POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

KlevR and DeepR: a benchmark for exploring deductive reasoning functionalities of variational autoencoders

Tesi di Laurea Magistrale in
Computer science and Engineering
Ingegneria Informatica

Author: **Amedeo Pachera**

Student ID: 940134
Advisor: Prof. Emanuele Della Valle
Co-advisors: Prof. Riccardo Tommasini
Academic Year: 2021-2022

Abstract

In the context of Artificial intelligence, approximate reasoning is a well-known technique that aims at improving the efficiency of deductive artificial intelligence by trading inference correctness for performance (e.g., execution time). In particular, prominent examples of approximate reasoning include, but are not limited to approximate consistency checking using linear classifiers, relational learning with Machine Learning models (e.g. RESCAL and TransE), and Knowledge Graph completion using Deep Learning techniques (e.g. ConvE). To the best of our knowledge, none of the existing approximation techniques has been applied to materialization, i.e., the task of computing all the entailed assertions given a knowledge base. A major obstacle is the lack of a benchmark for the task that enables the investigation of expressiveness and efficiency trade-off. In this thesis work, we address these problems by designing a full-fledged benchmark named *KlevR* that is based on an accepted deep learning dataset called CLEVR. In particular, *KlevR* is a knowledge graph that includes three OWL ontologies (one for each OWL profile) that captures the CLEVR abstractions, scenes, and queries. We tested *KlevR* using a baseline model set named *DeepR*, which performs the approximate materialization task. In particular, *DeepR* is a set of deep learning models performing approximate materialization for each OWL profile using a tensor representation for *KlevR*. Considering the generative nature of the materialization task (adding new assertions), we designed DeepR as a generative model based on Variational Autoencoders. Moreover, as the embedding model for the tensor, we opted for a triple-based subject-predicate-object, since it allows a fair reconstruction of the knowledge graph structure from the vector representations. Finally, we tested *KlevR* using *DeepR* and verified a set of metrics. The result of our work is a synthetic and scalable benchmark for testing Deep Learning models addressing future investigations about the approximate materialization task.

Keywords: approximate reasoning, variational autoencoder , Knowledge Graph.

Abstract in lingua italiana

Nel contesto dell'intelligenza artificiale, il ragionamento approssimativo è una tecnica ben nota che mira a migliorare l'efficienza della deduzione, in termini di una ridotta precisione a fronte di una migliore performance (in termini di tempo di esecuzione). In particolare, esempi rilevanti di ragionamento approssimativo includono, ma non si limitano, all'approssimazione del task di consistency checking utilizzando classificatori lineari, al relational learning con modelli di Machine Learning (es. RESCAL e TransE) e al completamento di grafi di conoscenza utilizzando tecniche di Deep Learning (es. ConvE). Allo stato delle attuali conoscenze e delle informazioni a nostra disposizione, nessuna delle tecniche esistenti è stata applicata alla materializzazione, cioè il task di calcolare tutte le asserzioni implicate data una base di conoscenza. Uno dei principali ostacoli è la mancanza di un benchmark di riferimento per il task di materializzazione approssimata, che consenta di indagare l'espressività e il compromesso di efficienza. Nel seguente lavoro di tesi, affrontiamo questi problemi progettando *KlevR*, un benchmark completo basato su un dataset noto nell'ambito del Deep Learning chiamato CLEVR. In particolare, *KlevR* è un grafo di conoscenza scritto in OWL2 che include tre ontologie OWL (una per ogni profilo), e che cattura le astrazioni di CLEVR, le sue scene e domande. Abbiamo testato il benchmark con un insieme di modelli di Deep Learning chiamato *DeepR*, che approssima la materializzazione per ciascun profilo OWL, utilizzando un modello tensoriale di embedding per *KlevR*. Considerando la natura generativa del compito di materializzazione (aggiungere nuove asserzioni), abbiamo progettato *DeepR* con un modello generativo basato su Variational Autoencoder. Inoltre, come modello di embedding per i tensori, abbiamo optato per un modello basato su triple soggetto-predicato-oggetto, poiché consente una fedele ricostruzione della struttura del grafo di conoscenza dalle rappresentazioni vettoriali. Infine, abbiamo testato *KlevR* utilizzando *DeepR* e verificato una serie di metriche. Il risultato del nostro lavoro consiste in un benchmark sintetico e scalabile per il training di modelli di Deep Learning utilizzabili per approfondire il task di materializzazione approssimata.

Parole chiave: ragionamento approssimativo, autoencoder variazionale, grafo di conoscenza.

Contents

List of Figures

List of Tables

1 | Introduction

This chapter introduces our work. Section 1.1 provides the motivations that led to this thesis, Section 1.2 presents the related research questions, Section 1.3 describes the contributions offered by this work, and finally, Section 1.4 describes the thesis' general outline.

1.1. Motivations

Knowledge can be defined as the information available to an agent for effective action. Gilbert Ryle in *The Concept of Mind* (1949), distinguished two types of knowledge:

- *Knowing how* (procedural knowledge), i.e., the set of skills and abilities to perform tasks of different complexity.
- *Knowing that* (declarative knowledge), which is the conceptual knowledge that supports reasoning and can be verbalized and communicated using natural language or specialized symbolic systems.

Knowing how is immediately available for action planning, *knowing that* instead becomes available for action through reasoning, which is the process of inference made by the mind.

Figure 1.1 shows the three types of inference that are usually distinguished:

- *Deductive reasoning* (or deduction): the conclusions are entailed by the premises.
- *Abductive reasoning* (or abduction): an inference from certain observed effects to their plausible causes.
- *Inductive reasoning* (or induction): an inference from specific data to general rules.

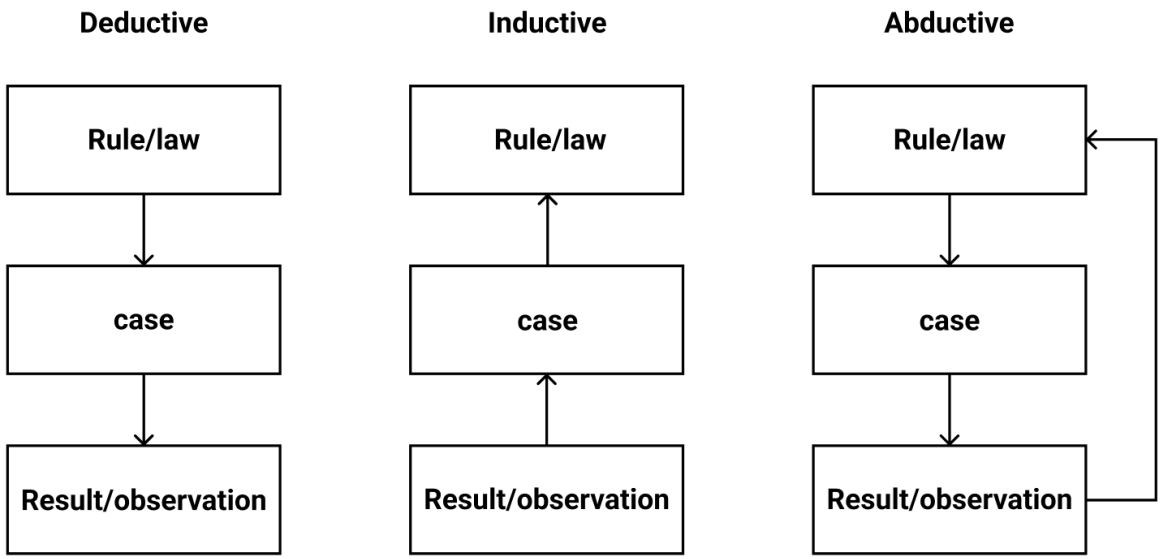


Figure 1.1: Schematic representation the types of inference.

Our work focuses on deductive and inductive reasoning. In particular, deduction entails valid conclusions from valid premises, while induction generalizes from examples. It is well known that induction does not guarantee the validity of a conclusion. Alas, if an argument is strong, the probability of the premise would mean the conclusion is likely.

In the deductive reasoning context, well-known formalisms to describe knowledge are the Description Logics (DLs). DLs are formal languages used to construct ontologies. They allow the encoding of knowledge about specific domains and often include reasoning rules that support the processing of that knowledge. Reasoning with DLs consists of three main tasks: consistency checking, classification, and materialization. There exists a trade-off between the expressiveness of a DL and its reasoning cost. Different languages and technologies explore the trade-off by maximizing the expressiveness under some limitations on the reasoning effort.

For what concerns induction, the most reliable approaches are based on statistics and Machine/Deep Learning. These methods use data to approximate functions and perform probabilistic inference. In particular, Machine and Deep Learning are good examples of inductive inferences. The target function to approximate is the general rule, while data are the specific examples that have to be generalized. In the human mind, the more something happens, the higher is the probability that the fact will repeat in the future. With the same concept, pattern and regularities in the data are extracted by models and

used to generalize future observations.

Induction and deduction can be combined together. A first approach consists of reducing the uncertainty of inductive methods with deduction, for example, in the knowledge-infused learning [10]. Another approach instead uses induction to increase the efficiency of the deduction. An example of the latter approach is the approximate reasoning. Nevertheless, the approaches in the literature focus only on a specific reasoning task or on a limited set of rules. Moreover, there is no benchmark that evaluates the task taking into consideration the trade-off expressiveness/reasoning effort. Our work tries to solve this problem through a synthetic benchmark, representing the dataset CLEVR [8] using Knowledge Representation standards, together with a baseline to test it made of a deep learning models set.

1.2. Research Questions

Now that we frame the problem space, we present the related research questions through the Macro-Meso-Micro framework [19] (see Figure 1.2).

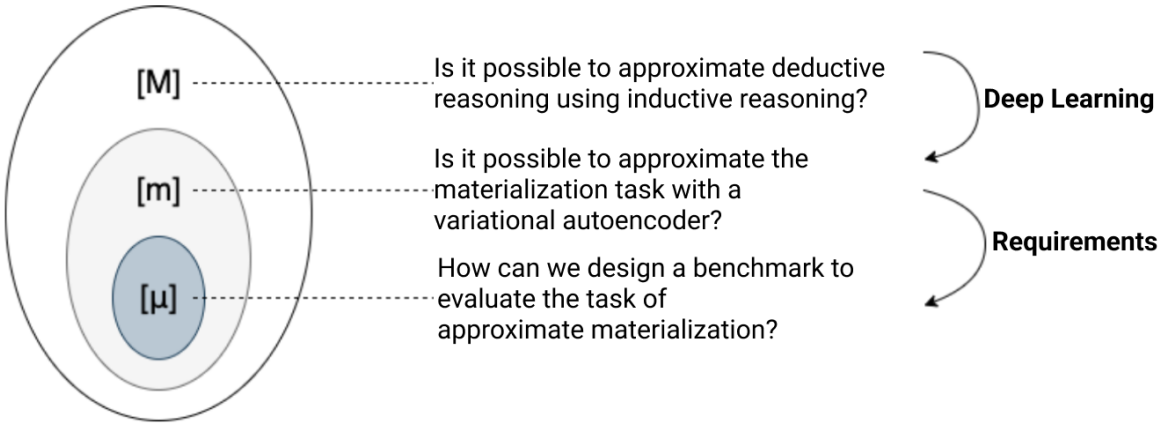


Figure 1.2: The research questions, expressed through the Macro-Meso-Micro framework.

Our macro question [M] covers numerous possible solutions, therefore, it is hard to answer. To restrict our scope we focus on the materialization of Knowledge Bases (KBs). In particular, we focus on approximate materialization using a generative Deep Learning model. This approach leads to the meso [m]. To address this problem, we elicit a set of requirements for the construction of an embedding model for the KB. In designing our experiments, we have realized the need for a benchmark that evaluates the task of approximate materialization. Thus, it comes our research micro question [μ]

1.3. Contributions

The final research question is traduced into a design problem, whose purpose is nearing the KB reasoning with Deep Learning.

In particular, the problem consists of designing a benchmark based on CLEVR to evaluate the approximation of the materialization task. CLEVR is a synthetic dataset used to test deep learning models on visual question answering. CLEVR includes a large number of training samples that guarantees a proper set for training Deep Learning models. Moreover, the entities and properties of its abstraction are already well identified, allowing a simpler KR design workflow. For this reason, we built *KlevR*, a knowledge graph that captures the CLEVR abstractions and describes its scenes. We designed *KlevR* following the KR standards (i.e., designing the OWL profiles) with the goal of maximizing the reasoning effort of the reasoners. To evaluate the quality of the benchmark, we proposed a baseline consisting of a generative Deep Learning model. Thus, we built *DeepR*, a variational autoencoder (VAE) that approximates the materialization task on *KlevR*. Moreover, we proposed SPO, an embedding model to encode the knowledge graphs describing the CLEVR scenes. Finally, we realized a proof-of-concept implementation of the benchmark and the baseline. Thanks to the synthetic nature *KlevR*, our research could be extended to deeper investigations. In particular, the benchmark allows different extensions for both the ontologies (i.e., more complex class hierarchies and property chains) and the data (i.e., the introduction of data noise).

1.4. Outline of the Thesis

- **Chapter 2**: provides the basements to understand this Thesis, with an overview of the Knowledge Representation and Deep Learning technicalities.
- **Chapter 3**: defines the research questions and the related problem addressed in this thesis.
- **Chapter 4**: describes the design process for the benchmark, modeling *KlevR* and *DeepR*.
- **Chapter 5**: describes the implementation of the benchmark and exhibits the result of the experiment.
- **Chapter 6**: draws the conclusions of this work, depicting its limitations and describing possible future works.

2 | Background

This Chapter provides an overview of the related search areas. Section 2.1 provides an overview of Knowledge Representation and reasoning domain, while Section 2.2 focuses on Semantic Web technologies. Finally, Section 2.3 describes the Deep Learning fundamentals and provides a detailed explanation of the model exploited in this Thesis.

2.1. Knowledge Representation and Reasoning

Knowledge Representation (KR) is a branch of Artificial Intelligence that aims at making information accessible to software applications in order to carry out certain computational tasks. **Knowledge bases** are repositories of knowledge containing data structures encoding relevant facts that can be made available for effective action through automated reasoning. The software in charge of carrying out new information is called *reasoner*. A logical system **L** has three components:

- *A formal language* (typically defined by a context-free grammar), which specifies different types of expressions (or formulas); an important subset of such expressions are called statements.
- *A formal semantics*, which interprets the expressions of the formal language as being about the elements of some domain of discourse. In turn, the domain of discourse is modeled as a mathematical structure belonging to a suitable structure type. A semantic interpretation assigns a truth value (either true or false) to every statement, relative to the chosen mathematical structure. The rules for assigning truth values to statements are called the truth conditions of the statement.
- *A reasoning procedure*, which allows one to build proofs of certain statements (conclusions) starting from other statements (premises).

2.1.1. Knowledge Bases

A *knowledge base* (KB) is composed of an ontology and a collection of facts. An ontology

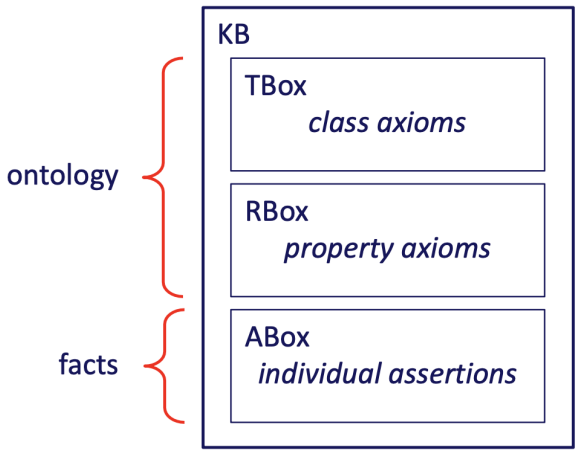


Figure 2.1: Composition of a Knowledge Base.

is a representation of a domain through formal naming and definition of its categories, properties, and relations between the concepts, data, and entities. The common components of an ontology are:

- **Individuals**: Instances or objects
- **Classes**: Sets, concepts, types of objects
- **Attributes**: Aspects, properties, characteristics, or parameters that objects may have
- **Relations**: Ways in which classes and individuals can be related to one another
- **Function terms**: Complex structures formed from certain relations that can be used in place of an individual term in a statement
- **Restrictions**: Formally stated descriptions of what must be true for some assertion to be accepted as input
- **Axioms**: Assertions in a logical form that together comprise the overall theory that the ontology describes in its domain of application.

As shown in Figure 2.1, an ontology is composed of a *TBox* ("Terminology box"), the collection of all the class axioms, and a *RBox* ("Role box"), the collection of all the property axioms, and can be written in DL (Description logic) or in OWL (see Section 2.2.3). The collection of facts instead, namely *ABox* ("assertion box"), contains all the individual assertions and can be written in RDF (see Section 2.2.1).

2.1.2. Reasoning

A knowledge base K entails statement φ when the truth of all the axioms of K is sufficient to guarantee the truth of φ. In particular:

$$K \models \varphi$$

if, and only if, every model of K satisfies φ that is, any interpretation $\mathcal{I}$ that satisfies every axiom of K ($I \models K$). On the contrary:

$$K \not\models \varphi$$

means that there is at least one interpretation $\mathcal{I}$ that satisfies every axiom of K but does not satisfy φ, and so K does not entail φ . The function of a reasoning procedure $\mathbf{R}$ is to derive from K exactly those statements that are entailed by K. To express this requirement, $\mathbf{R}$ must have the properties of *soundness* and *completeness*:

- $\mathbf{R}$ is *sound* if, and only if: if $\mathbf{R}$ derives φ from K ($K \vdash_R \varphi$), then K is guaranteed to entail φ : if $K \vdash_R \varphi$ then $K \models \varphi$
- $\mathbf{R}$ is *complete* if, and only if: if K entails φ, then $\mathbf{R}$ is guaranteed to derive φ from K in a finite number of reasoning steps: if $K \models \varphi$ then $K \vdash_R \varphi$.

An essential property of a knowledge base is *consistency*. K is *consistent* if it has at least one model, otherwise it is *inconsistent*. If K admits of no model, then a *fortiori* there is no interpretation $\mathcal{I}$ such that $\mathcal{I} \models K$ and $\mathcal{I} \not\models \varphi$, and this for every possible statement φ. This implies that every inconsistent KB entails all possible statements, which makes the KB completely useless. A reasoner usually provides a number of reasoning services, of which the most common ones are listed below. Given a knowledge base K, its classes C and D and its individual a:

- Consistency: Determine whether K is or is not consistent.
- Classification of atomic classes: Build the hierarchy of subclass relationships between all the atomic classes of K.
- Classification of individuals: Assign every individual to the most specific atomic classes to which the individual belongs.
- Subclass relationship: Establish whether $K \models C \sqsubseteq D$ or $K \not\models C \sqsubseteq D$.
- Class equivalence: Establish whether $K \models C \equiv D$ or $K \not\models C \equiv D$.

- Class satisfiability: Establish whether C is *K-satisfiable* (there is at least one model of K in which the interpetation of C is not empty).
- Instance check: Given an arbitrary class C and an individual a, establish whether a belongs to C.
- Instance retrieval: In the set of all the individuals that appear in K, find all the individuals belonging to a given class C.

2.2. Semantic web

The Semantic Web is an extension of the World Wide Web standardized by the W3C with the goal of making internet data machine-readable. The proposed framework consists of the so-called "semantic web stack" (Figure 2.2), composed of different layers with several technologies and encodings to represent data and semantics.

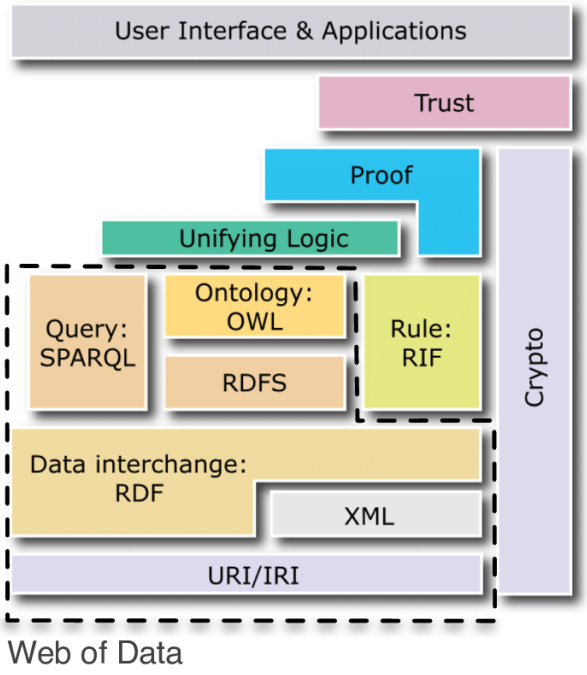


Figure 2.2: Semantic web stack. [2]

The Semantic Web involves the usage of specific languages such as Resource Description Framework (RDF), Web Ontology Language (OWL), and Extensible Markup Language (XML) designed specifically for data. In the context of this Thesis, RDF, OWL2, and SPARQL are used because they are the W3C standards languages in the field of knowledge representation.

2.2.1. RDF

RDF (Resource description framework [26]) is a directed, labeled graph data format for representing information in the semantic Web. An RDF data model is composed of triples (statements), expressions of the form

$$(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$$

Where I is a set of IRIs, L is a set of literals (couples $< String, Datatype >$) and B is a set of blank nodes, used to refer to anonymous resources. For example, to express the notion "The sun is yellow" in RDF, the triple used is

(Sun,hasColor,Yellow)

where "Sun" is the subject, "Yellow" is the object and the predicate "hasColor" represents the relationship of having that color. A collection of RDF statements can be viewed as a labeled, directed multi-graph and can be extended with RDF Schema (RDFS), which is a set of classes and properties using the RDF extensible knowledge representation data model, providing basic elements for the description of ontologies. The most common terms defined by the RDF/RDFS specification are grouped in Table 2.1.

Term	Semantics
rdf:XMLLiteral	the class of XML literal values
rdf:Property	the class of properties
rdfs:Resource	the class resource, everything
rdfs:Literal	the class of literal values, e.g. strings and integers
rdfs:Class	the class of classes
rdfs:Datatype	the class of RDF datatypes
rdfs:subClassOf	the subject is a subclass of a class
rdfs:subPropertyOf	the subject is a subproperty of a property
rdfs:domain	a domain of the subject property
rdfs:range	a range of the subject property

Table 2.1: RDF/RDFS main vocabulary.

