University of Bergen
Department of Informatics

Combining Query Rewriting and Knowledge Graph Embeddings for Complex Query Answering

Author: Anders Imenes
Supervisors: Ana Ozaki, Ricardo Guimarães

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

May, 2023

Abstract

The field of complex query answering using Knowledge Graphs (KGs) has seen substantial advancements in recent years, primarily through the utilization of Knowledge Graph Embeddings (KGEs). However, these methodologies often stumble when faced with intricate query structures that involve multiple entities and relationships. This thesis primarily investigates the potential of integrating query rewriting techniques into the KGE query answering process to improve performance in such situations. Guided by a TBox, a schema that describes the concepts and relationships in the data from Description Logics, query rewriting translates a query into a *union of rewritten queries* that can potentially widen the prediction scope for KGEs. The thesis uses the PerfectRef algorithm [14] for facilitating query rewriting, aiming to maximize the scope of query response and enhance prediction capabilities.

Two distinct datasets were employed in the study: The Family Dataset, a subset of Wikidata, and DBPedia15k, a subset of DBPedia. The effectiveness of the proposed methodology was evaluated against these datasets using different KGE models, in our case TransE, DistMult, BoxE, RotatE, and CompGCN. The results demonstrate a notable improvement in complex query answering when query rewriting is used for both The Family dataset and DBPedia15k. Furthermore, the amalgamation of query rewriting and KGE predictions yielded a performance boost for The Family dataset. However, the same was not observed for DBPedia15k, likely due to discrepancies and errors present within DBPedia15k compared to the Full DBPedia KG used for validation in our framework.

This research suggests that query rewriting, as a pre-processing step for KGE prediction, can enhance the performance of complex query answering, mainly when the dataset is not fully entailed. This study provides important insights into the potential and limitations of integrating query rewriting with KGEs. It may serve as a guidepost for future research to improve the complex query answering when a TBox is available.

Acknowledgements

I am deeply grateful to my supervisors, Ana and Ricardo, who have provided their expertise and guidance throughout this journey. Ana, who conceived the idea of introducing query rewriting, has been an invaluable mentor in my exploration of Logic and Knowledge Graphs. Ricardo has offered critical technical advice, guidance, and insightful feedback, which has been instrumental in realising this project. Balancing part-time studies with other commitments can be challenging. Therefore I appreciate the University of Bergen's administration's assistance in making this possible during my tenure in the military while writing this thesis. Finally, I appreciate my workplace, the 1st Corvette Squadron of the Royal Norwegian Navy, particularly my nearest leaders, Magnus and Torgrim. Their flexibility and support were pivotal in facilitating the completion of this work.

Anders Imenes
24 May, 2023

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Research Questions 2

1.3 Thesis Outline 3

2 Background **4**

2.1 *DL-Lite* Framework and Query Rewriting 4

2.1.1 Foundations of Description Logic and the *DL-Lite* Framework . . 5

2.1.2 Conjunctive Queries, Disjunctive queries, Unions of Conjunctive Queries, and EPFO Queries 8

2.1.3 Query Rewriting with PerfectRef 9

2.2 Knowledge Graphs 15

2.2.1 Knowledge Graph (KG) Completion 16

2.2.2 KG Embeddings 17

2.2.3 Performance indicators for Knowledge Graph Embeddings 28

2.3 Semantic Web Technologies: Resource Description Framework (RDF) and The W3C Web Ontology Language (OWL) 30

2.3.1 Understanding ontologies 30

2.3.2 Understanding RDF and RDF Schema (RDFS) 31

2.3.3 Exploring OWL 34

2.3.4 Relationship between OWL and Description Logic (DL) 36

3 Combining Query Rewriting with Knowledge Graph Embeddings for Complex Query Answering **39**

3.1 Strategy: Why It Works 39

3.2 Pipeline: How it works 41

3.2.1 Our Implementation of PerfectRef 41

3.2.2 Query answering from a Knowledge Graph Embedding 49

3.2.3 Comprehensive Examples 61

4 Results **70**
4.1 Datasets . . . 70
4.1.1 DBPedia15k: Overview and Characteristics . . . 71
4.1.2 The Family Dataset: A Wikidata5M Extract . . . 74
4.2 Research questions . . . 77
4.2.1 Question 1: How can we integrate query rewriting with Knowledge Graph Embeddings (KGEs) to enhance complex query answering? 77
4.2.2 Experiment introduction . . . 78
4.2.3 Question 2: How are the results affected by different KGEs? . . . 79
4.2.4 Question 3: How does our integrated query rewriting approach compare to standard KG lookups? . . . 93
4.2.5 Question 4: How do we interpret and compare our results fairly? 103

5 Related Work **106**

6 Conclusion **110**
6.1 Future Work . . . 112

List of Acronyms and Abbreviations **114**

Bibliography **116**

A Logic fundamentals **125**
A.1 Logic . . . 125
A.1.1 Propositional Logic . . . 125
A.1.2 First Order Logic . . . 126

B PerfectRef implementation running time **129**
B.1 Summation of all natural numbers . . . 129
B.2 Binomial coefficient . . . 130
B.2.1 Correlation between binomial coefficient and triangular numbers . 130
B.2.2 Summation over the first d triangular numbers . . . 131
B.3 Running time PerfectRef . . . 132

C Embedding results **135**

D Testcase results **143**
D.1 Family Dataset . . . 143
D.2 DBPedia15k . . . 146

E Experiment queries generation **149**
E.1 DBPedia15k 149
E.2 Family Dataset 165

List of Figures

2.1 Interplay between DL and First Order Logic (FOL). Illustration borrowed from [59]. 5
2.2 The PerfectRef Algorithm. Pseudocode image from [14] 11
2.3 A simple KG. Illustration from [28] 16
2.4 A very simple KGE illustration. Borrowed from Bratanic [12]. 17
2.5 Figure from [17]: *(a) TransE, (b) TransH, (c) TransR, (d) TransD, (e) TransA, (f) KG2E, (g) TransG* 20
2.6 TransE 21
2.7 An illustration from the BoxE paper, presenting a sample BoxE model for $d = 2$ [1]: *Each entity $e_i \in \mathcal{N}$ has an embedding $\boldsymbol{e}_i$ and defines a displacement on other entities, demonstrated with distinct colors. The binary relation $r \in \mathcal{R}$ is encoded via the box embeddings $\boldsymbol{r}^{(1)}$ and $\boldsymbol{r}^{(2)}$. This model generates the Knowledge Graph on r, as shown on the right.* . 22
2.8 Tensor-Factorization-based methods [17] 24
2.9 Figure from [53]: *Illustrations of TransE and RotatE with only one dimension of embedding* 26
2.10 Figure and description borrowed from original CompGCN paper [54] : "*Overview of CompGCN. Given node and relation embeddings, CompGCN performs a composition operation $\phi(.)$ over each edge in the neighbourhood of a central node (e.g. Christopher Nolan above). The composed embeddings are then convolved with specific filters $\boldsymbol{W}_O$ and $\boldsymbol{W}_I$ for original and inverse relations, respectively. We omit self-loop in the diagram for clarity. The message from all the neighbors is then aggregated to get an updated embedding of the central node. Also, the relation embeddings are transformed using a separate weight matrix. [54]*" 27
2.11 The Semantic Web 31
2.12 RDFS. Illustration from [13] 33
2.13 The Transition from RDFS to OWL. Source: [51] 36
2.14 OWL DL descriptions. Illustration from [51] 37

2.15 OWL Axioms and facts. Illustration from [51] 37

3.1 Our pipeline for integrating query rewriting with complex query answering using knowledge graph embeddings 42
3.2 The objects in our PerfectRef query parser 46
3.3 PerfectRef Implementation simplified overview 47
3.4 Queries using (p)rojection, (i)ntersection, and (u)nion 50
3.5 Variable hierarchy in a 'pi'-query 51
3.6 The four different lists used for validation of the correctness of predicted entities 56
3.7 A general example of a frequency count of 3 relations and 3 concepts. . . 59
3.8 Variable hierarchy for the rewritten query 63
3.9 Using the Application Programming Interface (API) to query for a parent. It returns one entity. 69

4.1 Family Dataset with RotatE, showing impact of **rewriting**: Optimistic Mean Reciprocal Rank (o-MRR) for List of entities from local KG with initial query (L/I) and List of entities from local KG with rewritten queries (L/R) 94
4.2 Family Dataset with RotatE, showing impact of **rewriting**: o-MRR for List of entities from online KG with initial query (O/I) and List of entities from online KG with rewritten queries (O/R) 95
4.3 Family Dataset with RotatE, showing impact of **predictions**: o-MRR for L/I and O/I 95
4.4 Family Dataset with RotatE, showing impact of **predictions**: o-MRR for L/R and O/R 96
4.5 DBPedia15k Dataset with TransE, showing impact of **rewriting**: o-MRR for L/I and L/R 97
4.6 DBPedia15k Dataset with TransE, showing impact of **rewriting**: o-MRR for O/I and O/R 98
4.7 DBPedia15k Dataset with TransE, showing impact of **predictions**: o-MRR for L/I and O/I 98
4.8 DBPedia15k Dataset with TransE, showing impact of **predictions**: o-MRR for L/R and O/R 99
4.9 Family Dataset with RotatE, showing impact of **predictions**: o-MRR for L/I and O/R 102
4.10 DBPedia15k Dataset with TransE, showing impact of **predictions**: o-MRR for L/I and O/R 103

5.1 BoxEL Embedded Space. Illustration from [60]: *The geometric interpretation of logical statements in ABox (left) and TBox (right) expressed by DL EL++ with BoxEL embeddings.* 108

A.1 FOL atom 127

List of Tables

2.1 Mapping between FOL and *DL-Lite* 6
2.2 The relationship of terms between KGs and Knowledge Bases (KBs) . . . 16
2.3 Orders of tensors 23
2.4 The relation matrix $\boldsymbol{M_r}$ 24
2.5 Key Meta Properties in RDF Schema (RDFS) 33
2.6 Relationship between terms in FOL, OWL and DL 38

3.1 The entity output of the query example: Q7565(?w) ∩ P40(?w,?y). Results from GitHub. 65
3.3 The entity output of the query example: q(?w) :- P25(?x,?w) 67
3.4 Updated entity output of the query example: q(?w) :- Q7566(?w). 69

