL (extended), *arXiv preprint arXiv:2306.00739* (2023).
[93] T.P. Tanon, G. Weikum and F.M. Suchanek, YAGO 4: A Reason-able Knowledge Base, in: *The Semantic Web - 17th International Conference, ESWC 2020, Heraklion, Crete, Greece, May 31-June 4, 2020, Proceedings*, A. Harth, S. Kirrane, A.N. Ngomo, H. Paulheim, A. Rula, A.L. Gentile, P. Haase and M. Cochez, eds, Lecture Notes in Computer Science, Vol. 12123, Springer, 2020, pp. 583–596. doi:10.1007/978-3-030-49461-2_34.
[94] M. Uschold, *Demystifying OWL for the Enterprise*, Morgan & Claypool Publishers, 2018.
[95] M. Uschold and M. Gruninger, Ontologies: principles, methods and applications, *The Knowledge Engineering Review* (1996), 93–136.
[96] F. van Harmelen, V. Lifschitz and B. Porter, *Handbook of Knowledge Representation*, Elsevier Science, 2008. ISBN 9780080557021.
[97] K. Vidhya, R. Soorya, N. Saranavan, T. Geetha and M. Singaravelan, Entity resolution for symptom vs disease for top-K treatments, in: *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*, IEEE, 2017, pp. 1–8.
[98] D. Vrandečić, L. Pintscher and M. Krötzsch, Wikidata: The Making Of, in: *Companion Proceedings of the ACM Web Conference 2023*, WWW'23 Companion, Association for Computing Machinery, New York, NY, USA, 2023, pp. 615–624–. ISBN 9781450394192. doi:10.1145/3543873.3585579.
[99] Q. Wang, Z. Mao, B. Wang and L. Guo, Knowledge graph embedding: A survey of approaches and applications, *IEEE Transactions on Knowledge and Data Engineering* **29**(12) (2017), 2724–2743.
[100] S. Wang, X. Sun, X. Li, R. Ouyang, F. Wu, T. Zhang, J. Li and G. Wang, Gpt-ner: Named entity recognition via large language models, *arXiv preprint arXiv:2304.10428* (2023).
[101] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery and D. Zhou, Self-consistency improves chain of thought reasoning in language models, *arXiv preprint arXiv:2203.11171* (2022).
[102] Z. Wang, J. Zhang, J. Feng and Z. Chen, Knowledge graph embedding by translating on hyperplanes, in: *Proceedings of the AAAI conference on artificial intelligence*, Vol. 28, 2014.
[103] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q.V. Le, D. Zhou et al., Chain-of-thought prompting elicits reasoning in large language models, *Advances in neural information processing systems* **35** (2022), 24824–24837.

[104] M.D. Wilkinson., M. Dumontier, I.J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L.B. da Silva Santos, P.E. Bourne, J. Bouwman, A.J. Brookes, T. Clark, M. Crosas, I. Dillo, O. Dumon, S. Edmunds, C.T. Evelo, R. Finkers, A. Gonzalez-Beltran, A.J.G. Gray, P. Groth, C. Goble, J.S. Grethe, J. Heringa, P.A.C. 't Hoen, R. Hooft, T. Kuhn, R. Kok, J. Kok, S.J. Lusher, M.E. Martone, A. Mons, A.L. Packer, B. Persson, P. Rocca-Serra, M. Roos, R. van Schaik, S.-A. Sansone, E. Schultes, T. Sengstag, T. Slater, G. Strawn, M.A. Swertz, M. Thompson, J. van der Lei, E. van Mulligen, J. Velterop, A. Waagmeester, P. Wittenburg, K. Wolstencroft, J. Zhao and B. Mons, The FAIR Guiding Principles for scientific data management and stewardship, *Scientific data* (2016), 1–9.

[105] *The Description Logic Handbook: Theory, Implementation and Applications*, 2nd edn, Cambridge University Press, 2007. doi:10.1017/CBO9780511711787.

[106] A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann and S. Auer, Quality assessment for Linked Data:A Survey: A systematic literature review and conceptual framework, *Semantic Web* **7**(1) (2015), 63–93. doi:10.3233/sw-150175.

[107] B. Zhang, Y. Ye, G. Du, X. Hu, Z. Li, S. Yang, C.H. Liu, R. Zhao, Z. Li and H. Mao, Benchmarking the text-to-sql capability of large language models: A comprehensive evaluation, *arXiv preprint arXiv:2403.02951* (2024).

[108] S. Zhang, H. Tong, J. Xu and R. Maciejewski, Graph convolutional networks: a comprehensive review, *Computational Social Networks* **6**(1) (2019), 1–23.

[109]