W3C defines RDF/XML as the default serialization format for RDF models, however, other formats like Turtle (TTL), JSON for linked data (JSON-LD), or NTriples (NT) are supported. The serialization format of our choice is Turtle because it is more simple to

read. Listing 2.1 shows the same triple of the previous example, "The sun is yellow", expressed in Turtle.

```
<http://ex.org/Sun> <http://ex.org/hasColor> <http://ex.org/Yellow
   >.
```

Listing 2.1: RDF triple in Turtle format.

Turtle allows to define Base URIs or Prefixes in order to make the document more readable and to use shortcuts such as semicolons to group predicates of the same subject or commas to group objects with the same predicate. Some examples are reported in Listing 2.2.

```
<!--Example of Base URI definition-->
BASE <http://ex.org/>
<Sun> <hasColor> <Yellow> .

<!--Example of Prefix definition-->
PREFIX ex: <http://ex.org/>
ex:Sun ex:hasColor ex:Yellow .

<!--Example of semicolon shortcut to group the subject-->
ex:Sun ex:hasColor ex:Yellow;
       ex:belongsTo ex:MilkyWay.


<!--Example of comma shortcut to group the predicate-->
ex:Sun ex:hasColor ex:Yellow,
                   ex:Orange.
```

Listing 2.2: Turtle shortcuts examples.

2.2.2. RML

The RDF Mapping Language (RML [18]) is a mapping language defined to express customized mapping rules from heterogeneous data structures and serializations (CSV, TSV,

XML and JSON data sources) to the RDF data model. RML is a superset of the W3C-standardized language R2RML, it follows exactly the same syntax, but RML mappings are themselves RDF graphs. No formalization exists to define an RML mapping. Listing 2.3 shows an example of an input file in JSON, Listing 2.4 the RML mapping and Listing 2.5 shows the relative output in Turtle format.

```
#input.json
{
  "Sun":
  {
    "color": "Yellow",
  }
}
```

Listing 2.3: Example of input in JSON format.

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix rml: <http://semweb.mmlab.be/ns/rml#>.
@prefix ql: <http://semweb.mmlab.be/ns/ql#>.
@prefix ex: <http://ex.corg/>.

<#VenueMapping> a rr:TriplesMap;
  rml:logicalSource [
    rml:source "input.json";
    rml:referenceFormulation ql:JSONPath;
    rml:iterator "$"
  ];
  rr:subjectMap [
    rr:template "http://ex.org/Sun";
    rr:class ex:Star
  ];
  rr:predicateObjectMap [
    rr:predicate ex:hasColor;
    rr:objectMap [
      rml:reference "sun.color"
    ]];
```

Listing 2.4: RML mapping.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix ex: <http://ex.org/>

<http://ex.org/Sun> rdf:type ex:Star.
<http://ex.org/Sun> ex:hasColor ex:Yellow.
```

Listing 2.5: Output in Turtle format.

2.2.3. OWL

The Web Ontology Language (OWL [25]) is the knowledge representation language for the Semantic Web, and it is a standard of the W3C recommendation. Its last updated version is OWL 2, which is the default for this thesis. OWL language is characterized by formal semantics. It is built upon the RDF Schema language, extending it with additional classes and properties to design ontologies. With OWL relying on the theoretic semantics of Description Logic (DL) language, is possible to infer implicit knowledge from the already asserted axioms, which is also called reasoning (see Section 2.1.2). The main terms of the OWL vocabulary used in this thesis are reported in Table 2.2.

The first OWL version was based on SHOIN(D) Description logic, while OWL 2 is based on SROIQ(D), which differs in expressive power and reasoning complexity. OWL 2 could be uncontrollable in the worst case, but three particular fragments (called profiles) with bounded complexity have been identified.

The **OWL 2 EL** profile is designed for systems using huge ontologies with few individuals and allows performing basic reasoning in polynomial time w.r.t. the size of the ontology.

The **OWL 2 QL** profile is designed for systems using small ontologies with s huge number of individuals. The name QL reflects the intent of designing a query-answering-oriented profile with low data complexity.

The **OWL 2 RL** profile is designed for systems that require scalable reasoning. Base reasoning services such as ontology consistency, class expression satisfiability, class expression subsumption can be solved in polynomial time w.r.t. the ontology size. Figure 2.3 shows the relation between the OWL2 language and its profiles. The difference between applications of the OWL profiles is due to their different entailment regimes (the subset of allowed axioms and expressions) which are summarized in Appendix A.

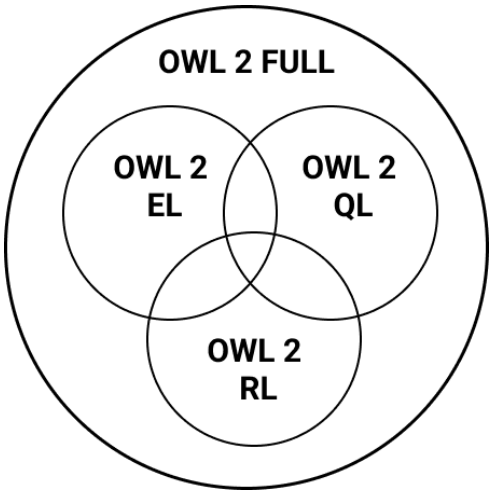


Figure 2.3: OWL 2 superset.

Term	Interpretation
owl:Class	Class definition.
owl:Thing	Top class (universe).
owl:NamedIndividual	Individual definition.
owl:Nothing	Bottom class (empty set).
owl:intersectionOf (C1 ... Cn)	Intersection of classes.
owl:unionOf (C1 ... Cn)	Union of classes.
owl:allValuesFrom C	Universal restriction.
owl:someValuesFrom C	Existential restriction.
owl:disjointWith C2	Class disjointness.
owl:equivalentClass C_{j+1}. $j = 1 \ldots n-1$	Class equivalence.
owl:AllDisjointClasses	Classes disjointness.
owl:Restriction	Restriction.
owl:someValuesFrom	Existential restriction.
owl:topObjectProperty	Universal property.
owl:ObjectProperty	Property definition.
owl:inverseOf Pn	Inverse property.
owl:SymmetricProperty	Symmetric property.
owl:TransitiveProperty	Transitive property.

Table 2.2: OWL2 main vocabulary.

2.2.4. SPARQL

SPARQL (SPARQL Protocol and RDF Query Language) is a graph query language standardized by the W3C used to retrieve and manipulate data stored in an RDF graph. SPARQL allows to extract information in the form of URIs, blank nodes, plain and typed literals, to extract RDF subgraphs or to construct new RDF graphs based on information

from a Knowledge Base. SPARQL queries are based on graph pattern matching, which consists of bindings between query variables in triple patterns and RDF terms. A triple pattern is similar to an RDF triple, but any component can be a query variable (indicated with "?"), as shown in Listing 2.6:

```
<?x ex:hasColor ?y>.
```

Listing 2.6: triple pattern in SPARQL.

As shown in Listing 2.6, SPARQL uses the Turtle syntax and allows Base URIs and prefixes definition. The result of a query that extracts information from an RDF dataset is the collection of triples that matches the desired triple pattern. In the case of queries that read data from the database, the SPARQL language specifies four different query variations for different purposes (i.e., SELECT, ASK, CONSTRUCT, and DESCRIBE). Each of these query forms optionally takes a WHERE block to restrict the query and may also include modifiers such as DISTINCT, ORDER BY and GROUP BY. For the purpose of this Thesis, the SELECT and the ASK are used, of which we present an example. The **SELECT query** extracts raw values from a dataset, the results are returned in a table format. Listing 2.7 shows a query that returns all the URI of entities having yellow color.

```
PREFIX ex: <http://ex.org/>
SELECT ?x
WHERE{
    ?x ex:hasColor ex:Yellow .
}
```

Listing 2.7: SELECT query example.

The output obtained applying the query on the example in 2.5 is reported in Listing 2.8.

```
<http://ex.org/Sun>
```

Listing 2.8: SELECT query result.

The **ASK query** provides a simple True/False result for a query. Listing 2.9 example shows a query that states if the sun is yellow. The output obtained applying the query on the example in 2.5 is "True".

```
PREFIX ex: <http://ex.org/>
ASK   { ex:Sun ex:hasColor ex:Yellow}
```

Listing 2.9: ASK query example.

2.3. Deep learning

Deep learning is a branch of machine learning based on artificial neural networks, models that consist of the aggregation of multiple perceptrons and can be universal classifiers thanks to nonpolynomial activation functions and hidden layers of unbounded width. The adjective "deep" refers to the use of multiple and deep layers in the networks. Deep learning is based on 3 main learning paradigms: Given a collection of data $\mathbf{D} = \mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, ..., \mathbf{x_N}$

- *Supervised learning*: given the desired outputs $\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}, ..., \mathbf{t_N}$ produces the correct output given a new set of input.
- *Unsupervised learning*: exploit regularities in $\mathbf{D}$ to build a representation to be used for reasoning or prediction.
- *Reinforcement learning*: producing actions $\mathbf{a_1}, \mathbf{a_2}, \mathbf{a_3}, ..., \mathbf{a_N}$ which affect the environment, and receiving rewards $\mathbf{r_1}, \mathbf{r_2}, \mathbf{r_3}, ..., \mathbf{r_N}$ learn to act in order to maximize rewards in the long term.

Deep learning mainly concerns about supervised learning, but deep models are widely used in combination with ML models in the other paradigms.

2.3.1. FFNN and CNN

Feedforward neural networks are non-linear models characterized by multiple layers of computational units (neurons) connected in a feed-forward way that are able, thanks to the universal approximation theorem, to approximate every continuous function that maps intervals of real numbers to some output interval of real numbers. Feedforward neural networks (FFNN) are the most general-purpose neural network. The entry point is the input layer, and it consists of several hidden layers and an output layer (see Figure 2.4). The output of a single neuron is given by an activation function (that must be differentiable) applied to the weighted sum of its input (output of the previous layer). FFNN follows the supervised learning paradigm, the training phase consists of adjusting the weight of the model in order to minimize the loss function, which choice relies on the

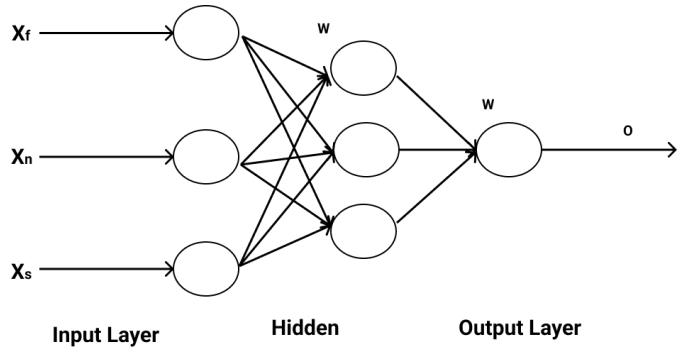


Figure 2.4: High level view of an FFNN's architecture.[6]

design task. It could be for example the mean squared error (MSE) for regressions or cross-entropy for classifications.

The optimization problem is solved by gradient-based algorithms such as stochastic gradient descent (SDG). Computing the gradient of the loss function with respect to the network's weights and adjusting them to the opposite direction of the gradient, the loss will gradually decrease until it converges to some local minima. The weights adjustment is performed iteratively during the training phase, and it's regularized by a learning rate that avoids the algorithm's divergence (if the loss 'jumps over' the minimum) and decreases the learning time. The gradient instead is obtained using backpropagation, which computes the gradient of the loss function with respect to each weight using the chain rule, that is, computing the gradient one layer at a time and iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule.

FFNNs have some drawbacks in dealing with complex tasks. While solving an image classification problem, for example, the first step is to convert a 2-dimensional image into a 1-dimensional vector prior to training the model, drastically increasing the number of trainable parameters and losing the spatial features of an image. A Large number of parameters could lead to vanishing or exploding gradient, problems associated with the backpropagation algorithm due to the multiplications in the chain rule, or more in general, increases the complexity of the models, making them prone to overfitting data (some techniques exist to avoid the overfitting, such as dropout or weight decay).

Convolutional neural networks (CNNs) take a different approach toward regularization: they take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their filters.

The layers of a CNN are, in fact, made of convolutional filters which extract the high-level features of the input tensor. The main advantage of the convolutional filters is the

parameter sharing: it relies on the assumption that if a patch feature is useful at some spatial position, then it should also be useful at other positions. The neurons in each filter have the same weights and bias, allowing the forward pass in each depth slice of the convolutional layer to be computable as a convolution of the neurons' weights with the input volume. The result of this convolution is an activation map, and the set of activation maps for each different filter are stacked together along the depth dimension to produce the output volume. Parameter sharing also contributes to the translation invariance of the CNN architecture.

The convolutional filters "slide" on the input tensor (or volume) to apply the convolution; the dimension of the output volume can be reduced or controlled using padding or stride (the "pass" of the filter on the input volume). The dimensionality reduction could also be made using pooling layers, decreasing the computational power required. They are also useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training the model.

CNNs are usually completed by adding a fully-Connected layer for learning non-linear combinations of the high-level features as represented by the output of the convolutional layers. Figure 2.5 shows an example of a CNN architecture with the fully-connected layers for classification.

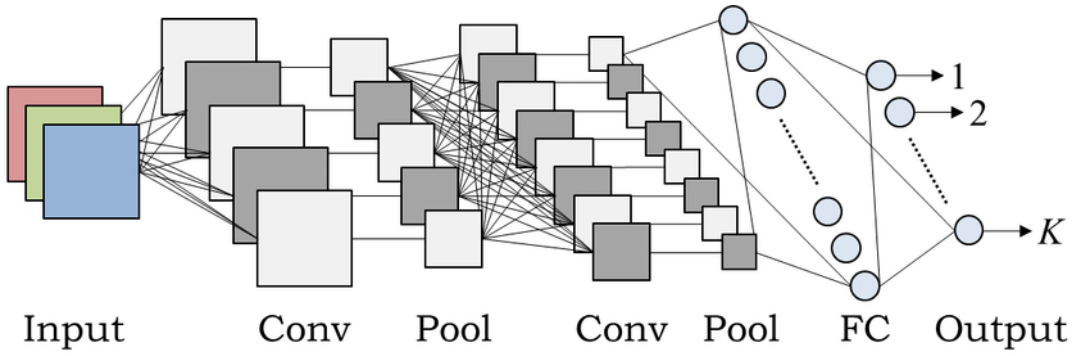


Figure 2.5: Example of a CNN's architecture.[5]

2.3.2. Autoencoder

Autoencoders are unsupervised artificial neural networks able to efficiently learn how to compress and encode data and how to reconstruct them back from the reduced encoded representation to a representation that is as close to the original input as possible. Autoencoders learn a representation (encoding) typically for dimensionality reduction by training the network to ignore the noise in the data. Autoencoders consists of 4 main parts:

- **Encoder**: The part of the model that learns how to reduce the input dimensions compressing the input data into an encoded representation.
- **Bottleneck**: The layer that contains the compressed representation of the input data.
- **Decoder**: The part of the model that learns how to reconstruct the data from the encoded representation.
- **Reconstruction Loss**: The function that measures how well the decoder is performing.

The training then involves backpropagation in order to minimize the network's reconstruction loss. Figure 2.6 shows a high-level view of the architecture of an autoencoder.

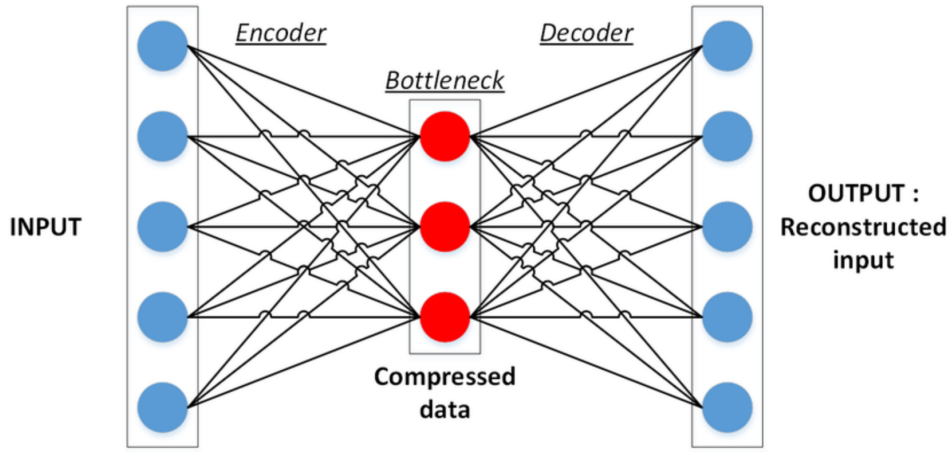


Figure 2.6: High level view of an autoencoder's architecture [22].

The network architecture could be a simple FeedForward network or Convolutional Neural Network depending on the use case, some of which are image denoising, anomaly detection, and word embedding.

2.3.3. Variational Autoencoder

Aiming to force the learned representations to assume useful properties, autoencoders can become generative models, taking the name of Variational Autoencoders (VAEs [9]). Just as standard autoencoders, variational autoencoders are architectures made of both an encoder and a decoder and are trained to minimize the reconstruction error between the encoded-decoded data and the input data. However, in order to introduce some regularisation of the latent space, a slight modification of the encoding-decoding process

is needed: instead of as a single point, a distribution over the latent space is encoded and the model is then trained as follows:

1. The input is encoded as a distribution over the latent space;
2. A point from the latent space is sampled from that distribution;
3. The sampled point is decoded;
4. The reconstruction error can be computed and backpropagated through the network.

Encoded the normal distribution, the encoder can be trained to return their means and covariance matrices, allowing to express the latent space regularization very naturally: the returned distributions are enforced to be close to a standard normal distribution. Thus, the loss function is composed of a "reconstruction term" (on the final layer) and a "regularization term" (on the latent layer) that tends to regularize the organization of the latent space by making the distributions returned by the encoder close to a standard normal distribution. As reconstruction loss, mean squared error and crossentropy are both often used, as distance loss between the two distributions instead, the reverse Kullback–Leibler divergence between the real posterior distribution and the encoded one is a good choice. However, without a well-defined regularisation term, the model can learn, in order to minimize its reconstruction error, to "ignore" the fact that distributions are returned and behave almost like classic autoencoders (leading to overfitting), either returning distributions with tiny variances or with very different means. In order to avoid these effects, the regularization is done by enforcing distributions to be close to a standard normal distribution (centered and reduced), requiring the covariance matrices to be close to the identity, preventing punctual distributions, and the mean to be close to 0, preventing encoded distributions from being too far apart from each other. Naturally, as for any regularisation term, this comes at the price of a higher reconstruction error on the training data. The trade-off between the reconstruction error and the KL divergence can however be adjusted using a variational inference formulation.

Being $\mathbf{z}$ a latent representation sampled from the prior distribution $p(\mathbf{z})$, the input data x is sampled from the conditional likelihood distribution $p(\mathbf{x}|\mathbf{z})$. Thus, the encoder is defined as $p(\mathbf{z}|\mathbf{x})$ while the decoder as $p(\mathbf{x}|\mathbf{z})$. The regularisation of the latent space appears in the definition of the data generation process: encoded representations $\mathbf{z}$ are assumed to follow the prior distribution $p(\mathbf{z})$. The link between the prior $p(\mathbf{z})$, the likelihood $p(\mathbf{x}|\mathbf{z})$, and the posterior $p(\mathbf{z}|\mathbf{x})$ is also enlightened by the Bayes theorem:

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{\int p(\mathbf{x}|\mathbf{u})p(\mathbf{u})\,du}$$

With the assumption that $p(\mathbf{z})$ is a standard Gaussian distribution and $p(\mathbf{x}|\mathbf{z})$ is a Gaussian distribution whose mean is defined by a deterministic function f of the variable $\mathbf{z}$ (with co-variance matrix $c\mathcal{I}$ where c is a constant), the function f is assumed to belong to a family of functions denoted F, having:

$$p(\mathbf{z}) \equiv \mathcal{N}(0, \mathcal{I})$$

$$p(\mathbf{x}|\mathbf{z}) \equiv \mathcal{N}(f(\mathbf{z}), c\mathcal{I}), \quad f \in F, \quad c > 0$$

Computing $p(\mathbf{z}|\mathbf{x})$ with the Bayesian theorem implies to solve the integral at the denominator which is often intractable and so requires the use of approximation techniques. Variational inference (VI) is a technique to approximate complex distributions, consists of setting a parametrized family of distribution and looking for the best approximation of the target distribution among this family, that is, the one that minimize a given approximation error measurement (Kullback-Leibler divergence between approximation and target) and is found by gradient descent over the parameters that describe the family. Let $q_x(\mathbf{z})$ be a Gaussian distribution whose mean and covariance are defined by two parametrized functions, g and h, of the parameter x belonging to the families of functions G and H:

$$q_x(\mathbf{z}) \equiv \mathcal{N}(g(\mathbf{x}), a(\mathbf{x})), \quad g \in G, \quad h \in H$$

$q_x(\mathbf{z})$ is a family of candidates for variational inference to approximate $p(\mathbf{z}|\mathbf{x})$. The best approximation among this family is found by optimizing g and h that minimize the Kullback-Leibler divergence between the approximation and the target $p(\mathbf{z}|\mathbf{x})$:

$$(g^*, h^*) = \underset{(g,h)\in G\times H}{\arg\min}\, KL\left(q_x(z), p(z \mid x)\right)$$

$$= \underset{(g,h)\in G\times H}{\arg\min} \left(\mathbb{E}_{z\sim q_x}\left(\log q_x(z)\right) - \mathbb{E}_{z\sim q_x}\left(\log \frac{p(x \mid z)p(z)}{p(x)}\right)\right)$$

$$= \underset{(g,h)\in G\times H}{\arg\min} \left(\mathbb{E}_{z\sim q_x}\left(\log q_x(z)\right) - \mathbb{E}_{z\sim q_x}(\log p(z)) - \mathbb{E}_{z\sim q_x}(\log p(x \mid z)) + \mathbb{E}_{z\sim q_x}(\log p(x))\right)$$

$$= \underset{(g,h)\in G\times H}{\arg\max} \left(\mathbb{E}_{z\sim q_x}(\log p(x \mid z)) - KL\left(q_x(z), p(z)\right)\right)$$

$$= \underset{(g,h)\in G\times H}{\arg\max} \left(\mathbb{E}_{z\sim q_x}\left(-\frac{\|x - f(z)\|^2}{2c}\right) - KL\left(q_x(z), p(z)\right)\right)$$

The trade-off there exists between maximizing the likelihood of the "observations" and staying close to the prior distribution comes up in the second last equation and expresses the balance that needs to be found between the confidence in the data and the confidence

in the prior. If the regularity is mostly ruled by the prior distribution assumed over the latent space, the performance of the overall encoding-decoding scheme highly depends on the choice of the function f, which defines the decoder. Indeed, as $p(\mathbf{z}|\mathbf{x})$ can be approximate (by variational inference) from p(z) and $p(\mathbf{x}|\mathbf{z})$ and as $p(\mathbf{z})$ is a simple standard Gaussian, the only two optimizable levers in the model are the parameter c (that defines the variance of the likelihood) and the function f (that defines the mean of the likelihood).