4.1 The most used properties in the family knowledge graph 72
4.2 Frequency count of concepts in the knowledge graph 73
4.3 TBox mapping in the Family Dataset 75
4.4 The most used properties in Family Dataset 75
4.5 Best o-MRR and Hits@3 values on local rewriting validation (L/R) result comparison of different models for each query structure on the family dataset. 81
4.6 Local Delta values (Δ) (Rewritten scores vs initial scores) with comparison of different models for each query structure on the family dataset. 82
4.7 Delta values (Δ) for predictions (Online KG vs local KG) with comparison of different models for each query structure on the family dataset. 83
4.8 Best o-MRR and Hits@3 values on Local KG with rewritings (L/R) comparison of different models for each query structure on the DBPedia15k dataset. 84
4.9 Local Delta values (Δ) (Rewritten scores vs initial scores) with comparison of different models for each query structure on the DBPedia15k dataset. 85
4.10 Delta values (Δ) for predictions (Online KG vs local KG) with comparison of different models for each query structure on the DBPedia15k dataset. 86

4.11 Summary of results for different query structures for the dataset **family** and model **RotatE** (dim: 192, epoch: 24). All query structures have a count of 25 different queries. L/I: Local KG prediction using solely initial query, L/R: Local KG prediction using rewritten queries, O/I: Online KG prediction using solely initial query, O/R: Online KG prediction using rewritten queries. 93
4.12 Family Dataset with RotatE: Delta values (Δ) for rewriting. o-MRR and Hits@3 for each query structure for Local and Online KGs 94
4.13 Family Dataset with RotatE: Delta values (Δ) for KGE predictions. o-MRR and Hits@3 for each query structure for Local and Online KGs . . 94
4.14 Summary of results for different query structures for the dataset **DBPedia15k** and model **TransE** (dim: 192, epoch: 24). All query structures have a count of 25 different queries. L/I: Local KG prediction using solely initial query, L/R: Local KG prediction using rewritten queries, O/I: Online KG prediction using solely initial query, O/R: Online KG prediction using rewritten queries. 96
4.15 Dbpedia Dataset with TransE: Delta values (Δ) for **rewriting**. o-MRR and Hits@3 for each query structure for Local and Online KGs 97
4.16 DBPedia15k Dataset with TransE: Delta values (Δ) for KGE predictions. o-MRR and Hits@3 for each query structure for Local and Online KGs . 97
4.17 Delta values (Δ) between Online KG prediction using rewritten queries (O/R) and Local KG prediction using solely initial query (L/I) for the dataset family and model **RotatE** (dim: 192, epoch: 24). Positive values indicate that O/R outperformed L/I. 101
4.18 Delta values (Δ) between Online KG prediction using rewritten queries (O/R) and Local KG prediction using solely initial query (L/I) for the dataset DBPedia15k and model **TransE** (dim: 192, epoch: 24). Positive values indicate that O/R outperformed L/I. 102
4.20 Updated entity output of the query example: q(?w) :- Q7566(?w) 103

5.1 Table borrowed from [6]: "*Test MRR results on complex query answering across all query types. avg_p is the average on EPFO queries; avg_{ood} is the average on out-of-distribution (OOD) queries; avg_n is the average on queries with negation. Results on Hits@1 are in Appendix G.1.* " 107

A.1 Operators in Propositional Logic 126
A.2 Quantification in First-order Logic 127

C.1 Hits@1 results 136
C.2 Hits@3 results 137
C.3 Hits@5 results 138
C.4 Hits@10 results 139
C.5 Mean Rank (MR) results. DBPEDIA15k Triples count = 32 174, Family ontology triples count = 48692 140
C.6 o-MRR results 141
C.7 AMRI results 142

D.1 Summary of results for different query structures for the dataset family and model **BoxE** (dim: 192, epoch: 24). All query structures have a count of 25 different queries. L/I: Validation using entities from local KG lookup on the initial query, L/R: Validation using entities from local KG lookup on the rewritten queries, O/I: Validation using entities from online KG lookup on the initial query, O/R: Validation using entities from online KG lookup with the rewritten queries. 143
D.2 Summary of results for different query structures for the dataset family and model **CompGCN** (dim: 192, epoch: 24). All query structures have a count of 25 different queries. L/I: Validation using entities from local KG lookup on the initial query, L/R: Validation using entities from local KG lookup on the rewritten queries, O/I: Validation using entities from online KG lookup on the initial query, O/R: Validation using entities from online KG lookup with the rewritten queries. 144
D.3 Summary of results for different query structures for the dataset family and model **DistMult** (dim: 192, epoch: 16). All query structures have a count of 25 different queries. L/I: Validation using entities from local KG lookup on the initial query, L/R: Validation using entities from local KG lookup on the rewritten queries, O/I: Validation using entities from online KG lookup on the initial query, O/R: Validation using entities from online KG lookup with the rewritten queries. 144
D.4 Summary of results for different query structures for the dataset family and model **RotatE** (dim: 192, epoch: 24). All query structures have a count of 25 different queries. L/I: Validation using entities from local KG lookup on the initial query, L/R: Validation using entities from local KG lookup on the rewritten queries, O/I: Validation using entities from online KG lookup on the initial query, O/R: Validation using entities from online KG lookup with the rewritten queries. 145

D.5 Summary of results for different query structures for the dataset family and model **TransE** (dim: 192, epoch: 24). All query structures have a count of 25 different queries. L/I: Validation using entities from local KG lookup on the initial query, L/R: Validation using entities from local KG lookup on the rewritten queries, O/I: Validation using entities from online KG lookup on the initial query, O/R: Validation using entities from online KG lookup with the rewritten queries. 145
D.6 Summary of results for different query structures for the dataset dbpedia15k and model BoxE (dim: 192, epoch: 24). All query structures have a count of 25 different queries. L/I: Validation using entities from local KG lookup on the initial query, L/R: Validation using entities from local KG lookup on the rewritten queries, O/I: Validation using entities from online KG lookup on the initial query, O/R: Validation using entities from online KG lookup with the rewritten queries. 146
D.7 Summary of results for different query structures for the dataset dbpedia15k and model CompGCN (dim: 192, epoch: 24). All query structures have a count of 25 different queries. L/I: Validation using entities from local KG lookup on the initial query, L/R: Validation using entities from local KG lookup on the rewritten queries, O/I: Validation using entities from online KG lookup on the initial query, O/R: Validation using entities from online KG lookup with the rewritten queries. 147
D.8 Summary of results for different query structures for the dataset dbpedia15k and model DistMult (dim: 192, epoch: 24). All query structures have a count of 25 different queries. L/I: Validation using entities from local KG lookup on the initial query, L/R: Validation using entities from local KG lookup on the rewritten queries, O/I: Validation using entities from online KG lookup on the initial query, O/R: Validation using entities from online KG lookup with the rewritten queries. 147
D.9 Summary of results for different query structures for the dataset dbpedia15k and model RotatE (dim: 192, epoch: 24). All query structures have a count of 25 different queries. L/I: Validation using entities from local KG lookup on the initial query, L/R: Validation using entities from local KG lookup on the rewritten queries, O/I: Validation using entities from online KG lookup on the initial query, O/R: Validation using entities from online KG lookup with the rewritten queries. 148

D.10 Summary of results for different query structures for the dataset dbpedia15k and model **TransE** (dim: 192, epoch: 24). All query structures have a count of 25 different queries. L/I: Validation using entities from local KG lookup on the initial query, L/R: Validation using entities from local KG lookup on the rewritten queries, O/I: Validation using entities from online KG lookup on the initial query, O/R: Validation using entities from online KG lookup with the rewritten queries. 148

List of Listings

1 A simple RDF triple. 32
2 Example of RDFS triples 34
3 The main method in the PerfectRef Python Library 45
4 Snippet from the implementation of PerfectRef where the atom checks if is already processed 48
5 Snippet from the implementation of PerfectRef where duplicates are identified 48

Chapter 1

Introduction

1.1 Motivation

Knowledge Graphs (KGs) have garnered significant attention recently as powerful tools for representing complex relationships among entities. They have been employed in numerous applications, including search engines, recommendation systems, and natural language understanding [24]. However, querying these graphs remains a significant challenge, especially for complex queries involving multiple entities and relationships. To address this challenge, this thesis explores if integrating query rewriting with Knowledge Graph Embeddings (KGEs) could enhance the outcomes of complex query answering.

Query rewriting is used in ontology-based data access to translate complex queries into simpler and reformulated queries that can be directly executed on the available data. With this, we can ensure that the initial query maximizes the scope of the query response. This translation, often facilitated by algorithms such as PerfectRef [14], is primarily guided by a schema known as a TBox in the Description Logics parlance that describes the various concepts and relationships in the data. In this work, we investigate the impact of query rewriting on complex query answering tasks, leveraging a TBox to guide the query rewriting process to maximize the prediction scope.

Knowledge Graph Embeddings, on the other hand, translate the nodes and relationships of a KG into a vector space, allowing machine learning methods to predict unknown facts, find entity similarities, and, importantly, in our context, answer complex queries. Although KGEs have shown promising results in many of these tasks, their ability to

handle complex queries, particularly ones involving multiple entities and relationships, is still limited [49, 48, 23, 4, 6]. By integrating query rewriting as a preprocessing step for KGE predictions, we aim to push the boundaries of what can be achieved in complex query answering.

Several fascinating studies have been conducted in directions similar to ours. For instance, DeepProbLog by Manhaeve et al. [40] presents a method for distinguishing between low-level perception and high-level reasoning, simultaneously extracting the benefits from both aspects. Furthermore, promising research has been done in representing TBoxes within the embedded space, which unveils exciting possibilities for enhancing query answering methods [22, 32, 43, 60].

The central hypothesis of our research is that query rewriting, guided by a TBox, can enhance the effectiveness of query answering amalgamated with KGEs in handling complex query answering tasks. If successful, this could pave the way for more efficient and accurate querying of KGs, thereby enhancing their utility in a wide range of applications.

By the end of this research, we expect to understand the potential and limitations of integrating query rewriting with KGEs. The insights gained from this study could guide future research in this area, opening new avenues for enhancing the querying capabilities of KGs.

1.2 Research Questions

This work is structured around four central research questions to understand and evaluate the potential of this integration. We first examine how query rewriting can be integrated with KGEs. We then explore how the selection of different models influences the results of this integration. The third focus of our research is to compare the performance of our proposed approach against a traditional KG lookup method. Lastly, we discuss and interpret the results, providing insights into the strengths and potential areas for improvement of this approach. Formally, the research questions are:

- How can we integrate query rewriting with KGE to enhance complex query answering?
- How are the results affected by different KGEs?