For any function f in F (each defining a different probabilistic decoder $p(\mathbf{x}|\mathbf{z})$) the best approximation of $p(\mathbf{z}|\mathbf{x})$ can be found, denoted $q_x^*(\mathbf{z})$, in order to get an encoding-decoding scheme as efficient as possible and, f needs to maximize the expected log-likelihood of $\mathbf{x}$ given $\mathbf{z}$ when $\mathbf{z}$ is sampled from $q_x^*(\mathbf{z})$. Thus, the optimal f^* is such that

$$\begin{aligned} f^* &= \arg\max_{f \in F} \mathbb{E}_{z \sim q_x^*}(\log p(x \mid z)) \\ &= \arg\max_{f \in F} \mathbb{E}_{z \sim q_x^*}\left(-\frac{\|x - f(z)\|^2}{2c}\right) \end{aligned}$$

where $q_x^*(\mathbf{z})$ depends on the function f. Gathering all the pieces together, the goals are f^*, g^* and h^* such that

$$(f^*, g^*, h^*) = \arg\max_{(f,g,h) \in FxGxH} \left(\mathbb{E}_{z \sim q_x}\left(-\frac{\|x - f(z)\|^2}{2c}\right) - KL\left(q_x(\mathbf{z}), p(\mathbf{z})\right)\right) \tag{2.1}$$

This objective function contains the reconstruction error between $\mathbf{x}$ and $f(\mathbf{z})$ and the regularization term given by the KL divergence between $q_x(\mathbf{z})$ and $p(\mathbf{z})$ (which is a standard Gaussian). The constant c rules the balance between the two previous terms. A higher c means a high variance around $f(\mathbf{z})$ for the probabilistic decoder and, so, the regularisation takes over the reconstruction term (and the opposite stands for lower c).

Since it's not easy to optimize over the entire space of functions, the optimization domain could be constrained and f, g and h expressed as neural networks. Thus, F, G and H correspond respectively to the families of functions defined by the networks architectures and the optimisation is done over the parameters of the networks. In practice, g and h are not defined by two completely independent networks but share a part of their architecture and their weights as Figure 2.7 shows:

$$g(\mathbf{x}) = g_2(g_1(\mathbf{x})) \quad h(\mathbf{x}) = h_2(h_1(\mathbf{x})) \quad g_1(\mathbf{x}) = h_1(\mathbf{x})$$

In order to reduce the number of parameters, $q_x(\mathbf{z})$ is assumed to be a multidimensional Gaussian distribution with a diagonal covariance matrix, so $h(\mathbf{x})$ is the vector of its

diagonal elements and has the same size as $g(\mathbf{x})$ (these assumptions could result in less accuracy). Contrarily $p(\mathbf{x}|\mathbf{z})$ is assumed to be a Gaussian with fixed covariance. The function f of the variable $\mathbf{z}$ defining the mean of that Gaussian is modeled by a neural network and can be represented as in Figure 2.8.

The overall architecture is then obtained by concatenating the encoder and the decoder parts. The sampling process has to be expressed in a way that allows the error to be backpropagated through the network. A simple trick, called *reparametrization trick*, is used to make the gradient descent possible despite the random sampling that occurs halfway through the architecture and consists in using the fact that if $\mathbf{z}$ is a random variable following a Gaussian distribution with mean $g(\mathbf{x})$ and with covariance $H(\mathbf{x}) = h(\mathbf{x}).h^T(\mathbf{x})$ then it can be expressed as

$$\mathbf{z} = h(\mathbf{x})\delta + g(\mathbf{x}) \quad \delta \sim \mathcal{N}(0, \mathcal{I})$$

Finally, the objective function is given by Equation 2.1 in which the theoretical expectancy is replaced by a more or less accurate Monte-Carlo approximation that consists, most of the time, into a single draw. Considering this approximation and denoting $C = \frac{1}{2c}$, the obtained loss is a function composed of a reconstruction term, a regularisation term and a constant to define the relative weights of these two terms:

$$loss = C\|x - f(z)\|^2 - KL\left(\mathcal{N}(g(\mathbf{x}), h(\mathbf{x})), \mathcal{N}(0, \mathcal{I})\right) \tag{2.2}$$

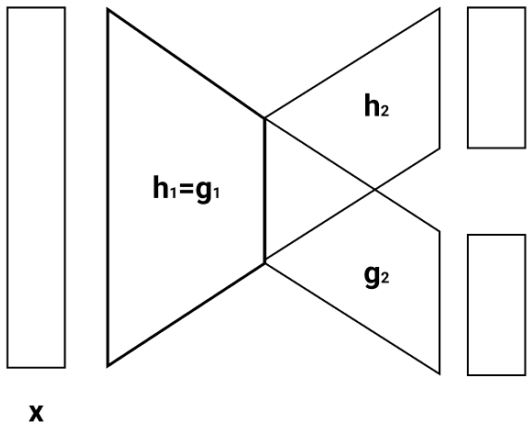


Figure 2.7: VAE encoder representation.

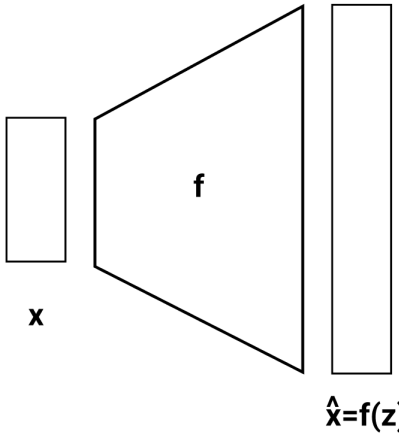


Figure 2.8: VAE decoder representation.

3 | Problem statement

This chapter presents our research problem, its context, and the research questions that guided our investigations. For the problem formalization we used the Macro-Meso-Micro framework [19] that allows to formulate research questions at three levels of analysis.

1. The **Macro** level is generally related to broad, complex, and unanswerable questions. It refers to large-scale analysis related to entire communities or families of entities.
2. The **Meso** level presents more specific but still unanswerable questions. It refers to medium-scale analysis involving small groups or a Ph.D. thesis.
3. The **Micro** level is where the actual investigation happens. It follows a list of feasible problems to solve.

3.1. From reasoning to approximate reasoning

Knowledge representation (KR) is the branch of artificial intelligence dedicated to representing information about the world in a form that a computer system can use to solve complex tasks. KR also incorporates findings from logics to automate reasoning, such as the application of rules or the relations of sets and subsets. A representation could be more or less expressive, depending on the semantics it can encode. Figure 3.1 shows how the increase in the semantics of a representation results in an increase in the reasoning capabilities. For example, with XML Schema we can only describe the elements in an XML document, while with RDF/RDFS we can define object classes and binary relationships.

The most expressive representation in Figure 3.1 is the Web Ontology Language 2 (OWL), which is the standard used in Semantic Web to represent rich and complex knowledge. In particular, information and the semantics of a given domain are encoded in Knowledge Bases (KB). KBs include sets of axioms defining classes and properties (TBox), and sets of individuals (ABox) that represent instances of the given domain (see Section 2.2.3 for more details). Thanks to automated reasoning, computer programs are able to produce new information by deducing facts of individuals in the ABox using the

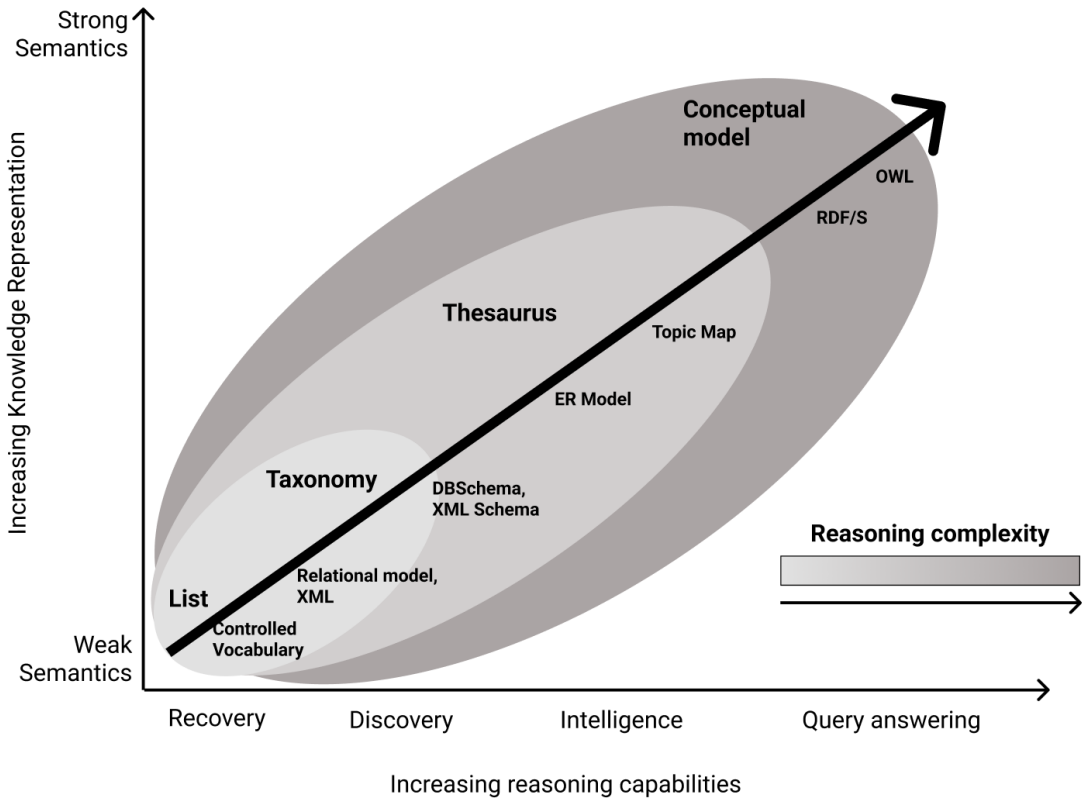


Figure 3.1: Relation between reasoning capabilities and knowledge representation.

axioms in the TBox and given inference rules. In particular, three reasoning tasks are allowed with OWL:

- **Consistency checking**: the task of determining whether an ontology is consistent or not.
- **Classification**: the task of computing all entailed class subsumption between named classes in the ontology.
- **Materialization**: the task of computing all entailed class assertions for named classes and individual names occurring in the ontology. Materialization can be divided in 3 different sub-tasks :
  - *Instance retrieval*: the task of listing all the individuals of a given class.
  - *Concept satisfiability*: the task of deciding if a given concept is consistent with respect to the KB.
  - *Axiom entailment*: the task of verifying if a given axiom is entailed by the KB.

Profile	Consistency checking complexity	DL
OWL2 EL	PTIME-complete	$\mathcal{EL}++$
OWL2 QL	NLogSPace-complete	$\mathcal{DL}-Lite$
OWL2 RL	PTIME-complete	$\mathcal{DLP}$
OWL2 FULL	Undecidable	$\mathcal{SROIQ}(\mathcal{D})$

Table 3.1: OWL2 profiles complexity of the consistency checking task and their DLs.

The formalisms underlying OWL are the Description Logics (DLs). DLs are logical languages designed for the specification of Knowledge Bases. They extend both Propositional Logic (PL) and First Order Logic (FOL) but also allow for the specification of binary relationships. Some DLs are more expressive than others. In general, the more expressive a DL is, the more complex reasoning becomes. Therefore, a balance between expressiveness and the complexity of reasoning has to be sought. OWL 2 is based on $\mathcal{SROIQ}(\mathcal{D})$, the most expressive decidable fragment of FOL. However, OWL 2 extends $\mathcal{SROIQ}(\mathcal{D})$ with axioms that makes the reasoning undecidable. For this reason, three fragments (profiles) of OWL are defined, based on more simple DLs, whose reasoning complexities are bounded. Table 3.1 shows the complexity of each OWL profile for the consistency checking task and their underlying DLs. For what concerns the other reasoning tasks instead, it is hard to evaluate their complexity because it depends on the algorithms used by the reasoners (we only know the upper and lower bounds). Nevertheless, Singh et Al. [20] have designed OWL2Bench, a benchmark that empirically compares standard reasoners for all the three reasoning tasks on the OWL profiles and clearly show how the time complexity increases as the expressiveness of the ontology increases. As an example, Table 3.2 shows the performances of HermiT on an ABox belonging to OWL2Bench, made of 5 universities. The reasoner runs out of time (t.o.) on all the tasks for the RL profile and on the classification and realization task for the EL profile.

Profile	CC	CT	RT
OWL2 EL	665.12	t.o.	t.o.
OWL2 QL	6.56	10.01	10.01
OWL2 RL	t.o.	t.o.	t.o.

Table 3.2: Hermit performances (in seconds) on consinstency cheking (CC), classification (CT) and realization (RT) tasks of OWL2Bench.

The complexity of the algorithms (in particular time complexity) increases as the expressiveness of a DL increases. The OWL profiles focus on the trade-off between complexity and expressiveness in an exact fashion, defining segments of the language in which the limited expressiveness has the maximum performance. Another approach instead con-

sists of the approximate reasoning, a technique that improves the efficiency of deduction by trading correctness for performance. Entering the world of approximation, the first assumption is the loss of information. Thus, the attention is brought towards the quality of the inference produced. Reasoning produces inference according to a so-called entailment regime. However, not all deductions are equally important. For example, given :

$$Amedeo \xrightarrow{type} Student \xrightarrow{subClassOf} Person \xrightarrow{subClassOf} Thing$$

knowing that "Amedeo is a Thing" does not add much information as knowing that "Amedeo is a Person".

OWL reasoning is done in a deductive way, so it needs valid premises to derive valid conclusions. The approximation process corrupts the deductive approach, removing for example some premises. For this reason, our first approach was taking into consideration inductive reasoning as an approximation medium. From that, it follows the first research question.

M Is it possible to approximate deductive reasoning using inductive reasoning?

The Macro **M** covers numerous possible alternatives, therefore, it is hard to answer. However, we can use the question to formulate some hypotheses that will guide our investigation.

H1 The approximated reasoner should be faster than a standard reasoner.

H2 The approximate reasoner should reach a sufficient soundness and completeness threshold for the materialization task.

H3 The loss of soundness and completeness doesn't have to be horizontal.

H1 is the result, in terms of computing time, which we are expecting doing an approximation, while *H2* is the result in terms of the quality of the approximation. *H3* specializes *H2*, in particular, we expect that the approximation compresses useless information in favor of the more meaningful ones.

3.2. From Macro to Meso

Guided by **M**, we have analyzed the inductive approaches adopted by the research community. In particular, we have begun with the survey by Nickel et al. [16], which lists the state-of-the-art approaches based on ML and the first approaches using deep learning. The first relevant approach is RESCAL [15] which is an embedding model that uses tensor

factorization to predict the probability of the existence of a triple. Instead, NTN [21] is the first approach that combines tensor factorization with standard neural network layers, resulting in a model that scores triples to predict their existence.

According to Nickel et al. [16] we can distinguish another family of ML methods that consists of translational embeddings, namely, TransE [1], TransH [28], TransD [7] and TransR [11]. These models predict the probability of relationships from the distance between latent representations of entities embedded in hyperspaces.

Two modern approaches based on CNN have been proposed: ConvE [3] and ConvKB [14]; the former uses a convolutional layer to complete a given couple <s,p> with the missing <o>, the latter scores the triples in input using convolutional filters and the dot product. The use of deep learning has gone even forward with more complex models such as DAPath [23] and GRL [27] which uses reinforcement learning in combination with path embedding or LSTM to perform triple completion. Nathani et Al. [13] proposed to use a graph attention network (GAT [24]) together with a multi-head attention mechanism to embed and score a set of given triples. Finally, with a different perspective, Paulheim et Al. [17] proposed to perform approximate ABox consistency checking by using a linear classifier, reaching an accuracy of more than 95% at computation times at least 50 times as fast as a standard ontology reasoner.

Table 3.3 summarizes the approaches and highlights some important characteristics. In particular,

- "type of inference" indicates the type of feature used by the model (features extracted by embeddings, attention for graphs, and paths);
- "semantic" stands for the usage of the ontology in building the model (explicitly means that the model was directly fed with the ontology while implicitly means that the model has learned some ontology's rules from data);
- "reasoning task" refers to the reasoning task performed by the model.

The "reasoning task" in Table 3.3 has no direct correspondence with the OWL tasks. On the contrary, it can be seen as an interpretation of the models' inference from an OWL point of view. For example, RESCAL performs triples prediction, which is similar to what a reasoner does with axiom entailment. Models that perform triples completion can be related to the instance retrieval task. The approach of Paulheim et Al. [17], instead, is the only one that explicitly aims to approximate the consistency checking task. Since Nickel et al. [16] did not report any reference to the reasoning tasks, we labeled the models as follows:

- **Axiom entailment**: if the model scores the triples or computes existence probabilities.
- **Instance retrieval**: if the model performs triple completion.
- **KB satisfiability**: if the model computes the satisfiability of the KB.

Model	**Technique**	**Type of inference**	**Semantic**	**Reasoning task**
RESCAL	Embedding via tensor factorization	Feature manipulation	Implicit	axiom entailment
NTN	Tensor factorization+NN	Feature manipulation	Implicit	axiom entailment
Trans_	Translational embedding	Feature manipulation	Implicit	Instance retrieval
GAT	Attention mechanism	Attention manipulation	Explicit	axiom entailment
ConvE	CNN	Feature manipulation	Implicit	instance retrieval
ConvKB	CNN	Feature manipulation	Implicit	axiom entailment
DAPath	RL + path embedding	Feature manipulation	Implicit	axiom entailment
GRL	LSTM + RL	Path manipulation	Implicit	Instance retrieval
Abox CC	Linear classifier	Feature manipulation	Explicit	KB satisfiability

Table 3.3: State of the art models summary.

From Table 3.3 emerges that none of the presented approaches approximates the materialization task, but rather completes single triple iteratively or "brute-forces" all the possible combinations of <s,p,o>. The task of designing a model for approximate materialization poses the challenge of designing a model that "generates" new triples using only the information given as input.

The generative nature of the task has led us towards generative models. In particular, Variational Autoencoders (VAE) consist of an encoding block that compresses the input

information into a constrained multivariate latent distribution and a decoding block that reconstructs it as accurately as possible. VAEs were initially designed for unsupervised learning, but their effectiveness has been proven for semi-supervised or supervised learning forcing the learned representations to assume useful properties. More details are listed in Section 2.3.3. Following the intuition of VAE we reduce question **M** into:

> **m** Is it possible to approximate the materialization task with a variational autoencoder?

3.3. From Meso to Micro

VAEs belong to the family of deep learning models, i.e., they are universal approximations for any given function. Thus, we are going to introduce the following formalization:

Let $\mathcal{MT}$ be the function that represents the materialization:

$$\mathcal{MT} : \{ABox + TBox\} \rightarrow ABox^* \tag{3.1}$$

$\mathcal{MT}$ takes as input the KB (ABox+TBox) and outputs $ABox^*$, the materialized Abox. Let $approx - \mathcal{MT}$ be the function that approximates $\mathcal{MT}$:

$$approx - \mathcal{MT} : \{ABox + TBox\} \tilde{\rightarrow} ABox^* \tag{3.2}$$

With this formalization, the meso **m** translates into designing a variational autoencoder that approximates $\mathcal{MT}$ with the function $approx - \mathcal{MT}$. So, we can elicit some requirements to guide our design.

- **R1**: The semantic (Tbox) shall be taken into account in the vector representation.
- **R2**: The model shall take as input the entire ABox.
- **R3**: A unique mapping function between the OWL representation and the vector representation used as input and output of the model shall exist.

R1 comes directly from the challenge of the Macro **M**, since we aim to approximate the function $\mathcal{MT}$ which uses the rules of the ontology to make inferences on the ABox. **R2** comes from the definition of the function $approx - \mathcal{MT}$ and guarantees a fair comparison with standard reasoners (their input are the complete ABoxes). Moreover, processing the entire ABox could increase the accuracy of the inference. For example, symmetric properties involve two triples (i.e., <a,P,b> and <b,P,a>, where P is the symmetric property). The symmetry could be lost when taking as input a single triples iteratively. The major-

ity of the models presented in Table 3.3 use unsupervised paradigms to build embeddings for the triples and use maximum likelihood probability to decode the embedded values. Thus, **R3** is essential to avoid ambiguities, especially during the evaluation of the model asserted triples. The requirements focus on the choice of an embedding model for the KB. In particular, **R3** imposes the use of Closed World Assumption (CWA), i.e., every fact that is not known to be true is considered false (it is opposite to the open world assumption where the answer of an unknown fact is "not known"). In our context, the CWA is essential to prevent the explosion of the triple generation for example, in the case of blank nodes. After this meditation, we can enlight the challenges that comes with **m**:

1. Find an embedding model that encodes the semantic of the TBox and supports **R1** and **R2**.
2. Find an architecture for the variational autoencoder able to approximate the function $\mathcal{MT}$ respecting **R2** and **R3**.

Now that we have chosen the model and set the requirements for the embedding, we must choose the experimental setting to prove the feasibility of our research question more. However, in the literature, there is no benchmark for the approximate materialization task that investigates the trade-off between expressiveness and efficiency. In particular, a meaningful benchmark for the task should include strong semantics to stress the reasoner and should follow the standards for what concerns Knowledge Representation (i.e., KG with OWL profiles) and Deep Learning (i.e., large and scalable dataset). The lack of a standardized datasets for evaluating approximated materialization represents a huge obstacle for our research. Thus, we decided to focus on the design of a reliable and flexible benchmark for the task. Therefore, our research micro question is:

> μ How can we design a benchmark to evaluate the task of approximate materialization?

A standard benchmark design workflow consists of three main steps:

1. Describe the purpose of the benchmark.
2. Select or design a representative dataset.
3. Design a baseline.

The scope of the benchmark has already been introduced: we aim to evaluate the performance of deep learning models on the approximate materialization task. OWL2Bench[20] is a benchmark that evaluates the performance of standard reasoners on different expressive KGs. However, it does not represent a feasible solution for evaluating the efficiency of

the approximate materialization because it does not provide the ground truth for the task. The reasoners tested by OWL2Bench, in fact, run out of time for most of the dimension of the proposed KG.