- How does our integrated query rewriting approach compare to standard KG lookups?
- How do we interpret and compare our results fairly?

The integration we are asking for in the first research question is covered in chapter 3, while the remaining research questions will be adressed in chapter 4.

1.3 Thesis Outline

This thesis is separated into chapters presented in the following list.

- Chapter 2 introduces the fundamental concepts related to Description Logics, Query Rewriting, Knowledge Graphs, Knowledge Graph Embeddings, and the Semantic Web.
- Chapter 3 details the methodology adopted for integrating queries and embeddings, comparing performance against traditional KG queries.
- Chapter 4 presents the results of our study, provides a comparative analysis of the performance of different methods. Here, we answer the research questions.
- Chapter 5 briefly discusses some related works to complex query answering to our work.
- Chapter 6 concludes the thesis and suggests directions for future research.
- The appendices contain more logic, running time calculation, embedding results, the test results and query overview for our experiment.

Chapter 2

Background

In this chapter, we will explore the core concepts and technologies that underpin our research. We commence by examining the *DL-Lite* framework and the concept of query rewriting, laying a theoretical foundation for our methodological approach. Here, we will discuss the basics of the *DL-Lite* framework, and how it facilitates conjunctive queries and unions of conjunctive queries, leading us to the implementation of query rewriting using PerfectRef [14].

Subsequently, we turn our attention to the field of knowledge graphs, detailing the embedding methods that enable a more nuanced representation of complex data. We will also elucidate the metrics used for evaluating these representations.

Finally, we explore the Semantic Web technologies RDF and OWL, foundational tools for structuring and interpreting web data. This will involve an examination of RDF and RDFS, and an exploration of OWL 2.

2.1 *DL-Lite* Framework and Query Rewriting

This section is devoted to a comprehensive exploration of the *DL-Lite* Framework, a variety of query types, and the query reformulation algorithm PerfectRef [14]. These concepts form the bedrock of the discussions to follow; thus, gaining a robust understanding of them is our primary objective. As we navigate through these topics, a sound grasp of the fundamental terms of logic can significantly facilitate our journey. For this purpose, an introductory exposition of logic and its principal terminologies is presented in appendix A. While this appendix is designed to provide an enriched understanding, those of us already well-versed in propositional logic and first-order logic might find it sufficient to proceed directly with this section.

2.1.1 Foundations of Description Logic and the *DL-Lite* Framework

Description Logic (DL) are typically fragments of First Order Logic (FOL) that are decidable. Though they possess greater expressive power than Propositional Logic, they fall short in comparison to FOL, as shown in fig. 2.1. Despite this, DLs provides a logical formalism for the semantic web and facilitate ontology management, as seen with OWL, a topic we will delve into later. While we borrow the concept of quantifiers from FOL, their application and interaction with variables differ significantly in DLs.

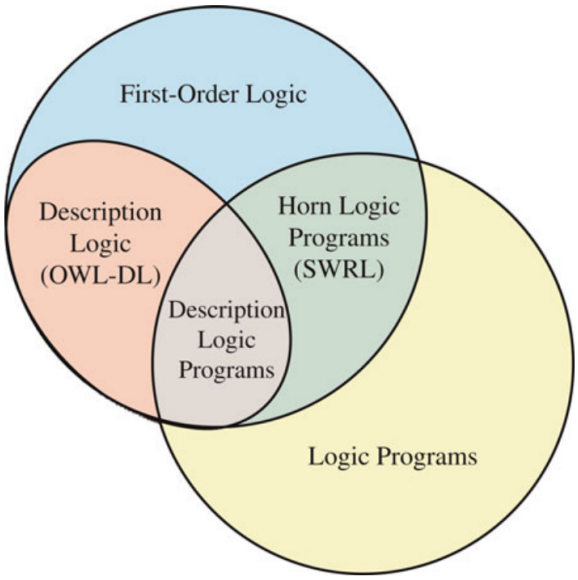


Figure 2.1: Interplay between DL and FOL. Illustration borrowed from [59].

Our focus narrows to a particular family of DLs, known as the *DL-Lite* framework. This framework, introduced by Calvanese et al. in 2007, offers tractable reasoning and lightweight query answering in Description Logic (DL) [14]. The authors highlight the unavoidable trade-off within DLs between expressive power and computational complexity. Despite many languages within DL being hampered by exponential time reasoning, *DL-Lite* provides polynomial time solutions for manageable reasoning in ontologies [14]. This computational efficiency becomes indispensable when working with large ontologies, a common scenario in web repositories.

DL-Lite comprises several DLs, three of which are particularly relevant: $\textit{DL-Lite}_{core}$, $\textit{DL-Lite}_{\mathcal{F}}$, and $\textit{DL-Lite}_{\mathcal{R}}$. The latter two extend the core language. However, in this thesis, we restrict our focus to $\textit{DL-Lite}_{\mathcal{R}}$, as $\textit{DL-Lite}_{\mathcal{F}}$ is used for specifying functionality, a feature absent in our implementation. In the following sections, we will introduce $\textit{DL-Lite}_{core}$ and $\textit{DL-Lite}_{\mathcal{R}}$. From this point forward, when we refer to the *DL-Lite* framework, we specifically mean the $\textit{DL-Lite}_{\mathcal{R}}$ profile.

Signature

The fundamental elements of the *DL-Lite* framework include *individuals*, *concepts*, and *roles*. These elements are used to articulate the domain of interest. For instance, atomic concepts in *DL-Lite* equate to unary predicates in FOL, and atomic roles align with binary predicates. Individuals and constants in *DL-Lite* correspond to constants in FOL. These elemental components together form the *signature* of the *DL-Lite* framework. These also correspond closely to FOL constructs, as illustrated in table 2.1 [14].

FOL	***DL-Lite***
Constant	Individual
Unary predicate	Concepts
Binary predicate	Role

Table 2.1: Mapping between FOL and *DL-Lite*

In *DL-Lite*, concepts generally begin with a capital letter, such as $Father(x)$ or $Country(x)$. Contrarily, roles typically start with a lowercase letter, examples being $hasFather(x, y)$ or $directedMovie(x, y)$. Lastly, individuals are usually denoted in lowercase, like *norway* or *inception*. Another way to identify is the number of terms in the predicate.

Syntax

DL-Lite syntax is covered by the following four axioms [14]:

$$B \rightarrow A \mid \exists R \qquad R \rightarrow P \mid P^{-}$$
$$C \rightarrow B \mid \neg B \qquad E \rightarrow R \mid \neg B$$

Here, A represents an *atomic concept* such as *Father*, and P indicates an *atomic role* like *isFatherOf*. The inverse of this role is expressed as P^{-}, for example, *hasFather*. The notation B stands for a *basic concept*, which could be an atomic concept A, or $\exists R$, where R is a *basic role* (either the atomic role P or its inverse property P^{-}). Finally, C represents a *(general) concept*, which is either a *basic concept* or its negation, and E extends this definition to roles. This notation facilitates the progression from atomic to basic to general concepts and roles.

A DL Knowledge Base (KB) comprises a TBox, $\mathcal{T}$, and an ABox, $\mathcal{A}$, forming a KB, $\mathcal{K} = \{\mathcal{T}, \mathcal{A}\}$. The TBox delineates how the domain elements are related, formally codifying the intentional knowledge required for reasoning. To draw a parallel, RDFS in the Semantic Web context functions as the TBox, outlining how the data elements interrelate. In *DL-Lite*, the TBox consists of *inclusion assertions* of the form $B \sqsubseteq C$, where C is a general concept, and B is a basic concept. The notation $\sqsubseteq$ signifies that B is *subsumed by* C, implying every instance of B is also an instance of C. As a side note, the term *inclusion assertion* is not much used in the community. The term *concept inclusion* is more familiar. However, using the concept inclusion term, we must remember that we can subsume B, not solely A.

The previous definitions are included in *DL-Lite*$_{core}$. However, *DL-Lite*$_{\mathcal{R}}$ also contains *role inclusion*, that is, $R \sqsubseteq E$. For instance, the role *isaSiblingTo* is subsumed by *isRelatedTo*. That example shows what *DL-Lite*$_{\mathcal{R}}$ adds to *DL-Lite*$_{core}$.

The ABox, $\mathcal{A}$, is what we think of as the dataset. It is a set of facts, or more formally, the set of atomic *concepts assertions* and *roles assertions*. We denote the constants with lowercase, yielding $A(a)$ for concept assertions and $P(a, b)$ for role assertions. An example of a concept assertion is *Father(bob)* and role assertion *isRelatedTo(alice,bob)*.

A KB in DL, $\mathcal{K} = \{\mathcal{T}, \mathcal{A}\}$, is the combination of the ABox and TBox. Note that the term KB is utilized by several communities and is often contextualized with KGs, which we will discuss later in this chapter.

Semantics

The semantics of *DL-Lite* define how its syntax maps onto real-world entities and relations. In DLs, this usually involves a descriptive semantics approach where an interpretation I is used. The interpretation I is a pair $(\Delta^I, .^I)$, where Δ^I is a non-empty set representing the *universe* or the set of all possible *things*, and the function $.^I$ provides a mapping that associates concepts, roles, and individual names to elements within Δ^I.

For *DL-Lite*$_{\mathcal{R}}$ [14], the interpretation works as follows:

- A concept A is interpreted as a subset of Δ^I, denoted as $A^I \subseteq \Delta^I$. For instance, the concept $Father$ would map to the subset of all entities in the universe that are fathers.

- A role P is interpreted as a subset of the Cartesian product $\Delta^I \times \Delta^I$, denoted as $P^I \subseteq \Delta^I \times \Delta^I$. For instance, the role $isFatherOf$ would map to the set of all pairs (x, y) where x is the father of y.
- An individual a is interpreted as a single element in Δ^I, denoted as $a^I \in \Delta^I$. For instance, an individual $john$ would map to a specific entity, John, in the universe.

Semantics lets us determine whether an assertion is true or false under a given interpretation. For instance, a TBox concept inclusion $B \sqsubseteq C$ is true under an interpretation I *if and only if* $B^I \subseteq C^I$. Similarly, an ABox membership assertion $A(a)$ is true under an interpretation I *if and only if* $a^I \in A^I$. By these means, the semantics ground the abstract logic in the concrete entities and relations of the real world.

2.1.2 Conjunctive Queries, Disjunctive queries, Unions of Conjunctive Queries, and EPFO Queries

In the context of *DL-Lite*, Conjunctive Query (CQ)s borrow their foundational premise from Conjunctive Query (CQ)s within FOL, as detailed in appendix A. To reiterate in the context of *DL-Lite*, a CQ within *DL-Lite* is characterized by the presence of *conjunctions* ($\cap$) and *existential quantifiers* ($\exists$). The query can be divided into a *head* and a *body*, separated by the symbols ':-' (In FOL: $\leftarrow$). The head of the query determines the variable (or individual) being queried, while the body contains a set of either concepts or roles. The variable in the head is defined as the *distinguished variable* and is the placeholder for entities we want to output from the body of the query.

A Union ($\cup$) of CQs is effectively a set of CQs, where the result is the union of the responses from each CQs.

Disjunctive queries are queries with a union ($\cup$). They can be rewritten (and interpreted) to a union of conjunctive queries as well. It holds as long as one of the conjunctive queries holds. For the conjunctive query, each atom in the query must hold.

Existential Positive First-order (EPFO) logical queries are a step further and encapsulate queries involving conjunction ($\cap$), disjunction ($\cup$), and existential quantifiers ($\exists$) [49]. Importantly, any EPFO query can be transformed into a union of CQs.

Consider the EPFO query with conjunction and disjunction:

$$q(?w) \text{:-} (ownsCompany(?x, ?w) \cup worksAt(?y, ?w)) \cap WorkPlace(?w) \tag{2.1}$$

This example can be rewritten into a set of CQs:

$$q(?w) :- ownsCompany(?x, ?w) \cap WorkPlace(?w) \tag{2.2}$$

$$q(?w) :- worksAt(?y, ?w) \cap WorkPlace(?w) \tag{2.3}$$

The final result is the union of CQ. The query answer is the union of the entities from each CQ.

In *DL-Lite*, the previously mentioned query types must be understood within the framework of a given KB. The atoms within the query must correspond to the concepts and roles specified within the TBox of the KB. Only *atomic* concepts and *atomic* roles can serve as atoms within a query. Consequently, we symbolize atoms in a CQ in *DL-Lite* as:

$$A(x) \text{ or } P(x_1, x_2)$$

For instance, if a KB contains the concept *Father* and the role *isFatherOf*, a potential CQ is:

$$q(x) \leftarrow Father(x) \cap isSiblingTo(x, y)$$

This query would return all individuals who are fathers and have a sibling.

2.1.3 Query Rewriting with PerfectRef

Introduction and motivation

In this section, we explore PerfectRef, a pioneering algorithm for query reformulation within DLs. Devised by Calvanese et al. [14], PerfectRef *rewrites* a CQ solely leveraging TBox information. In our work, we employ PerfectRef to expand the scope of our complex query answering. However, this algorithm possesses broader applications than usage for queries, which is beyond the scope of this thesis.

A KB can be queried directly for imminent query results. However, using a reasoner like HermiT [21], we can entail new facts across the entire KB to enrich the query results. This process, termed *reasoning* or "entailing new facts", uncovers new triples not explicitly stated in the ABox by utilizing connections within the TBox. However, loading the entire KB into memory can be significantly resource intensive. In contrast, PerfectRef offers an efficient alternative. It accepts a CQ as input, returning a union of CQs as output if potential rewritings exist from the TBox. PerfectRef ensures we cover all entailment implications without expanding the KB or loading it into memory. Instead, we extend the search scope to achieve query answers akin to full pre-entailment over the KB, greatly enhancing query answering efficiency.

PerfectRef is particularly beneficial for dynamic KBs, which frequently have additions to the ABox and TBox. In the context of query answering, it circumvents the need for a potentially demanding entailment process across the entire KB. Instead, it adeptly extends the original CQ into a larger union of CQs to include the addition. Thus, PerfectRef is a handy and robust tool for addressing query responses within evolving KBs.

It is worth noting, however, that this PerfectRef expansion strategy primarily contrasts with KB reasoning when it comes to query answering. There are valid reasons for conducting entailment to uncover new facts in a KB, such as maintaining and updating its completeness. As such, while PerfectRef provides an efficient alternative for query answering explicitly, it does not replace the value of traditional entailment processes within a KB.

Algorithm Overview

In broader terms, the PerfectRef algorithm (fig. 2.2) accepts two inputs, a TBox $\mathcal{T}$ and a CQ, q. It does not allow EPFO or disjunctive queries as input. Upon execution, it returns a union of conjunctive queries, denoted PR, induced by the information in the TBox. Each iteration of the algorithm performs two tasks: firstly, it determines whether an atom is rewritable with another based on the axioms in the TBox, and secondly, it checks if any pairs of atoms in the query are reducible while retaining their resolution (i.e. without changing the output scope of the CQ). New entailed queries add to PR, and the algorithm concludes when no new reformulations exist.

The TBox consists of a set of inclusion assertions or axioms. However, not all inclusion assertions are accepted by the algorithm. For this reason, we distinguish between Positive

Algorithm PerfectRef $(q, \mathcal{T})$
Input: conjunctive query q, TBox $\mathcal{T}$
Output: union of conjunctive queries PR
$PR := \{q\}$;
repeat
  $PR' := PR$;
  for each $q \in PR'$ **do**
  (a) **for each** g in q **do**
      for each PI I in $\mathcal{T}$ **do**
        if I is applicable to g
        then $PR := PR \cup \{q[g/gr(g, I)]\}$
  (b) **for each** g_1, g_2 in q **do**
      if g_1 and g_2 unify
      then $PR := PR \cup \{\tau(reduce(q, g_1, g_2))\}$;
until $PR' = PR$;
return PR

Figure 2.2: The PerfectRef Algorithm. Pseudocode image from [14]

Inclusions (PIs) and Negative Inclusions (NIs) [14]. PIs refer to assertions where $B_1 \sqsubseteq B_2$ or $R_1 \sqsubseteq R_2$, while NIs refer to inclusion assertions where we have a negated side, i.e., $B_1 \sqsubseteq \neg B_2$ or $R_1 \sqsubseteq \neg R_2$. It is important to note that PerfectRef is created to manage only PIs.

In addition to distinguished and non-distinguished variables, we introduce the term *shared* variables, which are variables that appear at least twice in the body of the query. Consequently, a *bound* variable is defined as a variable that is either distinguished, shared or a constant. In contrast, an *unbound* variable is a non-distinguished, non-shared variable. In the following algorithm breakdown, an unbound variable is represented by the notation '_'.

Algorithm breakdown

The algorithm accepts a TBox, $\mathcal{T}$, and a CQ, q, as its inputs and outputs a union of CQs, PR, derived from the TBox. The algorithm involves two primary operations per iteration: identifying if an atom is rewritable based on the TBox and evaluating if it can reduce any pairs of atoms in the query without compromising its resolution (i.e. without changing the output scope of the CQ).

The main loop of the algorithm initiates. Firstly, it creates a duplicate of PR, denoted PR'. It uses the latter set by the end of the iteration to validate if any entailments or

reductions did occur during its iteration. If not, the algorithm will terminate. The criterion of the loop is to continue iterating as long as PR and PR' are different. Based on its proof of termination [14], we know that it indeed terminates when these sets are identical.

After initializing PR', the algorithm begins its iteration over each CQ within PR. In the first run of PerfectRef, this is simply the input query, q. Once it selects a query from PR', the algorithm invokes its first feature. This feature scans the query for atoms that can be rewritten according to the TBox, as denoted by '(a)' in fig. 2.2. During this process, the algorithm iterates over every atom in the query, with the currently inspected atom represented as g. Subsequently, an iteration over all PIs within the TBox initiates. For each PI, denoted I, the algorithm assesses whether the current atom g can be rewritable to the current PI I. This assessment returns boolean values, true or false, based on the truth of one of the following conditions [14]. It is important to remember that $A(x)$ represents atomic concepts, while P and P^- signify atomic roles.

- A PI I applies to an atom $A(x)$ if I has A on its right-hand side;
- a PI I applies to an atom $P(x_1, x_2)$ if $x_2 = \_$, and the right-hand side of I is $\exists P$;
- a PI I applies to an atom $P(x_1, x_2)$ if $x_1 = \_$, and the right-hand side of I is $\exists P^-$; or
- a PI I applies to an atom $P(x_1, x_2)$ if I is a role inclusion, and the right-hand side is either P or P^-.

The purpose of these four states is to enable an early termination of the search for a match, as continuing the search would require a more time and space-consuming process. However, this step is not strictly necessary since these conditions will be more explicitly checked in the next phase when a rewriting occurs. Nonetheless, this method benefits the algorithms efficiency as it halts the process before it attempts to rewrite any mismatches.

Proceeding to the next phase within step (a), we are confident that the current atom g and the PI I are applicable. Consequently, we require a method that accurately identifies the states of the inclusion assertion and the atom and performs the correct rewrite. The following method, $gr(g, I)$, is extracted verbatim from definition 32 in the PerfectRef paper [14].

The $gr(g, I)$ method accepts an atom g and a PI I as inputs and then rewrites them based on one of the following scenarios.

- If $g = A(x)$ and $I = A_1 \sqsubseteq A$, then $gr(g, I) = A_1(x)$;
- if $g = A(x)$ and $I = \exists P \sqsubseteq A$, then $gr(g, I) = P(x, \_)$;
- if $g = A(x)$ and $I = \exists P^- \sqsubseteq A$, then $gr(g, I) = P(\_, x)$;
- if $g = P(x, \_)$ and $I = A \sqsubseteq \exists P$, then $gr(g, I) = A(x)$;
- if $g = P(x, \_)$ and $I = \exists P_1 \sqsubseteq \exists P$, then $gr(g, I) = P_1(x, \_)$;
- if $g = P(x, \_)$ and $I = \exists P_1^- \sqsubseteq \exists P$, then $gr(g, I) = P_1(\_, x)$;
- if $g = P(\_, x)$ and $I = A \sqsubseteq \exists P^-$, then $gr(g, I) = A(x)$;
- if $g = P(\_, x)$ and $I = \exists P_1 \sqsubseteq \exists P^-$, then $gr(g, I) = P_1(x, \_)$;
- if $g = P(\_, x)$ and $I = \exists P_1^- \sqsubseteq \exists P^-$, then $gr(g, I) = P_1(\_, x)$;
- if $g = P(x_1, x_2)$ and $I = P_1 \sqsubseteq P$, then $gr(g, I) = P_1(x_1, x_2)$;
- if $g = P(x_1, x_2)$ and $I = P_1^- \sqsubseteq P^-$, then $gr(g, I) = P_1(x_1, x_2)$;
- if $g = P(x_1, x_2)$ and $I = P_1 \sqsubseteq P^-$, then $gr(g, I) = P_1(x_2, x_1)$; or
- if $g = P(x_1, x_2)$ and $I = P_1^- \sqsubseteq P$, then $gr(g, I) = P_1(x_2, x_1)$.