For what concerns the dataset selection, two approaches are feasible: Using real-world data or designing a synthetic dataset. The first approach consists of using well-known ontology such as GALEN[1]. GALEN is an ontology with a strong semantic, but in our context comes with two problems: it has a small ABox with relatively few entities, which would need fragmentation to produce many samples for the deep learning model. On the other hand, DBpegia [2], Yago [3] and SNOMED [4] are examples of knowledge bases with huge ABoxs but not very expressive Tboxes. Instead, the second approach has the advantage of having both the desired expressiveness and a possibly infinite number of samples in the ABox. Moreover, real data contain irregularities that could stress the reasoner. A synthetic dataset could be seen as more regular and allows the introduction of additional noise for deeper investigations. So, given the desired level of expressiveness, the most important feature for training our deep learning model is scalability, which brings us towards the synthetic approach. Instead of designing a synthetic dataset from scratch, we have chosen to bring CLEVR [8] into the logic domain, a well-accepted benchmark for the visual question-answering task in Deep Learning [12]. The choice of CLEVR relies on having already identified abstractions which simplifies the KGs design, and a large synthetic dataset for training a deep learning model that could be extended or corrupted, and also includes questions that can be tranlsated into SPARQL queries.

The goal of a benchmark is to push the technological progress by guaranteeing a fair assessment. Therefore, it is crucial to test the designed task and dataset against a baseline that is sufficiently sophisticated to solve the task yet not more advanced than the state of the art. For our purpose, the baseline is a trivial model that shows the quality of the benchmark for the approximate materialization task and sets a starting point for future investigations. Therefore, we opted for a Variational Autoencoder, which satisfies **R2** and lays the foundation for future investigation about our meso research question **m**. For what concerns the KG embedding, the simpler model that could respect our requirements is the SPO, a tensor representation for triples in a KB used to build RESCAL [15], where an entry $\chi_{i,j,k} = 1$ if the triple $<s_i,p_j,o_k>$ exists. With SPO, we can encode every triple of the KBs in a single tensor, so the model uses the full available information for its predictions and satisfies **R1**. Moreover, SPO guarantees a unique mapping between the

[1]https://www.opengalen.org/themodel/ontology.html
[2]https://www.dbpedia.org
[3]https://yago-knowledge.org
[4]https://www.snomed.org

KGs and the vector representation (**R3**), being the indexes of its entries unique for every triple.

This lead us to the definition of our **design problem**.

> Design a benchmark based on CLEVR to evaluate the approximation of the materialization task and test it on a baseline model that satisfies the requirements.

The solution to this problem is a reusable and adaptable benchmark that lays the foundation for future works and insights.

4 | Benchmark Design

This chapter presents the solutions to our design problem. In particular, Section 4.1 describes the modeling of the benchmark and the design its Knowledge Graph *KlevR*. Section 4.2 instead, focuses on the design of the baseline to test the benchmark. In particular presents *DeepR* and on the choice of the embedding model for the KGs.

4.1. Benchmark Task: KlevR

This Section describes the knowledge representation process that we followed, consisting of the modeling of *KlevR*. In order to design the ontology, we have followed two principles:

1. The level of expressiveness has to be such that the reasoner is heavily stressed.
2. We have to follow the knowledge representation standards.

The first one is the basement of our research since we wanted to design a benchmark for the materialization task. The second principle consists of designing an ontology with all the OWL profiles. In particular, given a domain, a standard ontologies design workflow consists of the following steps:

1. Analyse the domain;
2. Identify the abstractions;
   (a) Identify the entities.
   (b) Identify the properties that link the entities.
   (c) Organize the entities in a hierarchy.
   (d) Specialize the entities with properties.
3. Test the information needs (using competency questions);
4. Populate the knowledge base;
5. Materialize with a reasoner;

The result of our design is **KlevR**, an Knowledge Graph written in OWL that models CLEVR.

4.1.1. Domain Analysis

CLEVR is a benchmark that consists of scenes and questions. It is a diagnostic dataset for analyzing visual question answering (VQA) systems on different visual reasoning tasks [8]. It includes 70000 images for the training set and 15000 for the test set.

A CLEVR **scene** is a collection of spatially related objects on the ground-plane. Each **scene** is associated with a scene graph. A **scene graph** is direct graph with a set of nodes that represents objects and a set of edges that represents relationships among the objects. The nodes are labeled with the attributes of the object they represent. Scene graphs **are agnostic from the point of view**. Agnosticism is essential to have a standard and invariant description of each scene. On the other hand, a scene includes a point of view, which is randomly changed on the scenes. Figure 4.1 shows an example of a scene with three objects, a small brown cube in the top right, a small gray cylinder in the center-left and a small brown cube in the bottom center.

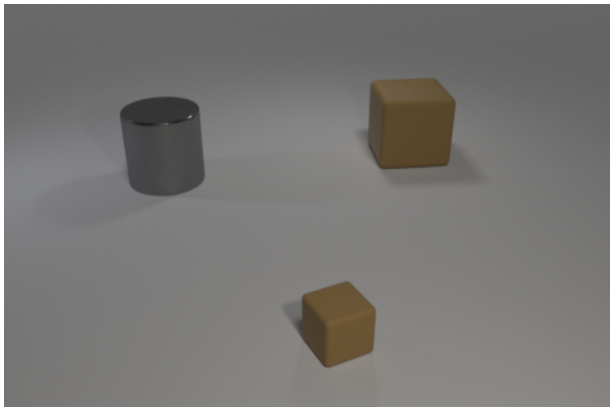


Figure 4.1: Example image of a CLEVR scene [8].

4.1.2. Abstractions

The first approximation we have made with respect to the VQA task is to use the scene graphs, which encode the ground truth information, removing the ambiguities due to the camera angle and point of vision.

The CLEVR entities include objects that can have the following atomic attributes: *{Shape, Size, Material, Color}*. Table 4.1 shows the possible values for each attribute, Figure 4.2 instead shows eight objects with different combinations of attributes.

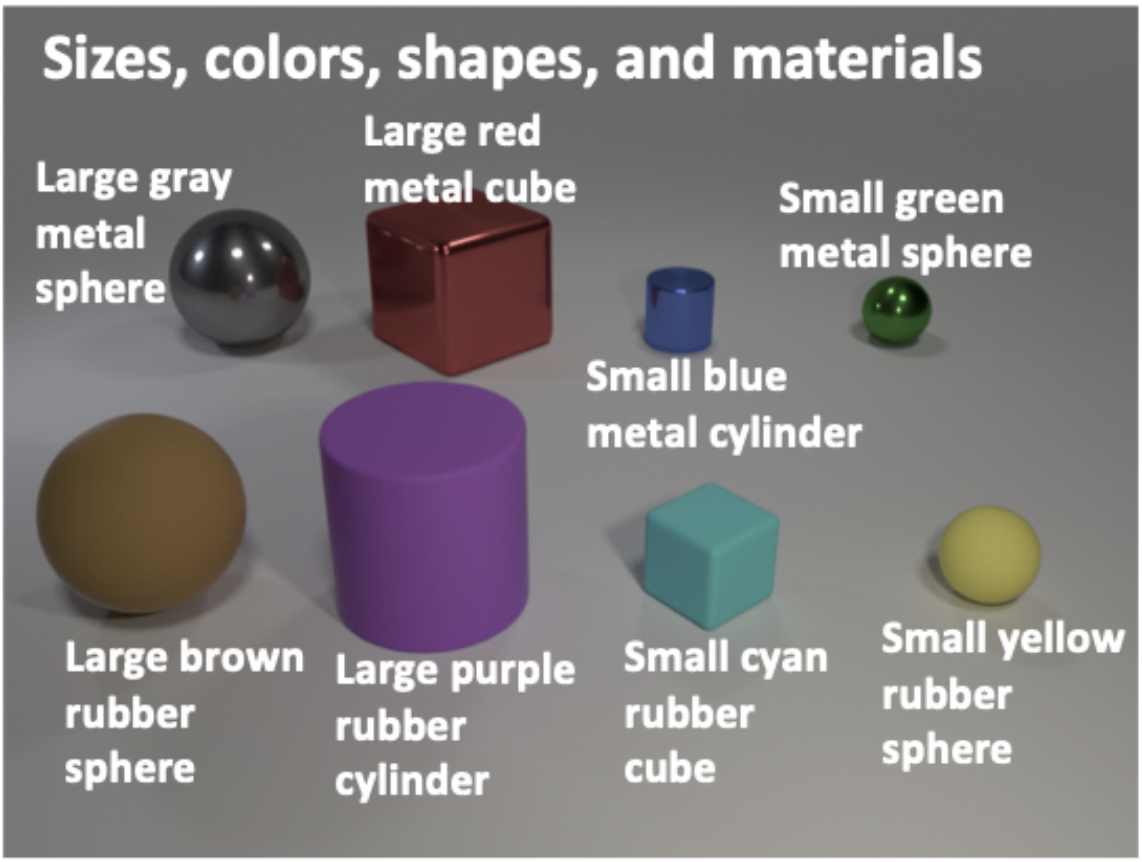


Figure 4.2: CLEVR objects with possible combinations of attributes [8].

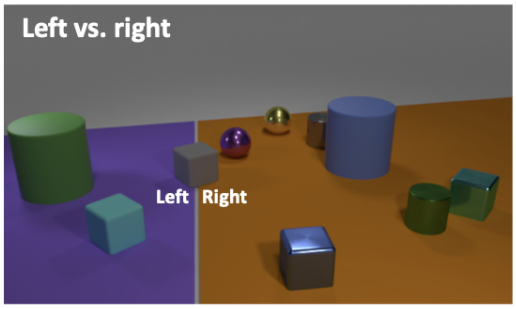


Figure 4.3: Left/Right relationship [8]

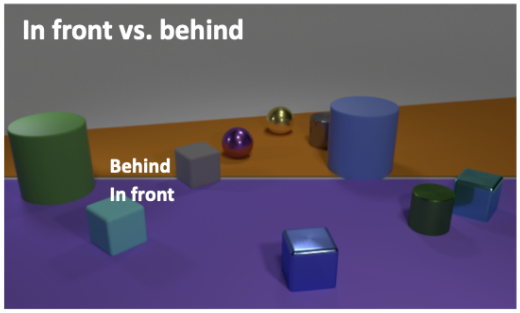


Figure 4.4: Front/Behind relationship [8]

Objects are spatially related via the the relationships: "left","right","behind","front". The semantics of the relationships depends on both object positions and camera viewpoint. Figures 4.3 and 4.4 show examples of how the spatial relationships are intended.

In *KlevR*, each object belongs to the class *object*, while a complex hierarchy of classes captures the semantics of the attributes. Conveying the combinatorial nature of CLEVR attributes, we have organized the numerous classes in two nested hierarchies.

Attribute name	Domain
Shape	Cube, Sphere, Cylinder
Size	Big, Small
Material	Rubber, Metal
Color	Blue, Brown, Gray, Red, Yellow, Green, Purple, Cyan

Table 4.1: CLEVR object's attributes.

Figure 4.5 shows the first hierarchy, which describes every possible combination of attribute types. In particular, the *level 1* in Figure 4.5 is made of the classes that represent a type of CLEVR attribute, namely *ColoredObject*, *MaterialObject*, *ShapedObject* and *SizedObject*. Their children instead are the intersection of the father with another attribute type. All the possible combinations of attributes at the conceptual level are shown in Figure 4.6. The standard we have followed in building the combinations consists of the alphabetical order of the attributes. A possible extension considering a different order is presented in Section 6.2.1.

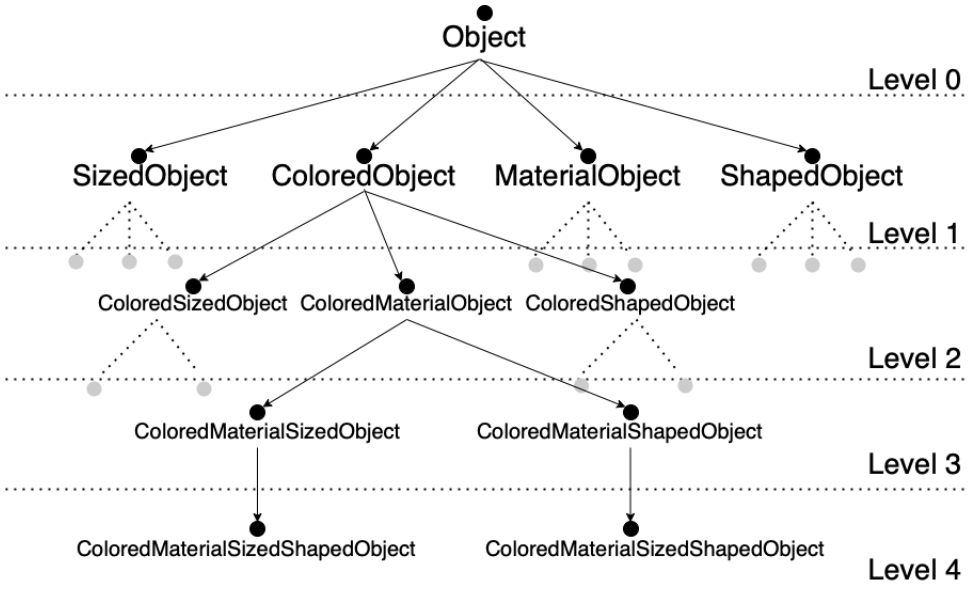


Figure 4.5: First KlevR class hierarchy.

The second hierarchy consists of attribute values. In particular, *KlevR* contains a class for each attribute value and for each possible combination, as shown in Figure 4.7. Considering the values reported in Table 4.1, the total number of classes is 323. For example, the attribute *Blue* is encoded with the class *BlueObject*, *Cube* with *CubeObject*. Combinations are made with both two values (e.g. *RedSphereObject*), three values (e.g. *GrayMetallicBigObject*) and four values (e.g. *GreenRubberSmallSphereObject*). The classes that describe an attribute value belong to both hierarchies. For example,

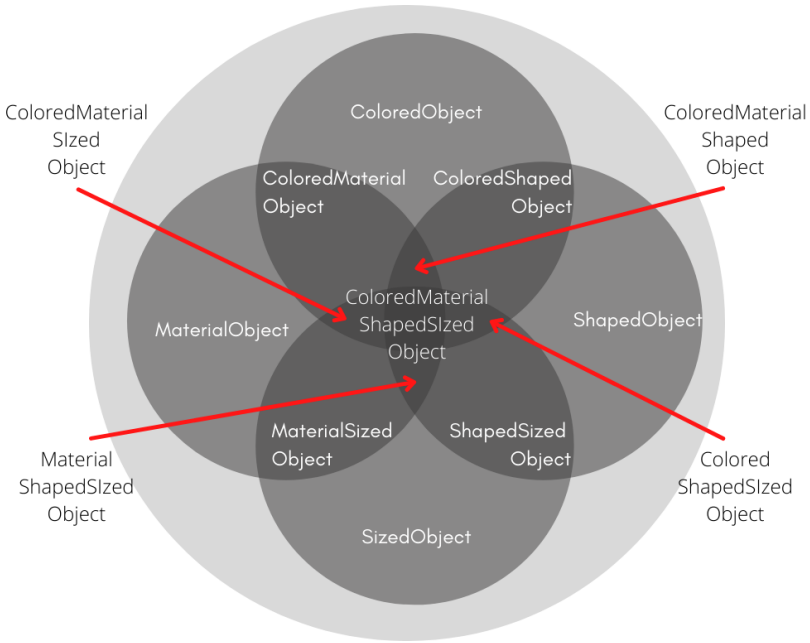


Figure 4.6: Venn diagram of object-attribute classes.

BlueMetallicObject is a subclass of both *BlueObject*, *MetallicObject*, and *ColoredMaterialObject*. Figure 4.7 shows some of the classes belonging the second hierarchy (in red) nested with the classes of the first hierarchy (in black). The properties of *KlevR* are the spatial relationships between the objects, both direct and not direct, which are subproperty of *hasNear* and *hasDirectlyNear*, respectively. Table 4.2 shows their domain, range, and characteristics (they are not supported by each profile, explanation follows). By defining direct and indirect properties, we have also built a property hierarchy with

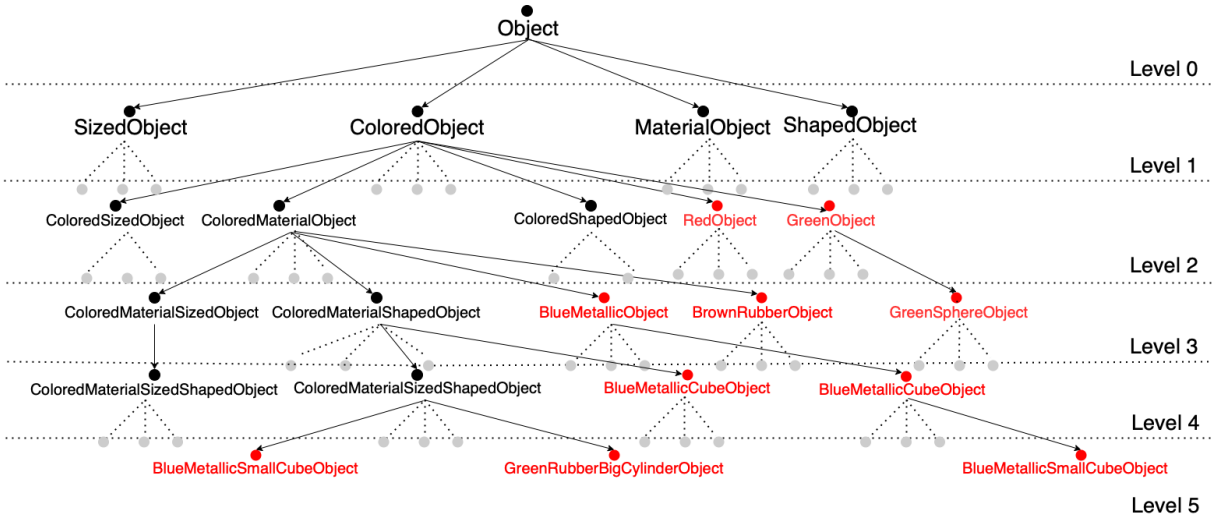


Figure 4.7: Complete KlevR class hierarchy.

the possibility of property chains, increasing the required reasoning effort of the reasoner. The design aspects just described are common for all the OWL profiles of *KlevR*. Then, each profile has been designed with different features accordingly to its entailment regime.

KlevR_RL is the OWL RL profile of *KlevR*. With existential quantification and *someValuesFrom* allowed, we have added classes to encode the semantics of the attribute values. For example, the class *Red* stands for "a red Thing" and is a subclass of *Color*. Their instances will be anonymous nodes, and are used in combination with four new properties whose semantic is "having that attribute", namely *hasColor*, *hasMaterial*, *hasShape* and *hasSize*. Each class of the first level of the hierarchy is restricted with an existential quantification. For example

BlueObject equivalentTo hasColor some Blue

defines the class of blue objects as the set of objects having blue color. *KlevR_RL* also supports transitive and symmetric properties, resulting in the most expressive profile.

KlevR_QL is the OWL QL profile of *KlevR*. The OWL QL profile is less expressive than the RL one, it supports all the RL expressions and axioms we used in *KlevR_RL* except for existential quantifications and transitive property. For this reason, the semantics of the attributes lies only in the class names in the hierarchy. Indeed, *KlevR_QL* was modeled starting from *KlevR_RL* and then removing the attribute-type classes (Color, Material, Shape and Size), their subclasses, the property hasColor, hasMaterial, hasShape, and hasSize, and the transitivity from the properties.

KlevR_EL is the OWL EL profile of *KlevR*. The OWL EL profile is less expressive than the RL one, but in our case, it doesn't differ so much from *KlevR_QL*; in particular,

Property	Domain	Range	inverseOf	characteristics
hasNear	Object	Object	-	transitive, symmetric
hasDirectlyNear	Object	Object	-	symmetric
hasOnLeft	Object	Object	hasOnRight	transitive
hasDirectlyOnLeft	Object	Object	hasDirectlyOnRight	-
hasOnRight	Object	Object	hasOnLeft	transitive
hasDirectlyOnRight	Object	Object	hasDirectlyOnLeft	-
hasOnFront	Object	Object	hasBehind	transitive
hasDirectlyOnFront	Object	Object	hasDirectlyBehind	-
hasBehind	Object	Object	hasOnFront	transitive
hasDirectlyBehind	Object	Object	hasDirectlyOnFront	-

Table 4.2: KlevR properties description.

supports its expressions and axioms except for inverse and symmetric properties. In addition, it supports transitivity and disjunction. *KlevR_EL*, in fact, was modeled starting from *KlevR_QL* and then changing the characteristics of the properties.

4.1.3. Information Needs

As per the competency questions, which we can use to frame the scope by capturing the information needs the ontology is intended to answer, we have translated the CLEVR questions into SPARQL queries in order to set the expressivity of the ontology so that the reasoning effort is maximized.

For each CLEVR *scene*, a set of questions is associated (train and validation sets also have the answers), generated using different templates. The templates are made of placeholders (reported in Table 4.3), that are substituted with randomly sampled values of the attributes to generate the real questions. Placeholders can be empty whenever an attribute or a relationship is omitted from the question. For example, the question *"How many red cubes are there?"* is generated using the template *"how many <Z> <C> <M> <S> are there?"* replacing the attributes "<C>" and "<S>" with "red" and "cube", <Z> and <M> instead are empty. CLEVR has 90 families of questions that can be grouped in 13 basic types. Table 4.4 shows some example templates for each type of basic question. Complex questions are made by composing the basic ones using spatial relationships, allowing the creation of a possibly infinite number of questions. An example is *"How many <Z> <C> <M> <S> are <R> of the <Z1> <C1> <M1> <S1>"* where <R> could be any of the spatial relationships.

Placeholder	values
$< S >$	Shapes: Cube, Sphere, Cylinder
$< Z >$	Size: Big, Small
$< M >$	Materials Rubber, Metal
$< C >$	Colors: Blue, Brown, Gray, Red, Yellow, Green, Purple, Cyan
$< R >$	Relationships: Left, Right, Front, Behind

Table 4.3: CLEVR template placeholders.