In the subsequent line of PerfectRef, $q[g/gr(g, I)]$ represents a new CQ where the output atom from $gr(g, I)$ replaces the original atom g. This new CQ appends to the union of CQs, PR. It is important to note that it adds to the set PR, not PR'.

We proceed to the second and final prominent feature of PerfectRef, reduction. In the following section of the algorithm, denoted '(b)', we remain within the same iteration of the current query q. It is important to note that potential additions from the previous step, (a), will not be considered during this iteration, since we are considering queries from PR'. At step (b), it identifies the set of all possible pairs of atoms in q. For example, if the query contains two atoms, there will only be one pair; if it has three atoms, there will be three combinations. This loop processes each pair, with each atom denoted as g_1 and g_2. The following method, $unify$, checks whether these atoms are *unifiable*. That is, it checks if the atoms represent the same concepts or roles, regardless of their variables. The occurrence of two identical atoms in a query may seem confusing since writing a query with two identical atoms is uncommon. However, a previous iteration of q could have created a duplicate atom through rewriting (step (a)), which will be addressed in this step.

If the unification test returns True, PerfectRef activates its *reduce* method. This method takes atoms g_1 and g_2 and the query as input. It returns *the most general unifier* between the two atoms, replacing these two atoms in the query with a single instance of that atom while maintaining resolution. If we consider two equal concepts with different

variables, if one is bound and the other is not, then the most general unifier is the bound one.

Consider the concepts $g_1 = A(x)$ and $g_2 = A(y)$. If both variables x and y are bound, no reduction occurs. If y is unbound, it reduces to $A(x)$. If both are unbound, it reduces to $A(_)$. If both are equal, $g_1 = A(x)$ and $g_2 = A(x)$, it reduces to one instance. The same procedure applies to roles. For $P(x_1, x_2)$ and $P(x_3, x_4)$, the most general unifier is selected between x_1 and x_3 for the first index, and x_2 and x_4 for the latter index. After the reduction, we obtain a new CQ q where g_1 and g_2 have been reduced. Note that a non-distinguished shared variable might have been reduced such that it only occurs once, resulting in an unbound state. Consequently, the new query q undergoes the τ-method, which updates whether variables are still shared, bound, or unbound after a reduction. Finally, the new query q is added to the union of CQs, PR (to clarify: not to PR').

Once the first query iteration completes, the current q within the set PR is designated as processed, meaning the algorithm will not visit that query again. Since new rewritten queries might have been introduced during steps (a) and (b), the sets PR and PR' may not align, prompting the algorithm to continue execution. This continuation begins by reassigning PR' to match the current state of PR. The algorithm, designed only to process unprocessed queries, will not initiate the next iteration with the same q as the previous one. Instead, it will select the first rewritten CQ from the preceding iteration, assessing its potential for additional reformulations or reductions.

The execution of the algorithm persists as long as unprocessed queries exist within the set PR'. The algorithm halts when it meets two conditions: (1) all queries within PR' have been processed, and (2) no new CQs has been added to PR on the last iteration (hence not adding any new unprocessed rewritings). Thus, the termination of the algorithm is governed by two interconnected criteria: the absence of unprocessed queries in PR', implying the lack of any new additions to the query set on the last iteration, signified by the equality of PR and PR'. Consequently, when the final query in PR' has been processed, and no new queries are introduced, PR will align with PR', resulting in the termination of the algorithm. The algorithm concludes by outputting the final union of CQs, PR.

2.2 Knowledge Graphs

This section provides an overview of KGs and explores the process of making predictions using these structures.

The definition of Knowledge Graph (KG) is not unanimously agreed upon within the KG community [25]. However, the fundamental components of a KG are well-established; a KG is, in essence, a graph (a compilation of nodes and edges) that illustrates structured knowledge [28]. This definition underscores the structural focus of a KG. Similarly, Krötzsch and Weikum defines KGs as a network of entities, inclusive of their semantic types, properties, and relationships [31]. An example of a KGs structure is illustrated in fig. 2.3.

Some definitions of KGs imply familiarity with the concept of KBs from a domain that will be introduced in a subsequent section of this chapter, the Semantic Web. However, the distinction between a KB and a KG is somewhat nebulous, leading to occasional (and incorrect) interchangeability of the terms [28]. This confusion is partly due to the frequent use of KGs for *reasoning* within the Semantic Web framework [25]. A definition of KGs that encapsulates this connection is: "In knowledge representation and reasoning, the KG is a KB that employs a graph-structured data model or topology for data integration" [58]. An alternative definition that does not presuppose familiarity with the Semantic Web is: "We define a KG as a graph of data designed to accumulate and convey knowledge of the real world, where nodes represent entities of interest and edges represent various relations between these entities" [25]. For this thesis, we will adhere to the definition that a KG is a graph-structured data model of a KB.

An RDF triple consists of the following terms:

```
subject     property    object .
```

In the context of KGs, where graph theory is employed, the terminology used for triples is graph-related:

```
head     relation    tail
```

Given a KG, $\mathcal{G} = (\mathcal{N}, \mathcal{R}, \mathcal{T})$, where $\mathcal{N}$ is a set of entities, $\mathcal{R}$ a set of edges, and $\mathcal{T}$ a list of triples, a triple is typically abbreviated to $(h, r, t) \in \mathcal{T}$, where $h, t \in \mathcal{N}$ and $r \in \mathcal{R}$. Furthermore, we highlight the relationships between terms shown in table 2.2.

KGs	KBs
Entities or nodes	Individuals
Relations or edges	Roles / Property
(head, relation, tail)	(subject, property, object)

Table 2.2: The relationship of terms between KGs and KBs

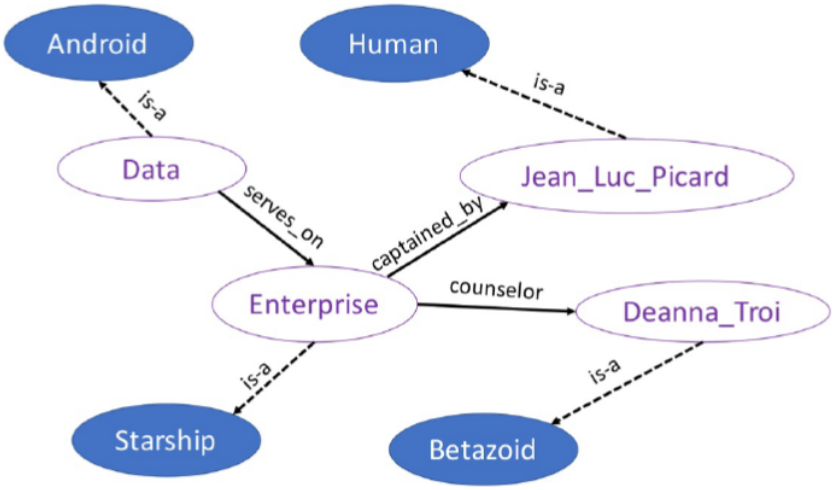


Figure 2.3: A simple KG. Illustration from [28]

2.2.1 KG Completion

In the field of knowledge representation and reasoning, the term *completeness* is a data quality measure that refers to the amount of information present in a particular dataset [62]. Most KGs are not *complete*, indicating that they are missing specific facts or statements. The study area devoted to addressing these missing facts is known as *KG Completion*. There are several methods to identify and fill in these missing facts. For instance, if a KG includes an ontology or schema, it is possible to *entail* new facts by employing a reasoner, such as HermiT [21]. This reasoning capability is underscored in a definition of KGs, which states: "A KG acquires and integrates information into an ontology and applies a reasoner to derive new knowledge" [20].

Open and Closed World Assumptions

Identifying and adding new facts raises a critical question: how can we ascertain the truth of these facts? This question bifurcates the approach to KG completion into two categories: the Open World Assumption (OWA) and the Closed-World Assumption (CWA).

Assume we have an incomplete KG, $\mathcal{G}'$. Then, we complete $\mathcal{G}'$ by finding triples $\mathcal{T}' = \{(h, r, t) \mid h \in \mathcal{N}, r \in \mathcal{R}, t \in \mathcal{N}, (h, r, t) \notin \mathcal{T}\}$ over the incomplete $\mathcal{G}'$ [50].

Under the CWA, we assume that new $\mathcal{T}'$ facts are false unless explicitly declared in the KG. In contrast, the OWA leaves the truth value of new facts undetermined, given the lack of proof either validating or refuting these facts.

2.2.2 KG Embeddings

In Machine Learning (ML), we find several *embedding* strategies. We can build automatic representation data, perform reasoning, and recognize patterns with embeddings. One type of embedding method is Knowledge Graph Embedding (KGE). The objective of KGEs is to acquire the depiction of a KG through the process of embedding the graph with a Vector Space Model (VSM) into lower dimensions [9]. The vector representations learned by the model can be used for various downstream tasks, such as link prediction, entity classification, and relation extraction. With KGEs, we embed our KG to a VSM, representing entities and relations as *vectors* in a subspace, shown in fig. 2.4.

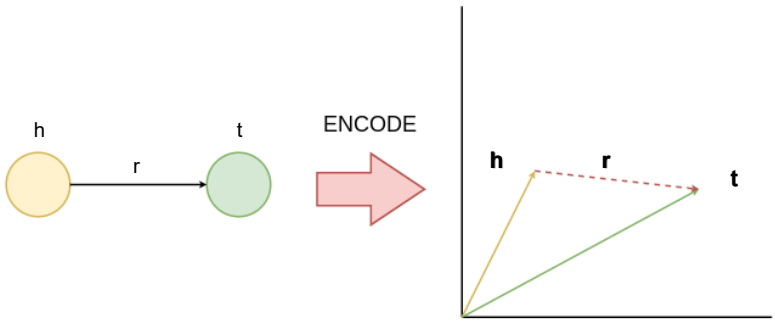


Figure 2.4: A very simple KGE illustration. Borrowed from Bratanic [12].

There are several methods to embed a KG, and all of them share the same design goals; They want to capture information patterns regarding entities, relations, domains, ranges, and hierarchies. It might not be the case that the same entity has the same vector. It might vary depending on whether it serves as the head or tail in the triple. Such variances often characterize more complex embedding approaches.

An abstract and general KGE pipeline

Building a KGE model is an intricate process that involves a series of steps akin to the standard procedure of training a machine learning model. First and foremost, we require a dataset encompassing training, validation, and test sets of triples. An additional step in this context is the generation of corrupt or negative triples, utilized in the objective functions as counterparts to correct triples.

Next, all vectors are initialized utilizing an appropriate method, for instance, *Xavier initialization* [33]. Although the forward step varies based on the model, a common strategy involves predicting the remaining part of the triple, given the other two components. This prediction is then passed through a loss and scoring (or objective) function, which aims to assign high scores to positive triples and low scores to negative ones.

Subsequently, the backpropagation process is performed, during which the gradients in the embedding are optimized using gradient descent as in traditional ML approaches. The final step involves updating the vectors in light of the most recent gradient adjustments. This iterative learning process continues across batches and epochs until a final KGE model is achieved. Standard ML techniques, such as early stopping and regularization parameters, are incorporated to mitigate the risk of overfitting.

Although KGE models vary in their structure and approach, making it challenging to classify them into distinct categories, Dai et al. [17] proposes a division into three primary methodologies:

- translation-based methods;
- tensor-factorization-based methods; and
- neural Network-based methods.

We will adopt this categorization for subsequent discussions of different models. This thesis will investigate models from each category; namely, TransE [10], DistMult [61], BoxE [1], RotatE [53], and CompGCN [54]. Before diving into these models, we will outline the two loss functions utilized in our research.

Loss functions

We utilize two different loss functions in this work. The first is Margin Ranking Loss (MRL) and the latter Self-adversarial negative sampling loss (NSSALoss).

Margin Ranking Loss The Margin Ranking Loss (MRL) function is a frequently used loss function for embeddings and is employed by all models except BoxE in this thesis. The MRL function is designed around the concept of a *margin*, which represents the minimum distance between the embeddings of a positive triple (i.e., a triple that exists in the KG) and a negative triple (i.e., a non-existent triple). The objective is to maximize this margin while simultaneously minimizing the loss [3].

We define L as the MRL function, shown in eq. (2.4).

$$L(k, \bar{k}) = \max\left(0, \gamma + f(\bar{k}) - f(k)\right) \tag{2.4}$$

Here, f is a scoring function that computes a similarity score between embeddings, k denotes positive triples (triples present in the KG), $\bar{k}$ signifies negative triples (triples not present in the KG), and γ is the margin. Depending on the specific embedding model in use, the form of the scoring function, f, can vary. We will discuss each scoring function for every model later.

During training, the models adjust the embeddings to minimize the sum of the pairwise hinge loss for all pairs of positive and negative triples in the training data, shown in eq. (2.5).

$$\min \sum L(k, \bar{k}), \tag{2.5}$$

In eq. (2.5), the sum is taken over all pairs of entities and relationships in the training set, both positive and negative. This function is referred to as the objective function.

NSSALoss The Self-adversarial negative sampling loss (NSSALoss) has a new approach for drawing negative samples [53]. The traditional negative sampling loss from Mikolov et al. [41] yields

$$L = -\log \sigma(\gamma - d_r(\boldsymbol{h}, \boldsymbol{t})) - \sum_{i=1}^{n} \frac{1}{k} \log \sigma(d_r(\boldsymbol{h'}_i, \boldsymbol{t'}_i) - \gamma),$$

where γ is a fixed margin, σ is the sigmoid function, (h'_i, r, t'_i) is the i-th negative triple and d_r is the scoring function.

However, NSSALoss new approach does not uniformly produce negative samples (determined by $\frac{1}{k}$ in the formula) but distributes them by the current embedding model [53]. It distributes by the fraction formula eq. (2.6).

$$p(h'_j, r, t'_j \mid \{(h_i, r_i, t_i)\}) = \frac{\exp \alpha f_r(\boldsymbol{h'}_j, \boldsymbol{t'}_j)}{\sum_i \exp \alpha f_r(\boldsymbol{h'}_i, \boldsymbol{t'}_i)}, \tag{2.6}$$

In eq. (2.6), α is the temperature of sampling. The formula yields a probability, and we substitute $\frac{1}{k}$ with the probability and treat it as the weight of the negative sampling. This yields the final NSSALoss formula shown in eq. (2.7).

$$L = -\log \sigma(\gamma - d_r(\boldsymbol{h}, \boldsymbol{t})) - \sum_{i=1}^{n} p(h'_i, r, t'_i) \log \sigma(d_r(\boldsymbol{h'}_i, \boldsymbol{t'}_i) - \gamma). \tag{2.7}$$

Translation-based methods

One of the categorizes for KGEs is the translation-based approach [17], which generates vector representations for entities $h \in \mathcal{N}$, $t \in \mathcal{N}$, and relationships $r \in \mathcal{R}$ by enforcing the vector for entity t to be in proximity to the *sum of vectors* for entities h and r [10].

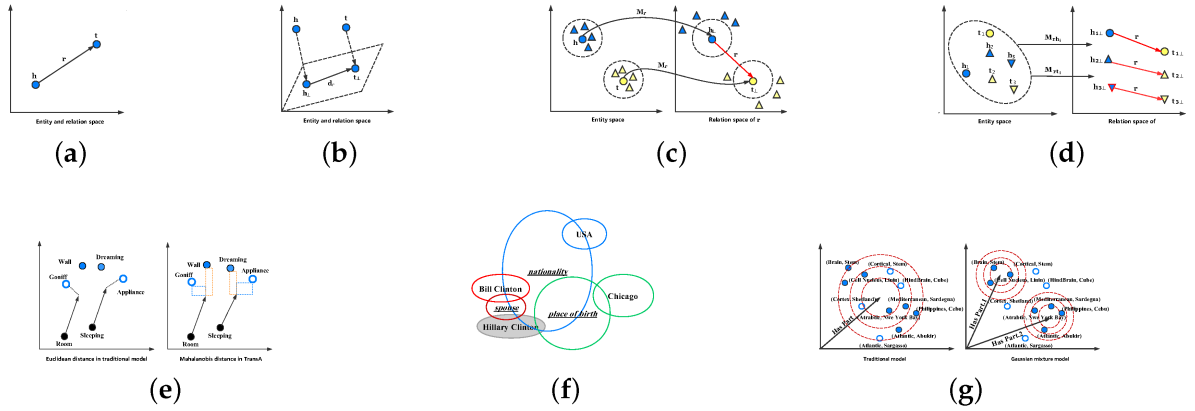


Figure 2.5: Figure from [17]: *(a) TransE, (b) TransH, (c) TransR, (d) TransD, (e) TransA, (f) KG2E, (g) TransG*

In the translation-based "family" of models, we find models like TransH [57], TransD [26], and TransR [38]. For instance, the first-mentioned adds h and t using hyperplanes. However, in this thesis, we will use the simplest model, TransE [10].

TransE TransE is a KG embedding model proposed in 2013 by Bordes et al. [10]. In TransE, each entity and relation is represented as a k-dimensional vector, where k is the dimensionality of the embedding space. The model defines the meaning of a relation as the translation between the embeddings of the head and tail entities connected by the relation. In other words, the embedding of the tail entity can be obtained by adding the embedding of the head entity and the embedding of the relation.

Formally, given a triple (h, r, t) representing a relation $r \in \mathcal{R}$ between entities $h \in \mathcal{N}$ and $t \in \mathcal{N}$, the TransE model learns the embedded vector representations $\boldsymbol{h}$, $\boldsymbol{r}$, and $\boldsymbol{t}$ such that:

$$\boldsymbol{h} + \boldsymbol{r} \approx \boldsymbol{t},$$

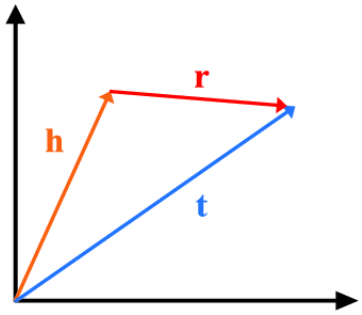


Figure 2.6: TransE

where '+' denotes vector addition, and ($\boldsymbol{h}$, $\boldsymbol{r}$, and $\boldsymbol{t}$) are the vector representations of (h, r, t). We denote $\mathcal{N}$ as the set of entities, $\mathcal{R}$ as set of relations, S as a set of correct triples from the KG. We denote S' as the set of corrupt triples shown in eq. (2.8).

$$S'_{(h,r,t)} = \{(h', r, t) \mid h' \in \mathcal{N}\} \cup \{(h, r, t') \mid t' \in \mathcal{N}\} \tag{2.8}$$

The model's objective is to minimize a MRL that penalizes the model for predicting incorrect tails for a given head and relation and incorrect heads for a given tail and relation. The loss function is specified for TransE as shown in eq. (2.9).

$$L = \sum_{(h,r,t) \in S} \sum_{(h',r,t') \in S} \max(0, [\gamma + d(\boldsymbol{h} + \boldsymbol{r}, \boldsymbol{t}) - d(\boldsymbol{h'} + \boldsymbol{r}, \boldsymbol{t'})]) \tag{2.9}$$

In eq. (2.9), 'max' denotes the positive part of x, $\gamma > 0$ is a margin hyperparameter, and d is a distance (scoring) function (e.g., L1 or L2 distance) that measures the dissimilarity between the embeddings of the head, relation, and tail, $\boldsymbol{t}$ and $\boldsymbol{t'}$ are the correct and incorrect tails, and $\boldsymbol{h}$ and $\boldsymbol{h'}$ are the correct and incorrect heads.

In summary, TransE learns embeddings for entities and relations in a KG by modelling relations as translations between entity embeddings in a low-dimensional vector space. The model is trained to minimize a MRL that penalizes it for predicting incorrect tails or heads.

BoxE BoxE, proposed by Abboud et al. in their 2020 paper, "BoxE: A Box Embedding Model for Knowledge Base Completion" [1], is a unique model that is somewhat challenging to categorize and is not explicitly discussed in the survey by Dai et al. [17]. Its creators classify BoxE as a "spatio-translational" model.