Each question is associated with a functional program (FP) that yields the answer to the question when applied to the scene graph. FP are built from basic functions that correspond to elementary operations of VQA (i.e., querying object attributes, counting

Type	Questions
Count	How many <Z> <C> <M> <S> are there?
	What number of <Z> <C> <M> <S> are there?
Exist	Are there any <Z> <C> <M> <S>s?
	Are any <Z> <C> <M> <S>s visible?
	Is there a <Z> <C> <M> <S>?
Query shape	What shape is the <Z> <C> <M> <S>?
Query color	The <Z> <C> <M> <S> has what color?
Query size	What is the size of the <Z> <C> <M> <S>?
Query material	What is the <Z> <C><S> made of?
Compare integer, less	Is there fewer <Z> <C> <M> <S> than <Z1> <C1> <M1> <S1>?
Compare integer, greater	Is there more <Z> <C> <M> <S> than <Z1> <C1> <M1> <S1>?
Compare integer, equal	Are the <Z> <C> <M> <S> the same number of <Z1> <C1> <M1> <S1>?
Compare size	Is the <Z> <C> <M> <S> the same size of the <Z1> <C1> <M1> <S1>?
Compare shape	Does the <Z> <C> <M> <S> and the <Z1> <C1> <M1> <S1> have the same shape?
Compare material	Are the <Z> <C> <M> <S> and the <Z1> <C1> <M1> <S1> made of the same material?
Compare color	Is the <Z> <C> <M> <S> the color of the <Z1> <C1> <M1> <S1>?

Table 4.4: Examples of CLEVR question templates [8]

objects, comparing values) and can be chained to build complex questions as shown in Figure 4.8.

FPs are essential for benchmarking VQA models since they tell us exactly which reasoning abilities are required to solve it, allowing performance comparison on questions requiring different types of reasoning.

Given the *KlevR* profiles, we have translated the competency questions into SPARQL queries. Listings 4.1 shows an example of the query related to the question *""How many <Z1> <C1> <M1> <S1>s are there?""* for the RL profile. We need to filter dinjuncted blank nodes ?c, ?z, ?m, ?s since the RL profile doesn't support class disjunction. The complete list of queries is reported in the appendix B.

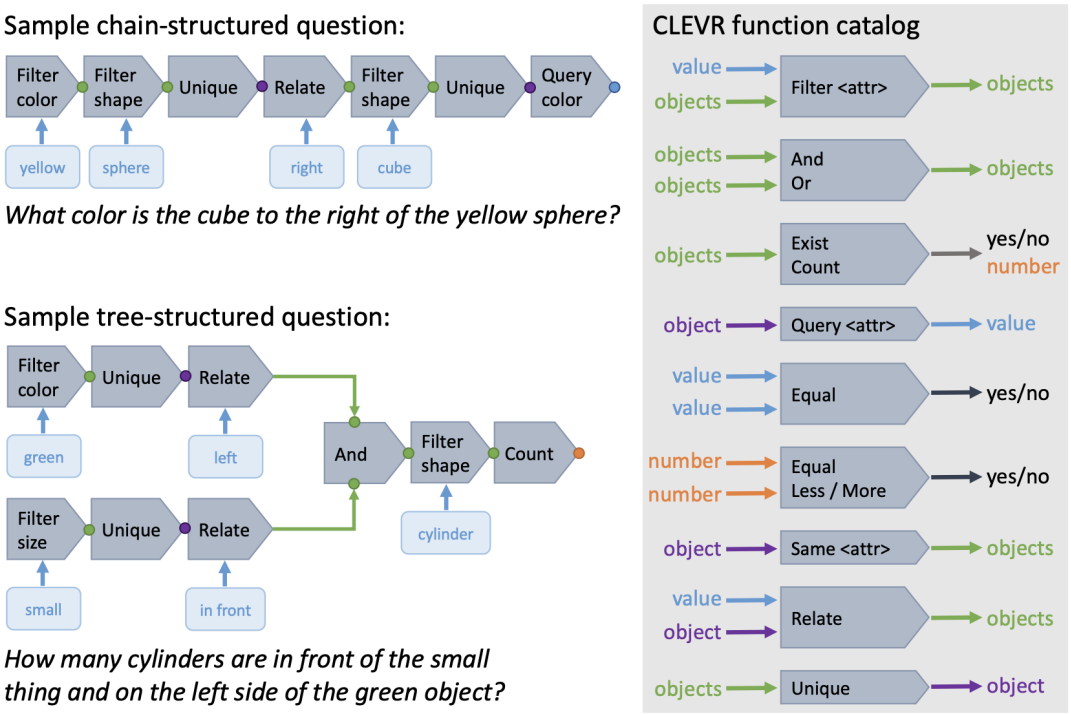


Figure 4.8: Left: Examples of questions and their associated functional programs. Right: Catalog of basic functions used to build questions. [8].

```
SELECT (COUNT(?x) as ?xCount)
WHERE
{
   ?x ex:hasColor  ?c ;
        ex:hasSize ?z;
        ex:hasMaterial ?m;
        ex:hasShape ?s .

       ?c  rdf:type ex:<C> .
       ?z  rdf:type ex:<Z> .
       ?m  rdf:type ex:<M> .
       ?s  rdf:type ex:<S> .

FILTER(?s <> ?c) FILTER(?s <> ?z)
FILTER(?s <> ?m) FILTER(?c <> ?z)
FILTER(?c <> ?m) FILTER(?z <> ?m)
    }
```

Listing 4.1: RL SPARQL query for "How many <Z1> <C1> <M1> <S1>s are there?".

4.1.4. KlevR Population

In populating the KBs, our goal was to feed the reasoner with individuals belonging to the lower level classes (e.g. *BlueMetallicSmallCubeObject*), forcing it to scale up the hierarchy using the *subclass* entailment rule. Moreover, we wanted the objects to be related only with their direct positional relationships, to enforce the reasoner to inference about subproperties and transitivity. With the dataset annotation, we have obtained an RDF graph for each scene, which we call *KlevR scene graph*.

Definition 4.1.1 (KlevR scene graph)**.** A **KlevR scene graph** is the RDF encoding of a plain *scene graph*. The graph's nodes describe the objects and the attributes, and relations connect objects to objects and objects to attributes.

Figure 4.9 shows a *KlevR scene graph* with the objects labeled with RDF annotation in Turtle format.

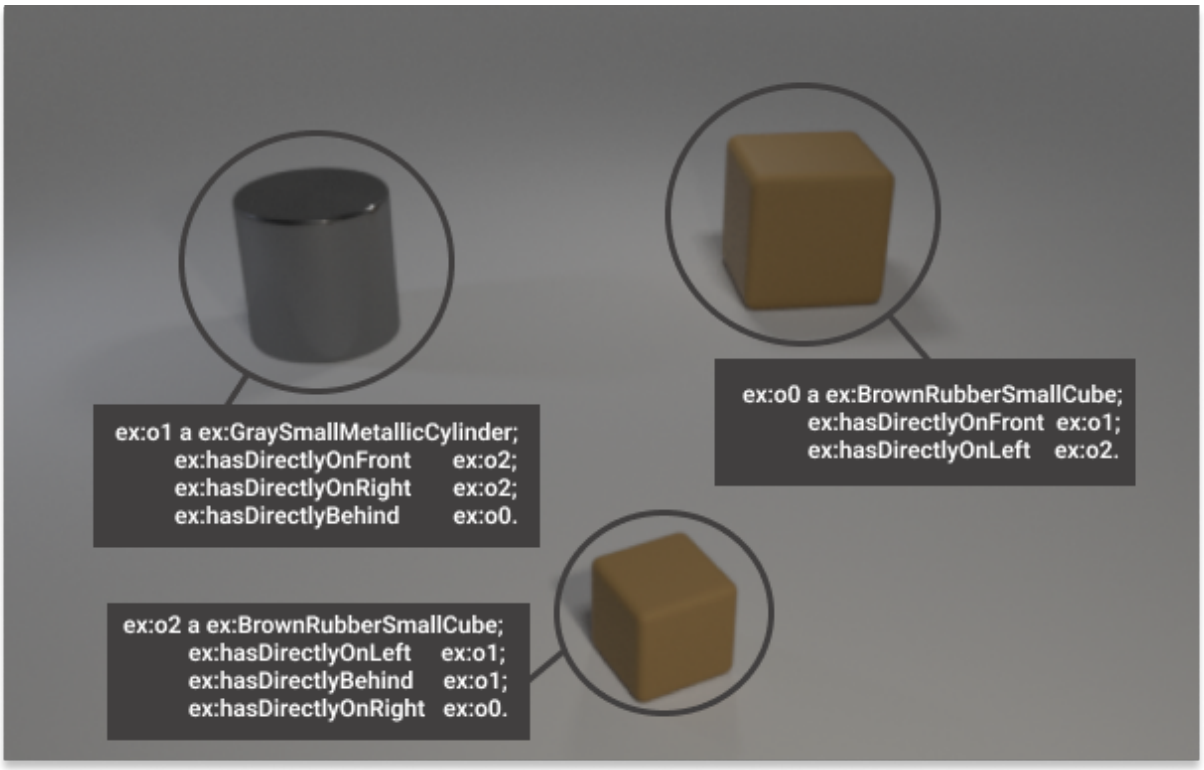


Figure 4.9: KlevR scene graph example.

CLEVR provides a file in JSON format which reports the *scene graphs* for each image. First, we have adjusted the given file removing all the information intended for the blender. Then, we gave ids to objects, and set the direct positional properties between the objects.

For example, Listing 4.2 shows an example of CLEVR scene and Listing 4.3 shows the scene after the adjustment for the annotation.

```
{
"image_index": 0, "image_filename":
"CLEVR_train_0.png", "split": "train",
"objects":[
    {"color": "blue", "size": "large", "shape": "cube",
    "material": "rubber", "rotation": 269.85,
    "3d_coords": [-1.371, 2.079, 0.699],
    "pixel_coords": [269, 88, 12.661]},...],
"relationships": {
    "right": [[2,5],[0,2,3,4,5],[5],[0,2,4,5],[0,2,5],[]],
    "behind": [[],[0,5],[0,1,5], [0,1,2,5],[0,1,2,3,5],[0]],
    "front": [[1,2,3,4,5],[2,3,4],[3,4],[4],[],[1,2,3,4]],
    "left": [[1,3,4],[],[0,1,3,4],[1],[1,3],[0,1,2,3,4]]},
"directions": {
    "right": [0.65, 0.75, -0.0], "behind": [-0.75, 0.65, 0.0],
    "above": [0.0, 0.0, 1.0], "below": [-0.0, -0.0, -1.0],
    "left": [-0.65, -0.75, 0.0], "front": [0.75, -0.65, -0.0]}
}
```

Listing 4.2: Example of a CLEVR scene in JSON format.

```
{
"image_index": 0,
"objects": [ {
    "id": "o0",
    "color": "Blue","size": "Large",
    "shape": "Cube","material": "Rubber",
    "hasOnLeft": ["o1","o3","o4"],"hasOnRight": ["o2","o5"],
    "hasOnFront": ["o1","o2","o3","o4","o5"],"hasBehind": [],
    "hasDirectlyOnLeft": ["o1"],"hasDirectlyOnRight": ["o2"],
    "hasDirectlyOnFront": ["o1"],"hasDirectlyBehind": []},...],
}
```

Listing 4.3: Example of a CLEVR scene in the adjusted JSON format.

To annotate the dataset, we have used YARRRML's Matey[1], a web application that produce RML mappings and relative RDF outputs using YARRRML, a human readable text-based representation for declarative generation rules. We defined the YARRRML mapping reported in Listing 4.4, whose output is a *KlevR scene graph* in Turtle format. The result for the example scene in Listing 4.3 is reported in Listing 4.5.

```
prefixes:
  ex: "http://example.com/"
mappings:
  person:
    sources:
      - ['data.json~jsonpath', '$.objects[*]']
    s: http://example.com/$(id)
    po:
      - [a, ex::$(color)$(material)$(shape)$(size)Object]
      - [ex:hasDirectlyOnRight, ex:o$(hasDirectlyOnRight[0])]
      - [ex:hasDirectlyOnLeft, ex:o$(hasDirectlyOnLeft[0])]
      - [ex:hasDirectlyOnFront, ex:o$(hasDirectlyOnFront[0])]
      - [ex:hasDirectlyBehind, ex:o$(hasDirectlyBehind[0])]
```

Listing 4.4: YARRRML mapping for the KlevR scene graphs.

```
PREFIX ex: <http://example.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
<ex:o0> <rdf:#type> <ex:GrayRubberCubeLargeObject>.
<ex:o0> <ex:hasDirectlyOnLeft> <ex:o1>.
<ex:o0> <ex:hasDirectlyOnRight> <ex:o2>.
<ex:o0> <ex:hasDirectlyOnFront> <ex:o1>.
...
```

Listing 4.5: Example of a KlevR scene graph in Turtle format.

[1] https://rml.io/yarrrml/matey/

4.1.5. Dataset Materialization

Once we have annotated every CLEVR scene, we have proceeded with the materialization. The reasoner of our choice was HermiT[2] because it provides a simple to use API and it is widely tested and benchmarked. Unlike a standard KR task where the ABox contains all the individuals to materialize, each *KlevR scene graph* consists of a single ABox. To perform the inference according to the OWL profiles, we merged the three ontologies with the ABoxes, then we have materialized them together, and finally we have subtracted the ontologies to obtain the realized Aboxes. Figure 4.10 illustratively shows a materialized *KlevR scene graph*, labelling the image with its completed RDF annotation. With the materialization, we have obtained the set of targets for training Deep Learning models on the approximate materialization task. The targets represent the ground truth, both for the performance by means of computational time and the accuracy of the inference.

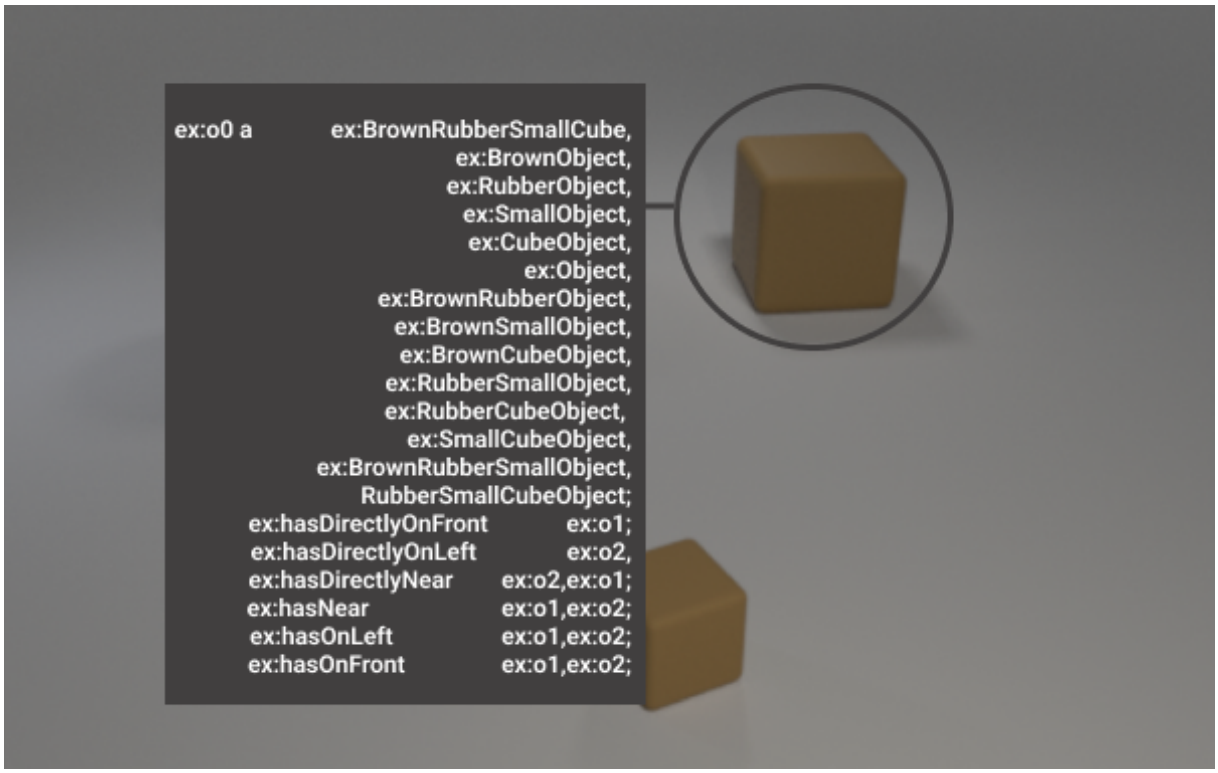


Figure 4.10: Materialized KlevR scene graph example.

[2]http://www.hermit-reasoner.com

4.2. Baseline solution: DeepR

In building *KlevR*, we have obtained three datasets. In particular, for each OWL profile, we have a training and a testing set that we define as $\mathcal{D} = \{< x_i, t_i >\}$ where x_i is a *KlevR scene graph* and t_i is a *materialized KlevR scene graph*. To design and train the baseline model for the benchmark, the first step consists of the choice of the embedding model for the RDF graphs which is described in Subsection 4.2.1. The second step, reported in Subsection 4.2.2, consists of the design of the Variational Autoencoders. Finally, in Subsection 4.2.3 we discuss how to evaluate the baseline using the standard Deep Learning metrics and how they could be interpreted to evaluate the quality of the approximation.

4.2.1. Embedding Model

As we discussed in Chapter 3, the embedding model we propose for our baseline is the Subject-Predicate-Object (SPO), a 3D-tensor representation of an RDF graph. With an SPO tensor, each triple is represented as an entry $\mathcal{X}_{i,j,k}$, which is such that $\mathcal{X}_{i,j,k} = 1$ if the triple $< s_i, p_k, o_j >$ exists in the graph, $\mathcal{X}_{i,j,k} = 0$ otherwise. In particular, each tensor slice encodes a properties p_k, while its rows and column are made of the concatenation of all the existing subjects and objects in the scenes. Figure 4.11 shows an example of a KlevR scene graph fragment encoded with the SPO embedding. The advantage of using this representation is that the semantic of the ontology is brought to the encoding. Every triple, in fact in the KG, can be represented with an entry in the tensor, not only the ones belonging to the ABox. However, in our approach, the triples of the TBox are subtracted from the graphs before encoding it. The semantic is still implicitly part of the representation because the position (indexes) of its elements is constant throughout the scenes, which means that the structure of the tensors reflects a part of the semantics of the ontology. If for example s_i and o_j are linked with a symmetric property p_k, both $\mathcal{X}_{i,j,k}$ and $\mathcal{X}_{j,i,k}$ will be equal to 1. This property is also valid across different slices, for example with inverse or transitive properties and for subclass relationships. Encoding all the scenes in this way introduces a bias for the model, but it is essential to comply with **R1** 3.3. **R2** 3.3 is also respected since an SPO tensor embeds an entire KlevR scene graph. For what concern **R3** 3.3, the algorithm that builds the embedding (presented in Subsection 5.2.1) keeps track of the indexes of the elements, building a mapping function between the RDF and the tensor representations.

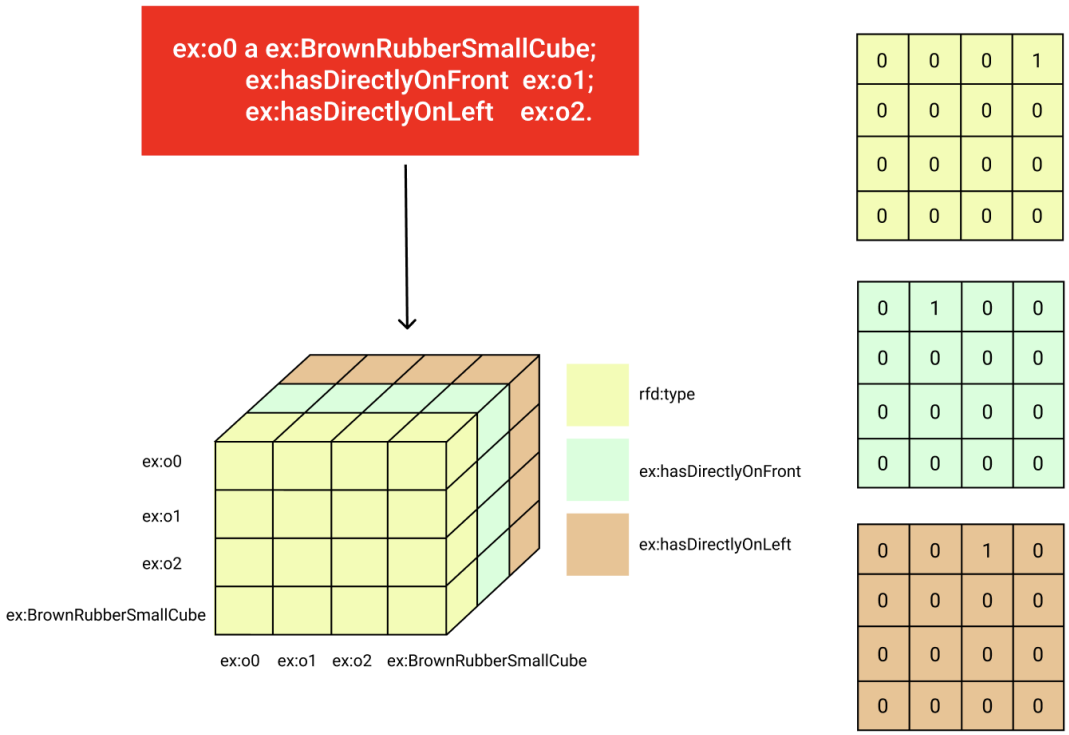


Figure 4.11: Example of SPO embedding.

4.2.2. VAE Design

In order to evaluate our benchmark for the approximation of the materialization task and set a baseline for future assessments, we have designed a model based on a simplified VAE called **DeepR**. In particular, *DeepR* has three different architectures, one for each OWL profile of Klevr. Our approach in designing the architectures was not model-driven. In the research community, there are no other studies regarding VAEs for reasoning approximation. Thus, we have decided to plan a hyperparameters tuning to find the best architectures that maximize the models' accuracies on the data. In particular, we have tested different number of layers, filters per layer, learning rates and optimizers. The only a priori decisions we have made are about the activation and loss functions. Subsection 2.3.3 and in particular Equation 2.2 describe the trade-off between the reconstruction term and the regularization term of the loss function. Our approach consists of ignoring this trade-off and using the Binary Cross Entropy (BCE) as loss function. With the BCE, there is no clear separation in the tasks performed by the encoder and the decoder, but the whole model contributes to the approximation, getting rid of redundant or useless information in the encoding and adding new triples in the encoding and decoding block. The choice of a simpler loss function leads to a simplified VAE, however, in Section 6.2 we will discuss an extension of the model that strictly follows the design of standards VAEs,

encoding the input in the first part of the architecture. For what concern the activation function, to constraint the output of the output of the model to be in the range $[0 : 1]$, we opted for the sigmoid function.

4.2.3. Metrics

As evaluation metrics to test the baseline on *KlevR*, we opted for standards Deep Learning metrics (i.e., accuracy, precision, and recall), giving them an interpretation from a reasoning point of view. Thanks to the SPO embedding, we evaluated the model's performances as if it was a classification task. In particular, true positives are valid triples predicted by the model, true negatives are triples not asserted both by the model and the reasoner, false positives are incorrectly asserted triples by the model, and false negatives are missing triples by the model. With this interpretation, *precision* is the percentage of valid triples correctly asserted by the model with respect to the total number of assertions, while *recall* is the percentage of valid triples asserted by the model with respect to the total number of expected assertions.

Moreover, given $\mathbb{T}$ the target tensor for *deepR*, $\mathbb{T}^*$ its prediction, and $\mathbb{X}$ the input, we have computed the metrics, both using the couples $< \mathbb{T}^*, \mathbb{T} >$, and the couples $< \mathbb{T}^* - \mathbb{X}, \mathbb{T} - \mathbb{X} >$. While the first approach focuses only on the predictions, the second one takes into consideration the input as well. Removing the input from the predictions and the outputs allowed us to distinguish if a triple is asserted by the model in approximating the materialization or if it was already present in the ABox.

Due to the sparse nature of the SPO embedding, we expected the models to push their weights towards zero during the training phase. In Figure 4.11, for example, three triples are mapped into a 4x4x3 tensor made of three 1s and forty-five 0s. To deal with the unbalanced classification problem, we evaluated the models using the *threshold-moving* technique, which consists of comparing the confidence of a model's predictions with different thresholds. Thus, we exploited the Receiver Operating Characteristic (ROC) curve to understand the trade-off between the true-positive rate and false-positive rate for different thresholds and the ROC Area Under the Curve (AUC) to compare the two approaches (with and without the input triples). The AUC represents the degree of separability. It tells how much a model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting true negatives and true positives. Given the ROC, the highest value of the Geometric Mean between the true-positive rate and the false-positive rate denotes the best threshold.

The ROC curve focuses on all model predictions. To evaluate the materialization

made by the model, however, the main focus should fall on the asserted triples (the positives). For this reason, we also relied on the Precision-Recall (PR) curve to compare the precision against the recall on different thresholds. Given the PR, the highest value of F-measure denotes the best threshold.

5 | Benchmark Implementation Experience and Results

This chapter presents the implementation experience based on the benchmark design described in the Chapter 4. In particular, Section 5.1 describes the development of the *KlevR* KGs, Section 5.2 describes the implementation of *DeepR*, and Section5.3 presents the results of our experiments testing the benchmark. The Github repository[1] of this thesis contains every algorithm or file we have produced. The setup we have used consists of an Ubuntu virtual machine with 64 VCPUs, 128GB ram and 1TB hard disk. We opted for more powerful CPUs instead of a GPU because our models are not sofisticated. Instead, the amount of data that has to be loaded on memory during our pipeline faced a bottleneck on CPU usage. In general, Stronger CPUs promises faster data transfer hence promising faster calculations. For what concerns the Deep Learning, the framework chosen for this Thesis is Pytorch[2], due to its popularity in the research field [4].

5.1. Benchmark Task: KlevR

This Section focuses on the technicalities of the KR aspect of our implementation. In particular, Section 5.1.1 describes the process of developing the ontologies, and Sections 5.1.2 the dataset annotations and the materialization.

5.1.1. KlevR TBox Generations

The combinatorial nature of the CLEVR entities makes extremely time-consuming the usage of traditional KR tools like Protegè[3]. For this reason, we have developed an algorithm to automate the hierarchy building. Algorithm 5.1 shows a fragment of the hierarchy builder. In particular, the algorithm loops over the possible attribute values and outputs the triples for the class and subclass declarations (with functions *write_class_statement*

[1] https://github.com/pake97/KlevR-and-DeepR-Master-Thesis.git
[2] https://pytorch.org
[3] https://protege.stanford.edu

and *write_subclass_statement*). The complete code includes more nested loops to declare every possible combination of attributes and every subclass relationship. Once the hierarchy was built, we relied on Protegè to define the properties and finalize the ontologies.

Algorithm 5.1 Hierarchy builder fragment

```
Require: List<String> colors, materials, sizes, shapes
  for c in colors do
    l1_class_name ← c+"Object"
    l0_class_name ← "Object"
    for m in materials do
      l2_class_name ← c + m+"Object"
      for s in sizes do
        l3_class_name ← c + m+"Object"
        for sh in shapes do
          l4_class_name ← c + m+"Object"
          write_class_statement(l4_class_name)
          write_subclass_statement(l4_class_name, l3_class_name)
        end for
      end for
    end for
  end for
```

Table 5.1 reports the statistics of the three OWL profiles of *KlevR* for what concerns the resulting number of classes, properties, and axioms.

Profile	#Classes	#Properties	#Axioms
KlevR_EL	339	10	1672
KlevR_QL	339	10	1673
KlevR_RL	358	15	2109

Table 5.1: Statistics of KlevR profiles.

As Table 5.1 shows, the EL and QL profiles have fewer classes and properties with respect to the RL profile because the latter includes specific classes for the attributes' values and properties for the existential quantifications (see Section 4.1).

Regarding the Aboxes, it is important to highlight some statistics about the CLEVR scenes, which reflect in the *KlevR scene graphs*. In particular, Figure 5.1 shows the distribution of the number of objects per scene. Considering the annotation of the scenes described in Section 4.1.4, for each object, an ABox includes at most five triples (one for the type, four for the direct positional relationships). Therefore, the dimension of the

ABoxes ranges between 15 and 50 triples. Figure 5.2, instead, shows how the attributes' values are equally distributed over all the objects.

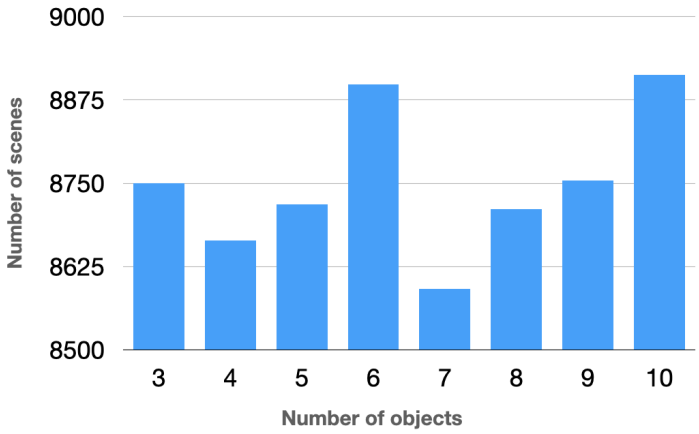
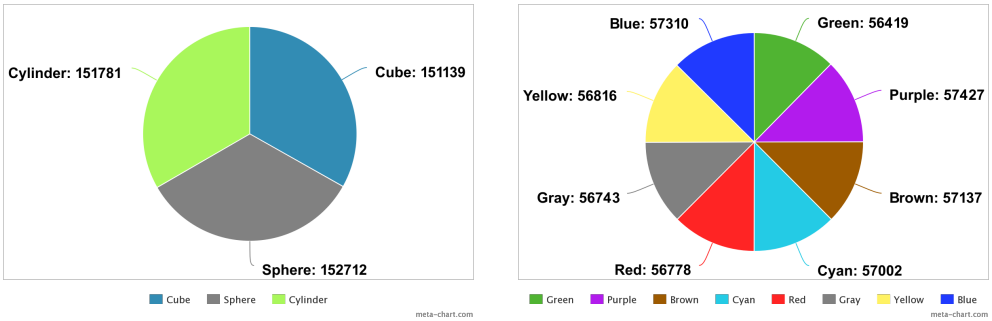
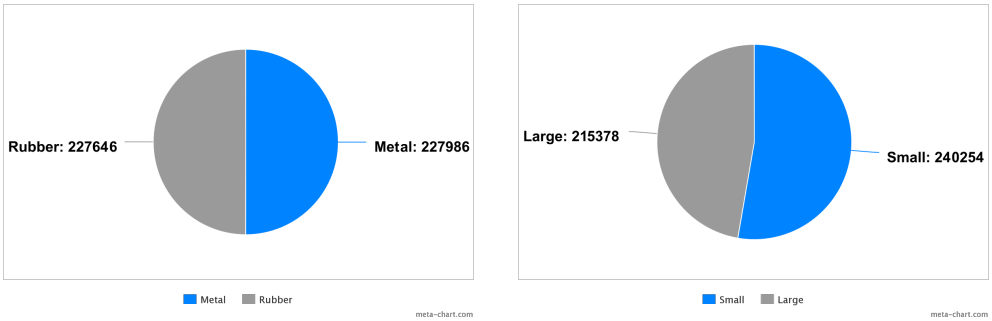


Figure 5.1: Distribution of objects per scene.



(a) (b)



(c) (d)

Figure 5.2: Attributes distributions.
(a) : Shape, (b) : Color, (c) : Material, (d) : Size.

5.1.2. KlevR Scene Graphs ABox Population and Materialization

To make the benchmark scalable, extensible, and reusable for further investigations, we designed a data pipeline from the CLEVR *scene graphs* to the *materialized KlevR scene graphs*. Figure 5.3 shows the components of the pipeline and the status of the data before and after each step.

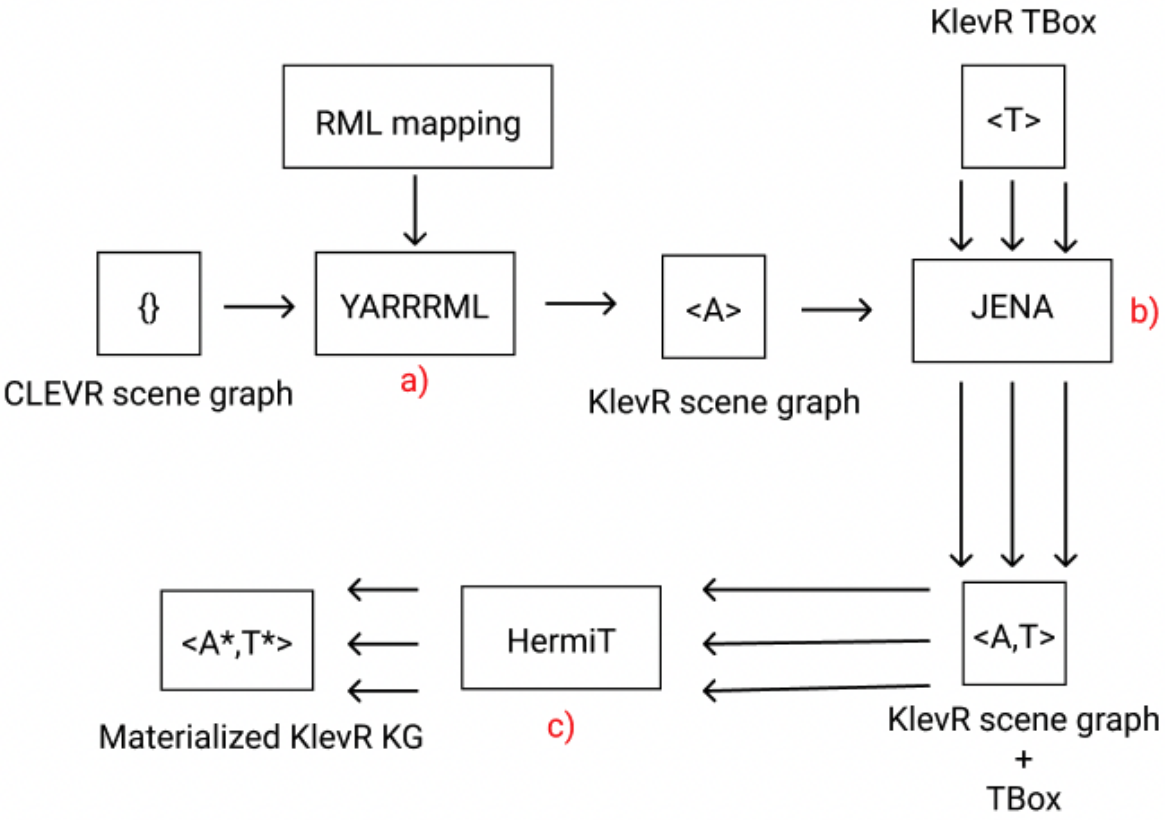


Figure 5.3: Implementation pipeline for the KG population and materialization.

Step a) consists of KGs population. To annotate every CLEVR *scene graph*, we opted for YARRRML's Matey as described in Subsection 4.1.4. Once the KG is populated, it should be materialized. In particular, thanks to Apache Jena[4], in step b), we have merged every *KlevR scene graph* with the three ontologies of *KlevR*. Therefore, we obtained three Knowledge Graphs for each CLEVR scene. Then in step c), we have materialized every graph using the HermiT API[5]. The outputs of the pipeline are three *materialized KlevR*

[4]https://jena.apache.org
[5]http://www.hermit-reasoner.com

scene graphs (one for each OWL profile of *KlevR*) for every CLEVR *scene graph*. Steps a) to c) guarantee flexibility of the pipelines, which easily welcomes the insertion of new scene graphs and updates to the annotation and/or to the Tboxes.

5.2. Baseline Solution: DeepR

This section describes the implementation experience of the baseline with which we tested the approximation of the materialization task on the benchmark. In particular, Section 5.2.1 describes SPO embedding algorithms we developed, and Sections 5.2.2 the hyper-parameter tuning of *DeepR*.

5.2.1. SPO Embedding

To embed each *KlevR scene graph* with the SPO embedding model (see Section 4.1.4), we developed Algorithm 5.2, which loops over the triples of a *KlevR scene graph* and embeds them in a 3D-tensor. It uses two hashing tables to map each triple $< s, p, o >$ into an entry of the tensor. The hashing table E returns the index of the row (s) and column (o), which are the entities, while the hashing table P returns the slice (p) of which is the property. A new record in the hashing tables is added when a new entity or property is found, assigning it a new unique index. At the end of the dataset embedding, the hashing tables represent the mapping function between the KlevR scene graph and the SPO tensors.

Algorithm 5.2 SPO embedding

Require: Hash tables E,P,list list_triples, integers max_e, max_p
Ensure: Tensor T
  $T \leftarrow$ Zeros(max_e, max_e, max_p)
  for $< s, p, o >$ in $list\_triples$ **do**
    $idx_s \leftarrow E.get(s)$ {Create new key if not present}
    $idx_o \leftarrow E.get(o)$ {Create new key if not present}
    $idx_p \leftarrow P.get(p)$ {Create new key if not present}
    $T[idx_s][idx_o][idx_p] \leftarrow 1$
  end for
  return T,E,P

The complexity of Algorithm 5.2 is $O(n)$, where n is the number of triples. The parameters max_e and max_p are the dimension of the tensor T and have been computed as the maximum number of nodes and properties in the dataset's graphs.

We have applied a slight modification for the embedding of the *KlevR* RL profile. As

Section 4.1 describes, there is one class for each value of CLEVR attributes (for example *Red* for the color "red"). Moreover, each class of the first level of the class hierarchy is restricted with an existential quantification that relates an object with a blank node representing the attribute, which impacts the number of individuals. For example, if a scene includes an object $o1$ belonging to the class *RedObject*, the triple

$$o1 \quad hasColor \quad b1$$

exists, where $b1$ is a blank node belonging to the class *Red*. Figure 5.4 shows the example of how the information *"o1 is red"* is captured by the three *KlevR* profiles. HermiT, which we have used for the materialization, could instantiate multiple blank nodes of the same type, resulting in the explosion of triples in the RDF graph and the loss of the mapping between the scene and the embedding. To avoid this, we manually added one blank node for each attribute to the embedding and mapped every repeated node to the same index when it was found.

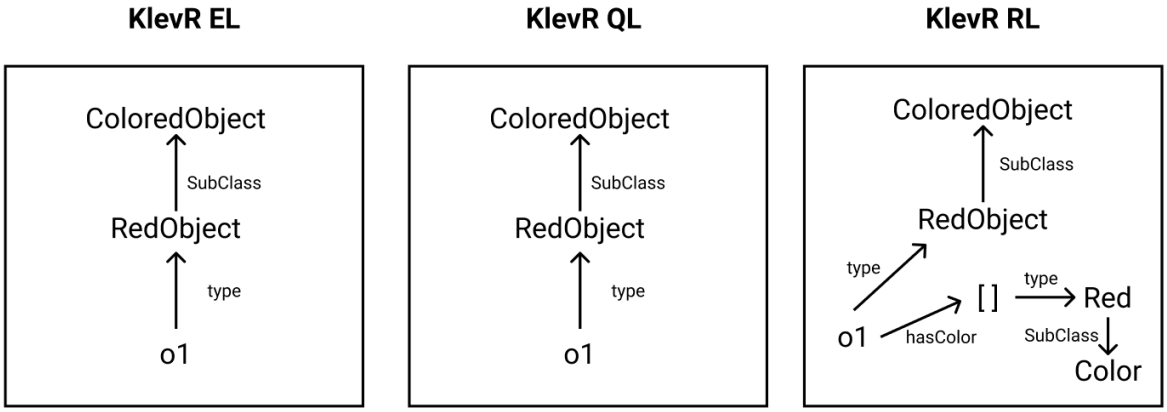


Figure 5.4: Example of a KlevR scene graph for each OWL profile.

5.2.2. VAE Implementation

In Section 4.2.2, we discussed that the architectures of *DeepR* are the result of an hyperparameters tuning. To plan the search for the optimal set of parameters, we have used a hyperparameter optimization framework compatible with Pytorch, which is called Optuna[6]. In particular, we have defined the model's accuracy as the objective function to maximize, then we have set the searchable parameters with the ranges shown in Table 5.2. The most important parameters are the number of autoencoder layers (the value is

[6]https://optuna.org

the same for the encoder and the decoder) and the kernel size because they determine the models' architectures. The latter is bounded to 15 because a higher number could cause an extreme dimensionality reduction, resulting in the loss of information. Moreover, the kernel size and the stride of the pooling layers are chosen for each layer of the network.

Parameter	**Range**
optimizer	["Adam", "RMSprop", "SGD"]
learning rate	$[10^{-5}:10^{-1}]$
Num layers	[1:5]
Kernel size	[3:15]
Max pool Stride	[1:2]

Table 5.2: DeepR hyperparameters.

We have run 1000 trials on the training set and obtained as results the values reported in Table 5.3. The difference between the depth of the networks can be interpreted from a complexity point of view: deeper and more complex models usually approximate more complex functions. Indeed, reasoning with EL and DL has a higher upper bound in terms of reasoning complexity with respect to QL (see Table 3.1).

Hyperparameter	DeepR EL	DeepR QL	DeepR RL
num layers	4	1	5
kernel size layer 1	9	9	5
stride pool layer 1	2	N/A	1
kernel size layer 2	11	N/A	9
num filters layer 2	2	N/A	2
stride pool layer 2	1	N/A	1
kernel size layer 3	9	N/A	15
num filters layer 3	3	N/A	3
stride pool layer 3	2	N/A	1
kernel size layer 4	7	N/A	13
num filters layer 4	4	N/A	3
stride pool layer 4	N/A	N/A	2
kernel size layer 5	N/A	N/A	3
num filters layer 5	N/A	N/A	4

Table 5.3: DeepR models hyperparameters.

5.3. DeepR Evaluation on KlevR.

This section presents the evaluation of the *DeepR* architectures on *KlevR*. In particular, we wanted to verify the hypothesis listed in Chapter 3. For this reason, we have evaluated both the efficiency of the approximation made by the model in terms of time performances and the quality of the approximation with the metrics described in Section 4.2.3.

5.3.1. DeepR QL

Given the parameters in Table 5.3, the final architecture of *DeepR* for the QL profile is shown in Figure 5.5.

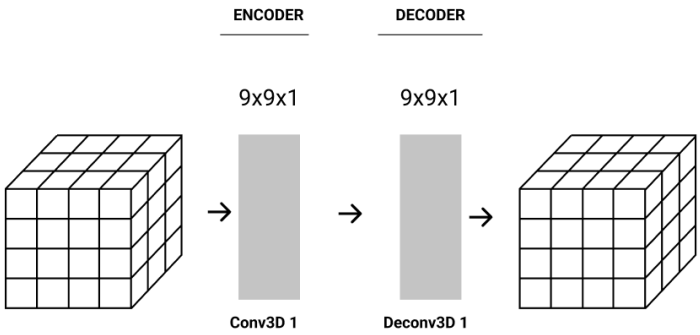


Figure 5.5: DeepR QL architecture.

DeepR QL has the simplest architecture. A possible interpretation regarding the model's architecture could rely on the complexity of reasoning with the OWL profile, which in fact is the lowest. Table 5.4 reports the time performances of the model in its different phases (scene parsing, embedding, training and testing) and the time performances of HermiT on the entire dataset.

Scene parsing	SPO embedding	DeepR Training	Total	Total/scene	DeepR testing	Testing/scene	HermiT	HermiT/scene
5027.3	4141.0	555266.1	564434.4	8.1	8835,71	0.589	536749	5.37

Table 5.4: Computing time of DeepR QL and HermiT.

Due to the loading and storing of a huge amount of data in the main memory and the multiple epochs, *DeepR* is slower than HemiT in the training phase. However, during the testing phase, the model widely outperforms HemiT, reaching the efficiency theorized in our hypotheses. Figure 5.6 shows a comparison between the HermiT average reasoning time per scene and *DeepR QL* training (limited to the last epoch) and testing time per

scene. The model reaches an average speedup of 14x with respect to the training and a speedup of 10x with respect to HetmiT.

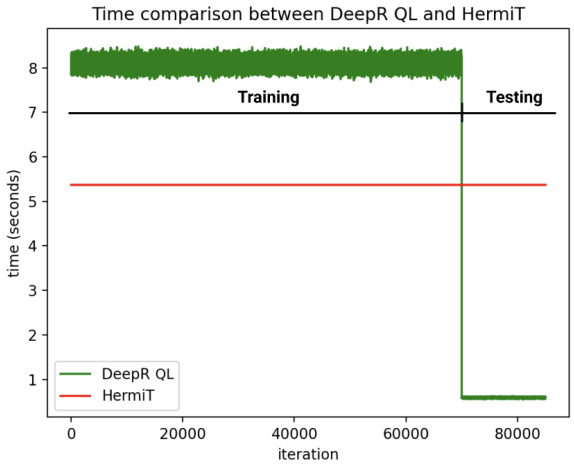


Figure 5.6: Time comparison between HermiT and DeepR QL.