The distinctiveness of BoxE lies in its geometric representation of entities and relations. Specifically, it depicts each entity and relation as a box-shaped hyperrectangle in a d-dimensional Euclidean space [1]. A hyperrectangle signifies each entity, while each edge is represented as a linear transformation that maps one hyperrectangle onto another, as demonstrated in fig. 2.7.

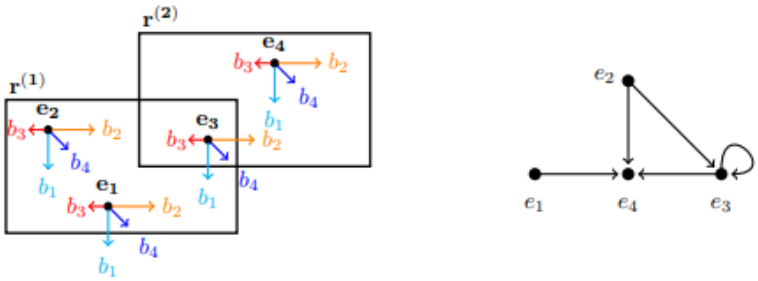


Figure 2.7: An illustration from the BoxE paper, presenting a sample BoxE model for $d = 2$ [1]: *Each entity $e_i \in \mathcal{N}$ has an embedding $\boldsymbol{e}_i$ and defines a displacement on other entities, demonstrated with distinct colors. The binary relation $r \in \mathcal{R}$ is encoded via the box embeddings $\boldsymbol{r}^{(1)}$ and $\boldsymbol{r}^{(2)}$. This model generates the Knowledge Graph on r, as shown on the right.*

In BoxE, every entity $e_i \in \mathcal{N}$ is associated with two vectors $\boldsymbol{e_i}, \boldsymbol{b_i} \in \mathbb{R}$. The vector $\boldsymbol{e_i}$ signifies the base position of the entity, depicted as black points in fig. 2.7. The vector $\boldsymbol{b_i}$, termed as the *translational bump vector*, is intended to displace the entity's base position in a triple, contingent on the other entities. This results in a dynamic entity representation, formulated as shown in eq. (2.10).

$$\boldsymbol{e_i^{e1,\ \cdots\ ,\ e_n}} = (\boldsymbol{e_i} - \boldsymbol{b_i}) + \sum_{1 \leq j \leq n} \boldsymbol{b_j} \tag{2.10}$$

For a given triple, both the head and the tail are represented by a hyperrectangle, as illustrated by $\boldsymbol{r^{(1)}, r^{(2)}}$ in fig. 2.7. In the case of a valid triple $(e_1, r^{(1)}, e_4)$, it receives a high rank because $\boldsymbol{e_1^{r(e1,e4)}} = (\boldsymbol{e_1} + \boldsymbol{b_4})$ is a point within the head box, $\boldsymbol{r_1}$, and $\boldsymbol{e_4^{r(e1,e4)}} = (\boldsymbol{e_4} + \boldsymbol{b_1})$ lies within the tail box, $\boldsymbol{r_2}$.

The scoring function in BoxE is defined as the sum of the L-x norms of the distances across all entity and relation boxes [1], shown in eq. (2.11).

$$\text{score}(r(e_1, \ldots, e_n)) = \sum_{i=1}^{n} \left|\left| \text{dist}(\boldsymbol{e_i^{r(e_1,\ldots,e_n)}}, \boldsymbol{r^{(i)}}) \right|\right|_x . \tag{2.11}$$

Our work uses the scoring function, eq. (2.11), with the NSSALoss as the objective function.

Tensor-Factorization-based methods

Tensor-factorization-based methods are a family of ML techniques that use tensor factorization to learn low-dimensional vector representations of entities and relations. Moreover, these methods represent the KG as a three-dimensional **tensor**, where the two dimensions correspond to the entities, and the third dimension corresponds to the relations (illustrated in fig. 2.8, where we see the design of RESCAL [17]).

The goal of tensor-factorization-based methods is to *decompose* the tensor into matrices (or vectors) representing the entity embeddings, the relation embeddings, and the interaction between entities and relations, respectively.

We define three orders for tensors for tensor-factorization-based methods

Order 0	Scalar-like	[0], [1], [0.5], ...
Order 1	Vector-like	[1,0], [1,4,5], [4,3,-2,1], ...
Order 2	Matrix-like	$\begin{bmatrix} 1 & 4 \\ -3 & 2 \end{bmatrix}$, ...

Table 2.3: Orders of tensors

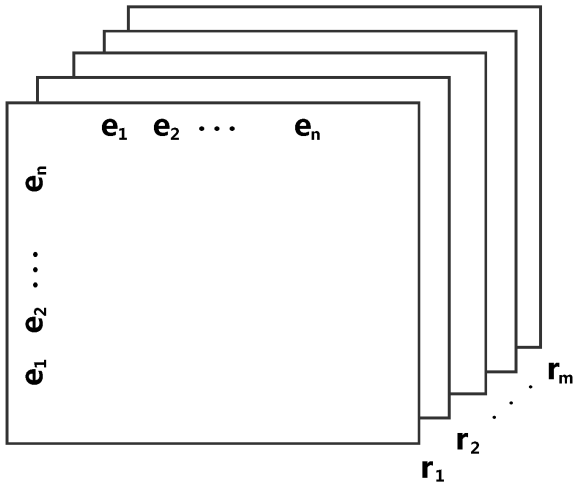


Figure 2.8: Tensor-Factorization-based methods [17]

DistMult DistMult is a tensor-factorization-based embedding method proposed in 2014 by Yang et al. [61]. This model uses the approach from the RESCAL model [42], but with some crucial distinctions seen in table 2.4. We denote set a $A \in \mathbb{R}^{n \times d}$ containing all entities, and we define the relation as a vector $\boldsymbol{r} \in \mathbb{R}^d$. In RESCAL, the relation is of the second order, while it is restricted to the first order for DistMult, shown in table 2.4 (following the rules per table 2.3). The relation vector is then up-dimensioned by making it a diagonal matrix. This simplifies the matrix multiplication in RESCAL to a simple dot product multiplication in DistMult. It is computationally more efficient than RESCAL due to its limitation to diagonal matrices; however, it is less expressive.

RESCAL	**DistMult**
$\boldsymbol{M}_r \in \mathbb{R}^{dxd}$	$\boldsymbol{M}_r = diag(\boldsymbol{r}) \mid \boldsymbol{r} \in \mathbb{R}^d$
Order 2	Order 1

Table 2.4: The relation matrix $\boldsymbol{M_r}$

The scoring function of the model is obtained as shown in eq. (2.12).

$$
\begin{aligned}
f_{DistMult}(h,r,t) &= \boldsymbol{h}^\top diag(\boldsymbol{r})\boldsymbol{t} \\
f_{DistMult}(h,r,t) &= \begin{bmatrix} h_1 & h_2 & \dots & h_d \end{bmatrix} \begin{bmatrix} r_1 & 0 & \dots & 0 \\ 0 & r_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & r_d \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_d \end{bmatrix} \\
f_{DistMult}(h,r,t) &= \sum_{i=1}^{d} h_i r_i t_i
\end{aligned}
\tag{2.12}
$$

Noteworthy, the head and tail order does not matter in the dot product. The final scoring function is shown in eq. (2.13).

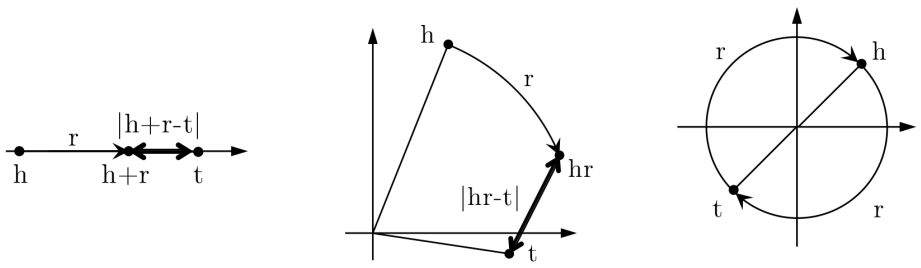
$$
f_{DistMult}(h,r,t) = \sum_{i=1}^{d} h_i r_i t_i = \sum_{i=1}^{d} t_i r_i h_i. \tag{2.13}
$$

As eq. (2.13) indicates, DistMult loses the ability to express asymmetric relations, since the order of multiplication does not matter. However, one advantage of DistMult is that it is computationally efficient, as the dot product can be computed quickly and in parallel. Another advantage is that the multiplicative interaction assumption allows the model to capture complex relationships between entities and relations while keeping the model relatively simple.

RotatE RotatE, proposed in 2019 by Sun et al. [53], is a unique model whose categorization is subject to debate. A survey on KGs consider it a translation-distance model [56], whereas the survey adopted for our definitions classifies it as a tensor factorization-based model [17].

In the RotatE model, the head and tail entities $h, t \in \mathcal{N}$ are mapped onto the complex plane for embeddings $\boldsymbol{h}, \boldsymbol{t} \in \mathbb{C}^k$. Given a triple (h, r, t), the functional mapping driven by each relation $r \in \mathcal{R}$ is defined as an *element-wise rotation* in the complex plane, from the head entity h to the tail entity t [53]. This relationship can be mathematically expressed as shown in eq. (2.14).

$$
\boldsymbol{t} = \boldsymbol{h} \circ \boldsymbol{r}, \quad \text{where } |r_i| = 1. \tag{2.14}
$$



(a) TransE models r as translation in real line.

(b) RotatE models r as rotation in complex plane.

(c) RotatE: an example of modeling symmetric relations r with $r_i = -1$

Figure 2.9: Figure from [53]: *Illustrations of TransE and RotatE with only one dimension of embedding*

Here, $\circ$ denotes the element-wise product. Since $\boldsymbol{r}$ comprises several elements, each element $r_i \in \mathbb{C}$. As the length of r is 1, we arrive at the complex formula $e^{i\phi_{r,i}}$. This expression represents a counterclockwise rotation of r_i, $\phi_{r,i}$, in the complex plane, implying that the entity embeddings in the complex vector space will solely influence the rotation phase.

RotatE employs the following scoring function shown in eq. (2.15).

$$d_r(\boldsymbol{h}, \boldsymbol{t}) = ||\boldsymbol{h} \circ \boldsymbol{r} - \boldsymbol{t}|| \tag{2.15}$$

This work uses the scoring function eq. (2.15) in conjunction with the MRL objective function.