Despite the results of the efficiency evaluation, *DeepR QL* does not perform well for what concerns the approximation quality. As discussed in section 4.2.3, we exploited the ROC curve to investigate the trade-off between the true-positive and false-positive rates for different thresholds and find the best one. Figure 5.7 shows the ROC curves derived from the predictions made by *DeepR QL* on the test set. The blue curve refers to the standard predictions, while the green one is derived by removing the triples belonging to the input from the predictions and the targets. The two ROCs are close to each other, denoting a similar degree of separability in both the experiments, and the high values of the AUC suggest good overall performances. However, the steepness of the curves reveals a high true-positive rate and a low false-positive rate, caused by the sparse nature of the SPO embedding and the resulting tendency of the model to push the weights toward zero. The best threshold for both experiments is in fact 0.000144. The overall accuracy reached by *DeepR QL* with the best threshold is 94.236%.

Unlike standard Deep Learning experiments, the high value of the accuracy in our baseline does not prove the quality of the approximation. The high number of 0s in the embedding leads to an elevated number of true negatives which increases the accuracy. To better deal with the unbalanced problem, we also derived the PR curves for both experiments, which are reported in Figure 5.8. Despite the high accuracy, the model performs a poor inference in terms of both correctly predicted triples, which is pointed by the low precision, and the number of predicted triples, which is pointed by the low recall. Moreover, the difference between the two PR curves highlights the real model's

ability to infer new triples. A lower precision means fewer true positives and/or more false positives, which translates into fewer asserted triples and/or more wrong assertions. Therefore, the majority of the true positives are due to the triples already belonging to the ABoxes before the materialization. The best thresholds for the experiments are 0.0853 (with input triples) and 0.0893 (without input triples). Table 5.5 reports the values for the accuracy, precision, and recall (with and without the input triples) for the three thresholds. The results highlight how the nature of the embedding impacts the evaluation of the metrics. The high values of accuracy for the materialization task with the SPO do not reflect the quality of the inference, which is instead explained by the precision and the recall.

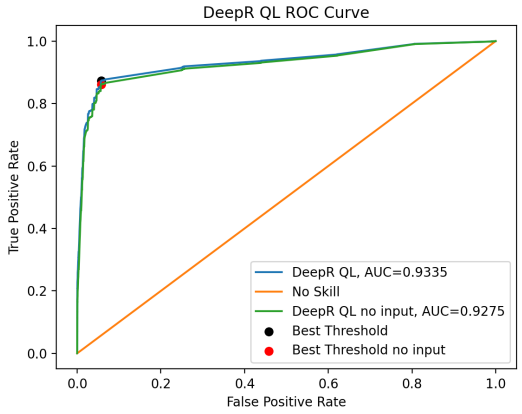


Figure 5.7: DeepR QL ROC curve.

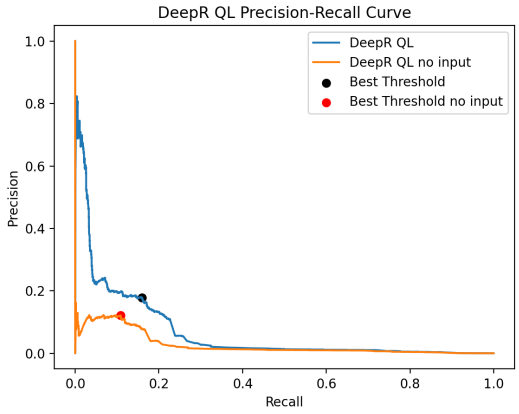


Figure 5.8: DeepR QL PR curve.

Threshold	AI	A	PI	P	RI	R
0.000144	0.94236	0.94236	0.00386	0.00349	0.87361	0.86199
0.0893	0.99961	0.99960	0.17999	0.12148	0.15806	0.10872
0.0853	0.99961	0.99960	0.17776	0.11855	0.16006	0.10872

Table 5.5: Thresholds comparison of DeepR QL accuracy, precision, and recall. (A) : accuracy, (P) : precision, (R) : recall, (I) : with input triples.

5.3.2. DeepR EL

Given the parameters in Table 5.3, the final architecture of *DeepR* for the EL profile is shown in Figure 5.9. The architecture of *DeepR EL* is more complex then *DeepR QL*. A possible interpretation regarding the model's architecture could rely on the complexity of reasoning with the EL profile, which in fact is higher with respect to QL .Table 5.6 reports

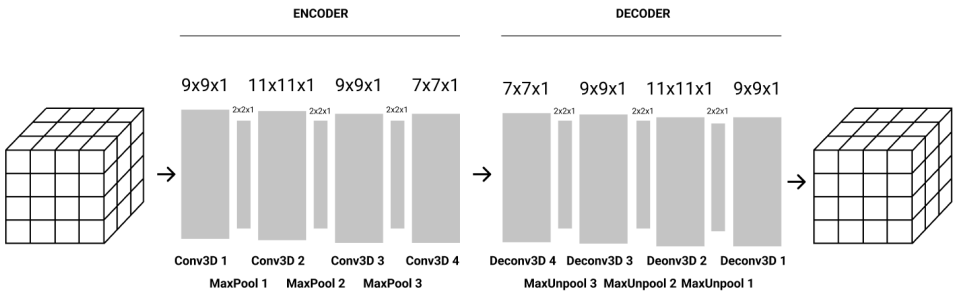


Figure 5.9: Time comparison between DeepR EL and HermiT.

the time performances of the model in its different phases (scene parsing, embedding, training and testing) and the time performances of HermiT on the entire dataset.

Scene parsing	SPO embedding	DeepR Training	Total	Total/scene	DeepR testing	Testing/scene	HermiT	HermiT/scene
4402.6	92.2	912505.2	917000	13.1	14340.51	0.956	211933	2.12

Table 5.6: Computing time of DeepR EL and HermiT.

As for the QL profile, *DeepR* is slower than HemiT during the training phase and outperforms it in the testing phase. However, in this case, the performances on the testing set are close, while the difference during the training phase is much greater. The reasons rely on a lower HermiT's computing time and on the more complex model. Figure 5.10 shows a comparison between the HermiT average reasoning time per scene and *DeepR EL* average training (limited to the last epoch) and testing time per scene. The model reaches an average speedup of 6x with respect to the training and a speedup of 2x with respect to HetmiT.

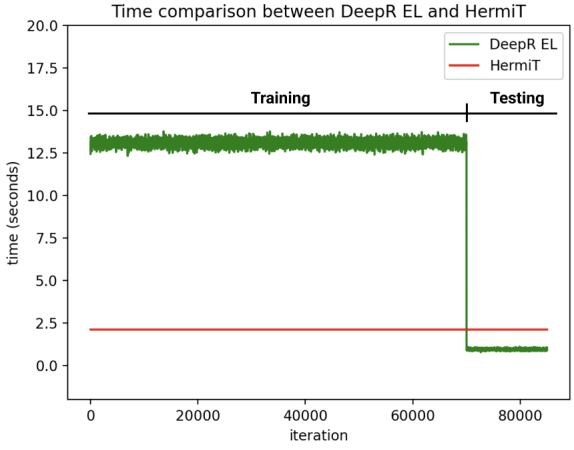


Figure 5.10: Time comparison between DeepR EL and HermiT.

As for the experiments with the QL profile, *DeepR EL* does not perform well for what concerns the approximation quality. Figure 5.11 shows the ROC curves derived from the predictions made by the model on the test set. The blue curve refers to the standard predictions, while the green one is derived by removing the triples belonging to the input from the predictions and the targets. Also in this case, there is a similar degree of separability in both experiments (the ROCs are close to each other), and the values of the AUC suggest good overall performances. However, the best threshold is lower than the one for QL: 0.00001. Indeed, the architecture of *DeepR EL* is more complex and the weights' tendency toward zero is greater. The overall accuracy reached by *DeepR EL* with the best threshold is 97.180%. The poorer performance of the model on the EL profile can also be seen from the PR curves for both experiments, which are reported in Figure 5.12. Once again, despite the high accuracy, the model performs a poor inference in terms of both correctly predicted triples, which is pointed by the low precision, and the number of predicted triples, which is pointed by the low recall. The smaller difference between the two PR curves, however, highlights that the model even misspredict the triples belonging to the input, removing them from its output. The best thresholds for the experiments are 0.105182 (with input triples) and 0.081462 (without input triples). The resulting thresholds do not differ much from those of *DeepR QL*, in fact, the lower recall and lower precision are due to the higher true negatives. Table 5.7 reports the values for the accuracy, precision, and recall (with and without the input triples) for the three thresholds. The results highlight how the architecture of the model and the true negatives impact even more the evaluation of the metrics as compared to the experiments with the QL profile.

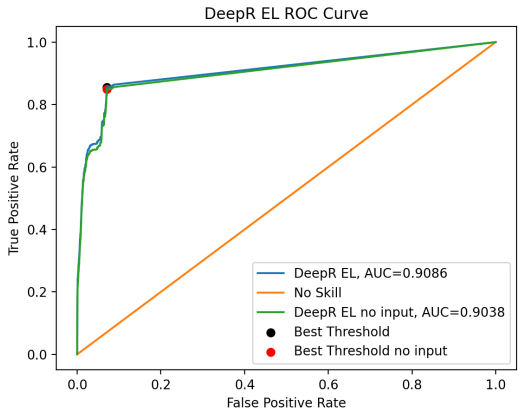


Figure 5.11: DeepR EL ROC curve.

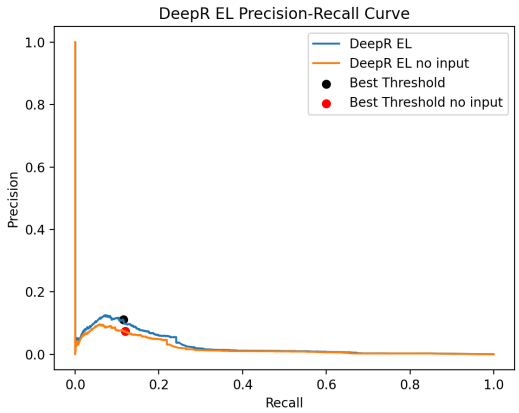


Figure 5.12: DeepR EL PR curve.

Threshold	AI	A	PI	P	RI	R
0.00001	0.97180	0.97181	0.00584	0.00519	0.65978	0.64189
0.105182	0.99955	0.99957	0.10966	0.08532	0.11183	0.09276
0.081462	0.99944	0.99946	0.09127	0.07341	0.13691	0.11829

Table 5.7: Thresholds comparison of DeepR EL accuracy, precision, and recall. (A) : accuracy, (P) : precision, (R) : recall, (I) : with input triples.

5.3.3. DeepR RL

For reasons of time, the training and testing on the RL profile were made on a limited portion of the dataset (10000 scenes). Given the parameters in Table 5.3, the final architecture of *DeepR* for the RL profile is shown in Figure 5.13.

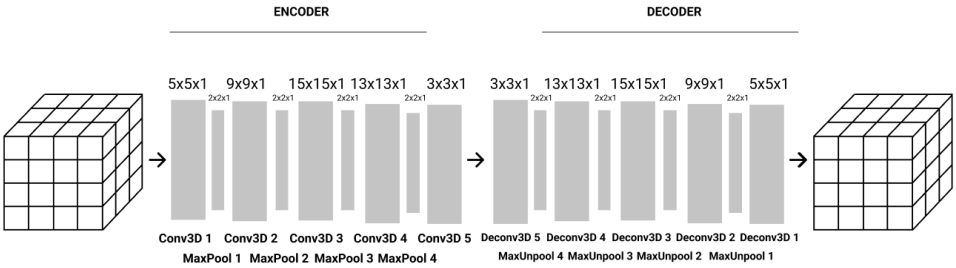


Figure 5.13: DeepR RL architecture.

The architecture of *DeepR RL* is a bit more complex than that of *DeepR EL*. The reason could rely on the greater dimension of the scenes. Table 5.8 reports the time performances of the model in its different phases (scene parsing, embedding, training and testing) and the time performances of HermiT on the entire dataset.

Scene parsing	SPO embedding	DeepR Training	Total	Total/scene	DeepR testing	Testing/scene	HermiT	HermiT/scene
5563.6	6438.1	193370	205371.5	29.1	3210.1	1.07	4764269	47.642

Table 5.8: Computing time of DeepR RL and HermiT.

With the RL profile, *DeepR* outperforms HemiT both on the training and testing phase. Figure 5.14 shows a comparison between the HermiT average reasoning time per scene and *DeepR RL* average training (limited to the last epoch) and testing time per scene. In average, the model reaches a 27x speedup of with respect to the training and 45x with respect to HetmiT.

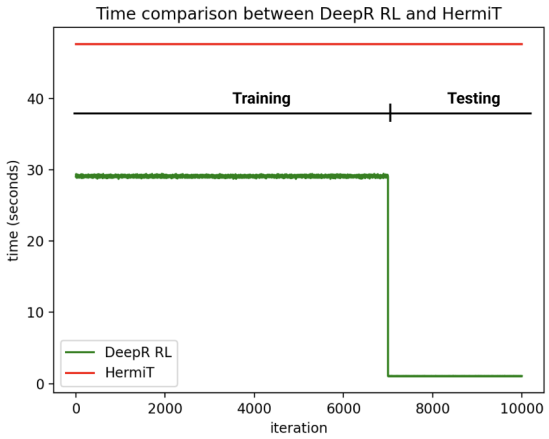


Figure 5.14: Time comparison between DeepR RL and HermiT.

DeepR RL, too, does not perform well in terms of the approximation quality. Figure 5.15 shows the ROC curves derived from the predictions made by the model on the test set. The blue curve refers to the standard predictions, while the green one is derived by removing the triples belonging to the input from the predictions and the targets. The ROCs almost coincide, and the values of the AUC still suggest good overall performances. The best threshold derived from the ROCs is 0.000165, and the curves are less steep, which translates into a higher FPR and a lover TPR. Indeed, the KGs of the RL profile have a higher number of triples, which results in more 1s in the SPO embedding. For this reason, the tendency of the model's weights toward zero could be lower. The overall accuracy reached by *DeepR RL* with the best threshold is 91.121%. The higher threshold, indeed, causes more mispredictions. Figure 5.16 shows the PR curves derived from the model's predictions on the test set. Once again, despite the high accuracy, the model performs a poor inference in terms of both correctly predicted triples, which is pointed by the low precision, and the number of predicted triples, which is pointed by the low recall. The PR curves are even closer than the ones of *DeepR EL*, highlighting more mispredictions on the triples belonging to the input. The best thresholds for the experiments are 0.029293 (with input triples) and 0.028836 (without input triples). Table 5.9 reports the values for the accuracy, precision, and recall (with and without the input triples) for the three thresholds. As for the other experiments, the accuracy does not reflect the quality of the approximation.

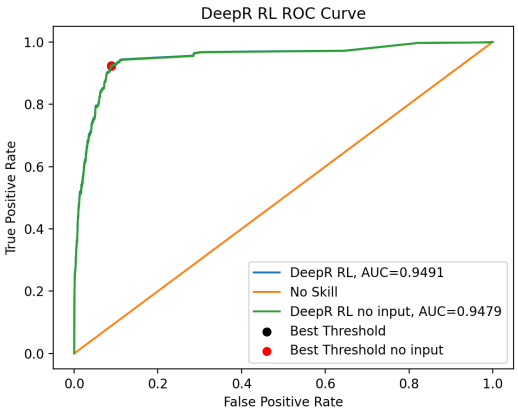


Figure 5.15: DeepR RL ROC curve.

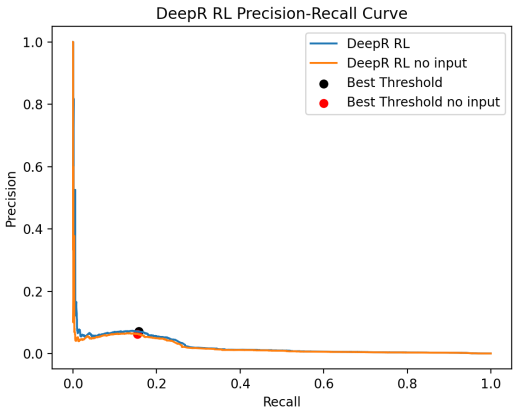


Figure 5.16: DeepR RL PR curve.

Threshold	AI	A	PI	P	RI	R
0.000165	0.91121	0.91121	0.00253	0.00242	0.92435	0.92243
0.029293	0.99929	0.99930	0.07117	0.06418	0.15715	0.14723
0.028836	0.99926	0.99927	0.06979	0.06313	0.16369	0.15389

Table 5.9: Thresholds comparison of DeepR RL accuracy, precision, and recall. (A) : accuracy, (P) : precision, (R) : recall, (I) : with input triples.

6 | Conclusion and Future Work

In this thesis, we presented our benchmark to test the approximation of the Knowledge Base materialization task, together with a baseline model implemented with a Variational Autoencoder.
In Chapter 3, we presented the challenges related to the trade-off between the expressiveness of a Description Logics and the complexity of the reasoning. Using the Macro-Meso-Micro framework, we introduced the research question *is it possible to approximate deductive reasoning using inductive reasoning?* This question helps us to analyze the the approaches existing in the research community and to narrow the scope towards the Deep Learning field. This led us to a more specific question: *Is it possible to approximate the materialization task with a Variational Autoencoder?* This question has brought us to the definition of some requirements regarding the data encoding. Then, we narrowed the investigation even further to identify a feasible problem to solve. We found out that there is no benchmark that takes into consideration both the knowledge representation and the deep learning effort. The final research question *How can we design a benchmark to evaluate the task of approximate materialization?* raised the design problem: design a benchmark based on CLEVR to evaluate the approximation of the materialization task and test it on a baseline model that satisfies the requirements.

In Chapter 4, we addressed the design problem by providing an Knowledge Graph called *KlevR* and a baseline model implemented with a simplified VAE called *DeepR*. We started by describing our process of designing the ontology starting from CLEVR. Then we focused on the embedding model for the *KlevR scene graphs*. Finally, we discussed our approach on the design of the VAE.
In Chapter 5 we presented the implementation experience of our research with the data pipeline and tools we have used. Moreover, we presented the performances of our baseline model on the benchmark we have defined.

6.1. Lessons Learned

In light of the results obtained from our experiments, we can draw some conclusions regarding the approximate reasoning task and the benchmark we proposed.

Although the metrics we propose are not exhaustive, *KlevR* proves how a trivial model based on a VAE is not able to successfully perform the approximation of the materialization. This result manifests the importance of the benchmark for the task, which lays the foundation for future assessments. The quality of the benchmark lies on the scalability of the dataset and on the reasoning effort proved by the ground-truth given by HermiT. However, the benchmark implementation is not exempt from improvements. The main challenges we have encountered in our experiments rely on the portability and usability of the data. In particular, the data pipeline we have used to compute the ground truth for the materialization task is computationally intensive and time-consuming. To ensure the usability of the benchmark for future assessment, the pipeline needs to be optimized and made more flexible and scalable.

The poor results obtained from the baseline proposed by us proves the complexity of the task. However, we can draw conclusions that reflect the results of our approach. In particular, the SPO embedding is not the best choice for encoding the Knowledge graphs. The sparse nature of SPO forces the network to very low thresholds, increasing the risk of false positives. Also, classic Deep Learning metrics can be misleading due to the high presence of true negatives. The accuracy of *DeepR*, in fact, is excellent on the test set, but it does not reflect the accuracy of the task objective, which instead resides in the true positives/negatives and the precision. For what concerns the *DeepR* models, the performances in terms of computing time reflect the effort of reasoning for the corresponding OWL profile during the training phase. However, this is not the case in the testing phase. The obtained approximations appear to gain speedups of the same order of magnitude for the three models.

6.2. Limitation and Future Works

For reasons of time, we did not have the opportunity to perform other complex experiments. An important extension that represents a future work consists of testing different existing embedding models on *KlevR* and verifying how different representations deals with our benchmark. To solve the problem of encoding the KGs, the design of an embedding model that takes into consideration the semantics of the ontology should be tackled. Moreover, other experiments could focus on the scalability property of *KlevR*. Thus, a

future work will be to provide a complete and detailed evaluation including extension on the ontologies (see Section 6.2.1) and different experiments (see Section 6.2.3). Moreover, future investigations include the design of a more efficient baseline model for the approximate materialization task, with the extensions explained in Section 6.2.2.

6.2.1. Ontologies Extension

Thanks to the combinatorial nature of the CLEVR attributes, the class hierarchies of *KlevR* ontologies are very spread. However, the ontologies could be further extended. In building the hierarchies we followed the alphabetic order of the attribute types for the class names, but other arrangements could be also used, together with the "owl:equivalentClass" axiom. Figure 6.1 shows an example for the class *BlueMetallicSmallCubeObject*. Considering n as the number of attibutes in a class name (n depends on the level in the hierachies), all the possible combinations are $n!$.

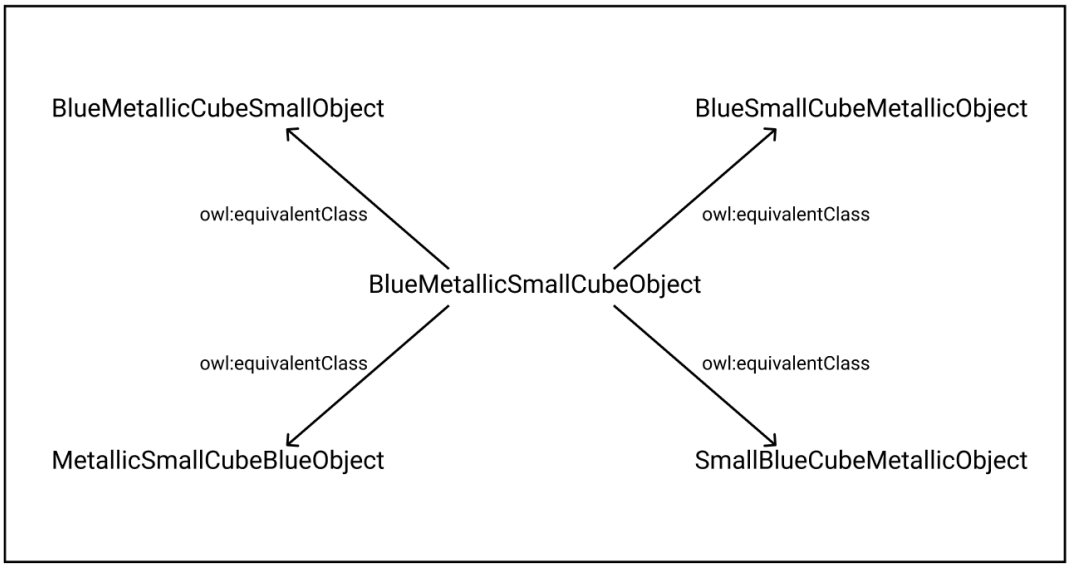


Figure 6.1: Example of a extended KlevR classes.

6.2.2. DeepR Extension

Our implementation of *DeepR* does not follow the standard Variational Autoencoder implementation. In particular, we do not have used the loss function descripted in Section 2.3.3. A future extension of *DeepR* could include the loss function with both the reconstruction and regularization terms. In particular, it would be interesting to train the encoder on embedding the input KG and the decoder to materialize it. To reach this goal, the reconstruction term of the loss function could take into account the difference

between the input and the output KGs. In this way, it is easier to derive if the errors made by the model rely on the encoder or on the decoder part of the architecture.

6.2.3. Further Investigations

Thanks to the synthetic nature of *KlevR*, more experiments can be planned to test the approximate materialization task. For example, the use of "negative triples", as Nichel et Al.[16] discuss, could be implemented to test if the model recognizes and removes the incorrect triples.

Another interesting investigation relies on the *DeepR* "abilities" to learn specific entailment rules. With *KlevR*, it is possible to design specific KGs, that force the reasoner to use only a specific rule. For example, if the model receives as input ABoxes which target implies reasoning only on transitivity, the accuracy on that specific entailment rule can be computed.

Bibliography

[1] A. Bordes, N. Usunier, A. García-Durán, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 2787–2795, 2013. URL `https://proceedings.neurips.cc/paper/2013/hash/1cecc7a77928ca8133fa24680a88d2f9-Abstract.html`.

[2] L. Costabello. *Context-Aware Access Control and Presentation of Linked Data*. PhD thesis, 11 2013.

[3] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel. Convolutional 2d knowledge graph embeddings. In S. A. McIlraith and K. Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1811–1818. AAAI Press, 2018. URL `https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17366`.

[4] H. He. The state of machine learning frameworks in 2019. *The Gradient*, 2019.

[5] A. Hidaka and T. Kurita. Consecutive dimensionality reduction by canonical correlation analysis for visualization of convolutional neural networks. volume 2017, pages 160–167, 12 2017. doi: 10.5687/sss.2017.160.

[6] M. M. Hussain, V. Raju, J. Kandasamy, and D. Govardhan. Prediction of tensile and shear strength of friction surfaced tool steel deposit by using artificial neural networks. *IOP Conference Series: Materials Science and Engineering*, 346:012086, 04 2018. doi: 10.1088/1757-899X/346/1/012086.

[7] G. Ji, S. He, L. Xu, K. Liu, and J. Zhao. Knowledge graph embedding via dynamic mapping matrix. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural*

Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers, pages 687–696. The Association for Computer Linguistics, 2015. doi: 10.3115/v1/p15-1067. URL https://doi.org/10.3115/v1/p15-1067.

[8] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, and R. B. Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. *CoRR*, abs/1612.06890, 2016. URL http://arxiv.org/abs/1612.06890.

[9] D. P. Kingma and M. Welling. An introduction to variational autoencoders. *Found. Trends Mach. Learn.*, 12(4):307–392, 2019. doi: 10.1561/2200000056. URL https://doi.org/10.1561/2200000056.

[10] U. Kursuncu, M. Gaur, and A. P. Sheth. Knowledge infused learning (K-IL): towards deep incorporation of knowledge in deep learning. *CoRR*, abs/1912.00512, 2019. URL http://arxiv.org/abs/1912.00512.

[11] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu. Learning entity and relation embeddings for knowledge graph completion. In B. Bonet and S. Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 2181–2187. AAAI Press, 2015. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9571.

[12] J. Mao, C. Gan, P. Kohli, J. B. Tenenbaum, and J. Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=rJgMlhRctm.

[13] D. Nathani, J. Chauhan, C. Sharma, and M. Kaul. Learning attention-based embeddings for relation prediction in knowledge graphs. *CoRR*, abs/1906.01195, 2019. URL http://arxiv.org/abs/1906.01195.

[14] D. Q. Nguyen, T. D. Nguyen, D. Q. Nguyen, and D. Q. Phung. A novel embedding model for knowledge base completion based on convolutional neural network. In M. A. Walker, H. Ji, and A. Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)*, pages 327–333. Association for Computational

Linguistics, 2018. doi: 10.18653/v1/n18-2053. URL https://doi.org/10.18653/v1/n18-2053.

[15] M. Nickel, V. Tresp, and H. Kriegel. A three-way model for collective learning on multi-relational data. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 809–816. Omnipress, 2011. URL https://icml.cc/2011/papers/438_icmlpaper.pdf.

[16] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs. *Proc. IEEE*, 104(1):11–33, 2016. doi: 10.1109/JPROC.2015.2483592. URL https://doi.org/10.1109/JPROC.2015.2483592.

[17] H. Paulheim and H. Stuckenschmidt. Fast approximate a-box consistency checking using machine learning. In H. Sack, E. Blomqvist, M. d'Aquin, C. Ghidini, S. P. Ponzetto, and C. Lange, editors, *The Semantic Web. Latest Advances and New Domains - 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29 - June 2, 2016, Proceedings*, volume 9678 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 2016. doi: 10.1007/978-3-319-34129-3_9. URL https://doi.org/10.1007/978-3-319-34129-3_9.

[18] rmlio. Rml mapping language, 2020. URL https://rml.io/specs/rml/.

[19] S. Serpa and C. Ferreira. Micro, meso and macro levels of social analysis. *International Journal of Social Science Studies*, 7(3):120–124, 2019. ISSN 2324-8041. doi: 10.11114/ijsss.v7i3.4223. URL https://www.redfame.com/journal/index.php/ijsss/article/view/4223.

[20] G. Singh, S. Bhatia, and R. Mutharaju. Owl2bench: A benchmark for OWL 2 reasoners. In J. Z. Pan, V. A. M. Tamma, C. d'Amato, K. Janowicz, B. Fu, A. Polleres, O. Seneviratne, and L. Kagal, editors, *The Semantic Web - ISWC 2020 - 19th International Semantic Web Conference, Athens, Greece, November 2-6, 2020, Proceedings, Part II*, volume 12507 of *Lecture Notes in Computer Science*, pages 81–96. Springer, 2020. doi: 10.1007/978-3-030-62466-8_6. URL https://doi.org/10.1007/978-3-030-62466-8_6.

[21] R. Socher, D. Chen, C. D. Manning, and A. Y. Ng. Reasoning with neural tensor networks for knowledge base completion. In C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*,

pages 926–934, 2013. URL https://proceedings.neurips.cc/paper/2013/hash/b337e84de8752b27eda3a12363109e80-Abstract.html.

[22] J. Sublime and E. Kalinicheva. Automatic post-disaster damage mapping using deep-learning techniques for change detection: Case study of the tohoku tsunami. *Remote. Sens.*, 11(9):1123, 2019. doi: 10.3390/rs11091123. URL https://doi.org/10.3390/rs11091123.

[23] P. Tiwari, H. Zhu, and H. M. Pandey. Dapath: Distance-aware knowledge graph reasoning based on deep reinforcement learning. *Neural Networks*, 135:1–12, 2021. doi: 10.1016/j.neunet.2020.11.012. URL https://doi.org/10.1016/j.neunet.2020.11.012.

[24] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

[25] W3C. Web ontology language (owl), 2013. URL https://www.w3.org/OWL/.

[26] W3C. Rdf 1.1 concepts and abstract syntax, 2014. URL https://www.w3.org/TR/rdf11-concepts/.

[27] Q. Wang, Y. Ji, Y. Hao, and J. Cao. GRL: knowledge graph completion with gan-based reinforcement learning. *Knowl. Based Syst.*, 209:106421, 2020. doi: 10.1016/j.knosys.2020.106421. URL https://doi.org/10.1016/j.knosys.2020.106421.

[28] Z. Wang, J. Zhang, J. Feng, and Z. Chen. Knowledge graph embedding by translating on hyperplanes. In C. E. Brodley and P. Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, pages 1112–1119. AAAI Press, 2014. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8531.

A | Appendix A

This appendix describes the entailment regimes of the OWL profiles. In particular, Table A.1 lists the OWL axioms supported by each profile. The axioms labelled with "RC" are restricted to class expressions, while the ones labelled with "RSub" are restricted to subclasses.

Axiom / expression	**EL**	**QL**	**RL**
ObjectSomeValuesFrom	RC	RSub	RSub
DataSomeValuesFrom	RC	RSub	RSub
ObjectIntersectionOf	RC	v	RSub
ObjectComplementOf	x	v	RC
SubClassOf	v	v	v
EquivalentClasses	v	v	v
DisjointClasses	v	v	v
InverseObjectProperties	x	v	v
SubObjectPropertyOf	v	v	v
SubDataPropertyOf	v	v	v
EquivalentObjectProperties	v	v	v
EquivalentDataProperties	v	v	v
ObjectPropertyDomain	v	v	v
DataPropertyDomain	v	v	v
ObjectPropertyRange	v	v	v
DataPropertyRange	v	v	v
DisjointObjectProperties	x	v	v
DisjointDataProperties	x	v	v
SymmetricObjectProperty	x	v	v
ReflexiveObjectProperty	v	v	x
IrreflexiveObjectProperty	x	v	v
AsymmetricObjectProperty	x	v	v

DifferentIndividuals	v	v	v
ClassAssertion	v	v	v
ObjectPropertyAssertion	v	v	v
DataPropertyAssertion	v	v	v
ObjectOneOf	RC	x	RSub
ObjectUnionOf	x	x	RSub
ObjectHasValue	v	x	RSub
DataHasValue	v	x	RSub
ObjectAllValuesFrom	x	x	RC
ObjectMaxCardinality 0/1	x	x	RC
DataAllValuesFrom	x	x	RSub
DataMaxCardinality 0/1	x	x	RC
ObjectHasSelf	RC	x	v
DataOneOf	v	x	RSub
DataIntersectionOf	v	x	v
TransitiveObjectProperty	v	x	v
SameIndividual	v	x	v
NegativeObjectPropertyAssertion	v	x	v
NegativeDataPropertyAssertion	v	x	v
FunctionalDataProperty	x	x	v
HasKey	v	x	v
DisjointUnion	x	x	x
ObjecComplementOf	x	RC	RC
ObjecIntersectionOf	v	RC	RC

Table A.1: OWL profiles entailment regimes.

B | Appendix B

This appendix lists the SPARQL query obtained translating the CLEVR questions (see Section 4.1.1). In particular, we reports a query for each profile of *KlevR* and for every question type. (see Section 4.1 for explanations).

- *How many <Z> <C> <M> <S> are there?*, *What number of <Z> <C> <M> <S> are there?*
  Listings B.1, B.2, B.3 shows the query for the RL, QL and EL profile respectively.
- *Are there any <Z> <C> <M> <S>?*, *Are any <Z> <C> <M> <S> visible?*, *Is there a <Z> <C> <M> <S>?*
  Listings B.4, B.5, B.6 shows the query for the RL, QL and EL profile respectively.
- *What shape is the <Z> <C> <M>?* Listings B.7, B.8, B.9 shows the query for the RL, QL and EL profile respectively. The queries for the color, size and material are the same, except for the placeholders.
- *Is there fewer <Z> <C> <M> <S> than <Z1> <C1> <M1> <S1>?*
  Listings B.10, B.11, B.12 shows the query for the RL, QL and EL profile respectively. The queries for the "grater" and "equal" are the same, except for the comparison symbols.
- *Is the <C> <M> <S> the same size of the <C1> <M1> <S1>?*
  Listings B.13, B.14, B.15 shows the query for the RL, QL and EL profile respectively. The queries for the color, material and shape are the same, except for the class names.

```
SELECT (COUNT(?x) as ?xCount)
WHERE
   {
         ?x ex:hasColor ?c;
            ex:hasSize ?z;
            ex:hasMaterial ?m;
            ex:hasShape ?s.

         ?c  rdf:type ex:<C>.
         ?z  rdf:type ex:<Z>.
         ?m  rdf:type ex:<M>.
         ?s  rdf:type ex:<S>.

    FILTER(?s <> ?c) FILTER(?s <> ?z)
    FILTER(?s <> ?m) FILTER(?c <> ?z)
    FILTER(?c <> ?m) FILTER(?z <> ?m)
    }
```

Listing B.1: RL SPARQL query for the question "How many <Z><C><M><S> are there?".

```
SELECT (COUNT(?x) as ?xCount)
WHERE
    {
     ?x rdf:type ex:<C>Object;
        rdf:type ex:<Z>Object;
        rdf:type ex:<S>Object;
        rdf:type ex:<M>Object.
    }
```

Listing B.2: QL SPARQL query for the question "How many <Z><C><M><S> are there?".

```
SELECT (COUNT(?x) as ?xCount)
WHERE
    {
     ?x rdf:type ex:<C>Object;
        rdf:type ex:<Z>Object;
        rdf:type ex:<S>Object;
        rdf:type ex:<M>Object.
    }
```

Listing B.3: EL SPARQL query for the question "How many <Z><C><M><S> are there?".

```
ASK
WHERE
{
      ?x ex:hasColor ?c;
         ex:hasSize ?z;
         ex:hasMaterial ?m;
         ex:hasShape ?s.

      ?c rdf:type ex:<C>.
      ?z rdf:type ex:<Z>.
      ?m rdf:type ex:<M>.
      ?s rdf:type ex:<S>.

FILTER(?s <> ?c)
FILTER(?s <> ?z)
FILTER(?s <> ?m)
FILTER(?c <> ?z)
FILTER(?c <> ?m)
FILTER(?z <> ?m)
    }
```

Listing B.4: RL SPARQL query for the question "Are there any <Z> <C> <M> <S>?".

```
ASK
WHERE
    {
     ?x rdf:type ex:<C>Object;
        rdf:type ex:<Z>Object;
        rdf:type ex:<S>Object;
        rdf:type ex:<M>Object;
    }
```

Listing B.5: QL SPARQL query for the question "Are there any <Z> <C> <M> <S>?".

```
ASK
WHERE
    {
     ?x rdf:type ex:<C>Object;
        rdf:type ex:<Z>Object;
        rdf:type ex:<S>Object;
        rdf:type ex:<M>Object;
    }
```

Listing B.6: EL SPARQL query for the question "Are there any <Z> <C> <M> <S>?".

```
SELECT  ?s ?shapeClass
WHERE
{
     ?x ex:hasColor ?c ;
        ex:hasSize ?z;
        ex:hasMaterial ?m;
        ex:hasShape ?s.

     ?c rdf:type ex:<C>.
     ?z rdf:type ex:<Z>.
     ?m rdf:type ex:<M>.
     ?s rdf:type ?shapeClass.

     ?shapeClass rdf:subClassOf ex:Shape.

FILTER(?s <> ?c)
FILTER(?s <> ?z)
FILTER(?s <> ?m)
FILTER(?c <> ?z)
FILTER(?c <> ?m)
FILTER(?z <> ?m)
    }
```

Listing B.7: RL SPARQL query for the question "What shape is the <Z> <C> <M>?".

```
SELECT ?shapedObject
WHERE
{
      ?x rdf:type ex:<Z><C><M>Object;
         rdf:type ?shapedObject.
      ?shapedObject rdf:subClassOf ex:ShapedObject.
}
```

Listing B.8: QL SPARQL query for the question "What shape is the <Z> <C> <M>?".

```
SELECT ?shapedObject
WHERE
{
      ?x rdf:type ex:<Z><C><M>Object;
         rdf:type ?shapedObject.
      ?shapedObject rdf:subClassOf ex:ShapedObject.
}
```

Listing B.9: EL SPARQL query for the question "What shape is the <Z> <C> <M>?".

```
SELECT ?result
WHERE
{
  {
  SELECT (COUNT(?x) as ?xCount)
    WHERE
    {
          ?x ex:hasColor ?c;
             ex:hasSize ?z;
             ex:hasMaterial ?m;
             ex:hasShape ?s.

          ?c  rdf:type ex:<C>.
          ?z  rdf:type ex:<Z>.
          ?m  rdf:type ex:<M>.
          ?s  rdf:type ex:<S>.

    FILTER(?s <> ?c)      FILTER(?s <> ?z)
    FILTER(?s <> ?m)      FILTER(?c <> ?z)
    FILTER(?c <> ?m)      FILTER(?z <> ?m)
        }
  }
  {
  SELECT (COUNT(?y) as ?yCount)
    WHERE
    {
          ?y ex:hasColor ?c1;
             ex:hasSize ?z1;
             ex:hasMaterial ?m1;
             ex:hasShape ?s1.

          ?c1  rdf:type ex:<C1>.
          ?z1  rdf:type ex:<Z1>.
          ?m1  rdf:type ex:<M1>.
          ?s1  rdf:type ex:<S1>.

    FILTER(?s1 <> ?c1)      FILTER(?s1 <> ?z1)
    FILTER(?s1 <> ?m1)      FILTER(?c1 <> ?z1)
    FILTER(?c1 <> ?m1)      FILTER(?z1 <> ?m1)
        }
  }
  BIND ( IF ( ?xCount < ?yCount , "yes" , "no" ) AS ?result )
}
```

Listing B.10: RL SPARQL query for the question "Is there fewer <Z> <C> <M> <S> than <Z1> <C1> <M1> <S1>?".

```
SELECT ?result
WHERE
{
  {
  SELECT (COUNT(?x) as ?xCount
    WHERE
        {
         ?x rdf:type ex:<C>Object;
            rdf:type ex:<Z>Object;
            rdf:type ex:<S>Object;
            rdf:type ex:<M>Object.
      }
  }
  {
  SELECT (COUNT(?y) as ?yCount)
    WHERE
        {
         ?y rdf:type ex:<C1>Object;
            rdf:type ex:<Z1>Object;
            rdf:type ex:<S1>Object;
            rdf:type ex:<M1>Object.
      }
  }
  BIND ( IF ( ?xCount < ?yCount , "yes" , "no" ) AS ?result )
}
```

Listing B.11: QL SPARQL query for the question "Is there fewer <Z> <C> <M> <S> than <Z1> <C1> <M1> <S1>?".

```
SELECT ?result
WHERE
{
  {
  SELECT (COUNT(?x) as ?xCount
    WHERE
        {
         ?x rdf:type ex:<C>Object;
            rdf:type ex:<Z>Object;
            rdf:type ex:<S>Object;
            rdf:type ex:<M>Object.
      }
  }
  {
  SELECT (COUNT(?y) as ?yCount)
    WHERE
        {
         ?y rdf:type ex:<C1>Object;
            rdf:type ex:<Z1>Object;
            rdf:type ex:<S1>Object;
            rdf:type ex:<M1>Object.
      }
  }
  BIND ( IF ( ?xCount < ?yCount , "yes" , "no" ) AS ?result )
}
```

Listing B.12: EL SPARQL query for the question "Is there fewer <Z> <C> <M> <S> than <Z1> <C1> <M1> <S1>?".

```
ASK
WHERE
{
      ?x ex:hasColor ?c;
         ex:hasSize ?z;
         ex:hasMaterial ?m;
         ex:hasShape ?s.

      ?c rdf:type ex:<C>.
      ?m rdf:type ex:<M>.
      ?s rdf:type ex:<S>.

      ?y ex:hasColor ?c1;
         ex:hasSize ?z;
         ex:hasMaterial ?m1;
         ex:hasShape ?s1.

      ?c1 rdf:type ex:<C1>.
      ?m1 rdf:type ex:<M1>.
      ?s1 rdf:type ex:<S1>.

      ?z rdf:type ?sizeClass.

      ?sizeClass rdf:subClassOf ex:Size.


FILTER(?s <> ?c) FILTER(?s <> ?z)
FILTER(?s <> ?m) FILTER(?c <> ?z)
FILTER(?c <> ?m) FILTER(?z <> ?m)
    }
```

Listing B.13: RL SPARQL query for the question "Is the <C> <M> <S> the same size of the <C1> <M1> <S1>?".

```
ASK
WHERE
    {
     ?x rdf:type ex:<C>Object;
        rdf:type ex:?z;
        rdf:type ex:<S>Object;
        rdf:type ex:<M>Object;

     ?y rdf:type ex:<C1>Object;
        rdf:type ex:?z;
        rdf:type ex:<S1>Object;
        rdf:type ex:<M1>Object;
    }
```

Listing B.14: QL SPARQL query for the question "Is the <C> <M> <S> the same size of the <C1> <M1> <S1>?".

```
ASK
WHERE
    {
     ?x rdf:type ex:<C>Object;
        rdf:type ex:?z;
        rdf:type ex:<S>Object;
        rdf:type ex:<M>Object;

     ?y rdf:type ex:<C1>Object;
        rdf:type ex:?z;
        rdf:type ex:<S1>Object;
        rdf:type ex:<M1>Object;
    }
```

Listing B.15: EL SPARQL query for the question "Is the <C> <M> <S> the same size of the <C1> <M1> <S1>?".

Acknowledgements

Verona, 30 Marzo 2022

Ringrazio il Professor Emanuele Della Valle per avermi concesso l'opportunità di intraprendere questo percorso di tesi. Ringrazio Riccardo Tommasini, che mi ha seguito in questo cammino, trasmettendomi la passione per la ricerca e supportandomi nella mia crescita personale. Grazie soprattutto per il supporto che mi hai dato nell'ultima fase del percorso e per la fiducia riposta in me. Ringrazio la mia famiglia per avermi dato la possibilità di intraprendere questa strada, sostenendomi sempre. Fin da piccolo mi avete dato la possibilità di crescere con esperienze formative e stimolanti, che mi hanno permesso di formarmi e di diventare quello che sono ora. Ringrazio tutte le persone che mi hanno sostenuto in questi lunghi e impegnativi anni. In particolare ringrazio Michele, che mi accompagna dai tempi delle scuole medie ed è stato un punto di riferimento e di confronto durante tutta la mia vita scolastica. Ringrazio il "socio", Riccardo. Soprattutto negli ultimi tempi sei stato l'unico a conoscere tutte le difficoltà che ho incontrato e hai sempre cercato di sostenermi con la tua presenza. Ringrazio Stefano e Matteo, gli amici con cui ho intrapreso esperienze stimolanti e che ci hanno legati ulteriormente. Ringrazio e auguro buona fortuna ai miei più cari colleghi di corso Davide e Francesco, con cui ho vissuto e condiviso le sfide e la vita universitaria qui al Politecnico di Milano. Un grazie particolare va a Enrico, il fantastico barista del "White ES", che ha camminato con me in questo sentiero, a partire dal primo esame fino al raggiungimento del traguardo di oggi.

Amedeo.