Neural Network-based methods

CompGCN CompGCN (Composition-based Graph Convolutional Networks) is a neural network-based method [17] that combines the power of graph convolutional networks (GCNs) with the compositional structure of KGs. It was introduced in 2020 in the paper "Composition-based Multi-Relational Graph Convolutional Networks" by Vashishth et al. [54].

We define a multi-relational graph $\mathcal{G} = (\mathcal{V}, \mathcal{R}, \mathcal{E}, \boldsymbol{\mathcal{X}})$, where $\mathcal{V}$ is the set of vertices, $\mathcal{E}$ is a set of edges, $\mathcal{R}$ is a set of relations, and $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{|\mathcal{V}| \times d_0}$ represents d_0-dimensional input features of each node in the GCN [54] (note that *nodes* now describe the neural network layers, and not the graph). Further, we denote the node representation in a single

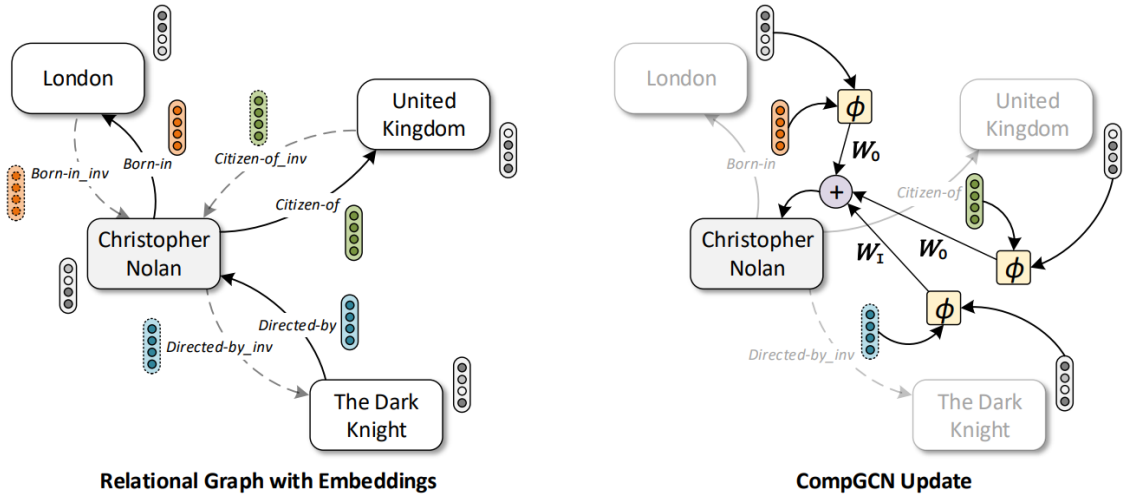


Figure 2.10: Figure and description borrowed from original CompGCN paper [54] : "*Overview of CompGCN. Given node and relation embeddings, CompGCN performs a composition operation $\phi(.)$ over each edge in the neighbourhood of a central node (e.g. Christopher Nolan above). The composed embeddings are then convolved with specific filters $\boldsymbol{W}_O$ and $\boldsymbol{W}_I$ for original and inverse relations, respectively. We omit self-loop in the diagram for clarity. The message from all the neighbors is then aggregated to get an updated embedding of the central node. Also, the relation embeddings are transformed using a separate weight matrix. [54]*"

GCN layer as $\boldsymbol{H} = f\left(\hat{\boldsymbol{A}}\boldsymbol{\mathcal{X}}\boldsymbol{W}\right)$, where $\hat{\boldsymbol{A}} = \tilde{\boldsymbol{D}}^{-\frac{1}{2}}(\boldsymbol{A}+\boldsymbol{I}))\tilde{\boldsymbol{D}}^{-\frac{1}{2}}$ is a normalized adjacency matrix with added self-connections, and $\tilde{\boldsymbol{D}}$ is $\tilde{\boldsymbol{D}}_{ii} = \sum_j (\boldsymbol{A}+\boldsymbol{I})_{ij}$. The function f denotes some activation function, and $\boldsymbol{W} \in \mathbb{R}^{d_0 \times d_1}$. Lastly, we define k as the number of layers in the model, such that for each layer $\boldsymbol{W}^k \in \mathbb{R}^{d_k \times d_{k+1}}$ is the current layer's parameters, and the root is $\boldsymbol{H}^0 = \boldsymbol{\mathcal{X}}$. Then, we define each layer as

$$\boldsymbol{H}^{k+1} = f\left(\hat{\boldsymbol{A}}\boldsymbol{H}^k\boldsymbol{W}_r^k\right). \tag{2.16}$$

Equation (2.16) can be rewritten into a sum equation, which defines the model's scoring equation, shown in eq. (2.17).

$$\boldsymbol{h}_v = f\left(\sum_{(u,r)\in\mathcal{N}(v)} \boldsymbol{W}_{\lambda(r)}\phi(\boldsymbol{x}_u, \boldsymbol{z}_r)\right) \tag{2.17}$$

In eq. (2.17), $\mathcal{N}(v)$ refers to the set of immediate neighbours of node v for its outgoing edges, and the function ϕ performs composition of neighbour node u with respect to r. The purpose of ϕ is to remove over-parameterization [54].

In simpler terms, with CompGCN, entities and relations are represented as node and edge embeddings in a graph as shown in fig. 2.10. The model applies a series of graph convolutional layers to learn the embeddings of the nodes and edges. The final embeddings are combined using a scoring function, and the model is trained to predict the likelihood of a triple using MRL.

2.2.3 Performance indicators for Knowledge Graph Embeddings

Performance indicators are crucial in evaluating the effectiveness of KGE models. However, assessing the learned embeddings' quality can be challenging because the embeddings cannot be directly observed. Therefore, performance indicators are necessary to quantitatively measure the accuracy and generalization capability of the embedding models. The performance indicators help compare different models and identify potential limitations of the models.

We must consider two parameters before utilizing the performance indicators; sidedness and types.

Sidedness Sidedness determines how we treat the triple-evaluation in the embedding:
Head - We only consider $(?, r, t)$.
Tail - We only consider $(h, r, ?)$.
Both - We consider both previous operations.

Types We have three types of interpretations,
Optimistic - the rank of a set of triples is the best rank among them.
Pessimistic - the rank of a set of triples is the worst rank among them.
Realistic - the average of the two previous.

Mean Rank MR measures the average rank of the correct object entity or relation when performing link prediction [3]. For each test triple (h, r, t), the model computes a score for all possible entities or relations and ranks them in descending order based on the score. The rank of the correct entity or relation is then averaged across all test triples to obtain the MR. A lower MR indicates better performance, meaning the correct entity or relation was ranked higher on average.

Formally, MR is defined as shown in eq. (2.18).

$$MR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \text{rank}^{Q_i}(h, r, t) \tag{2.18}$$

Mean Reciprocal Rank The Mean Reciprocal rank is similar to MR, except we limit the result x, such that $x \in [0, 1]$ [3]. We let the correct triple's rank be the denominator of the fraction. The MRR formula is shown in eq. (2.19).

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}^{Q_i}(h, r, t)}. \tag{2.19}$$

A correctly top-ranked triple will obtain the score $\frac{1}{1} = 1$. A second-placed correct triple will get a score $\frac{1}{2} = 0.5$. The final MRR is the average of all predicted triples.

We define the optimistic MRR as o-MRR. This is when we only consider the highest scoring rank. We will use this metric in this thesis.

Adjusted Mean Rank Index The Adjusted Mean Rank Index (AMRI) measures the mean rank over the expected mean rank [3]. The expectedness defines the *noise-rank*. The Noise-rank is the mean rank if we treat every prediction randomly. Then, we define this point as 0. The AMRI scales $AMRI \in [-1, 1]$. If the result is positive, it performs better than noise and worse if negative. Formally, the expected Mean Rank is defined as

$$\mathbb{E}(MR) = \frac{1}{2|Q|} \sum_{i=1}^{|Q|} (|S_i| + 1),$$

where S_i denotes the number of candidate fillers. This yields the final AMRI formula shown in eq. (2.20).

$$\text{AMRI} = 1 - \frac{MR - 1}{\mathbb{E}(MR - 1)} \tag{2.20}$$

Hits@k Hits@k (reads: Hits at k) measures the frequency with which the correct triples are ranked among the top-k scores generated by the model [3]. This metric is absolute, meaning it does not consider the knowledge graph's size. Consequently, it is unsuitable for direct comparisons between datasets of different sizes. Nevertheless, k where $k \in \{1, 3, 5, 10\}$ is often used as cut-off values. The final frequency is normalized, and we obtain a fraction in which triple ranks in the top-k.

2.3 Semantic Web Technologies: RDF and OWL

2.3.1 Understanding ontologies

The term *ontology* holds a broad spectrum of meanings across different domains. This thesis delves into the interpretation and usage of *ontology* within computational ontologies. As per "The Handbook on Ontologies" by Staab and Studer [51], an ontology is construed as "a specific type of information object or computational artefact". Consequently, an ontology is characterized as a model devised to structurally and formally systemize a set of observations, defining entities and their interrelations. Studer offered a widely accepted definition of ontologies in 1998: "An ontology is a formal, explicit specification of a shared conceptualization" [52]. The term 'conceptualization' captures an abstract, simplified representation of the world designed for a specific purpose [51]. This suggests that our conceptualization focuses solely on relevant aspects per our perception of the world (phenomena). Studer incorporated the term *shared* into his definition to mitigate discrepancies arising from individual perceptions. The final part of his definition, referring to a formal and explicit specification, outlines how this shared conceptualization should be interpreted to ensure a common understanding, accounting for potential variations in users' understanding of terms.

Given the comprehensive nature of ontologies, they serve many research purposes. Towards the end of the 1990s, the employment of ontologies for the 'Semantic Web' gained prominence [8]. The intent was to foster a common lexicon [51] among web agents, thereby facilitating a *shared* comprehension of information. An instance of an ontology could be a workplace, where entities comprise people, projects, and items, and relations serve to connect these entities.

Nevertheless, another interpretation of 'ontology' pertains to the specification of web resources in the Semantic Web. Consequently, an ontology can be expressed using DLs. What is known as the TBox in DLs aligns with our use of the term 'ontology' [51]. Indeed, an ontology can correspond to DLs KB.

2.3.2 Understanding RDF and RDFS

Tim Berners-Lee [8] introduced the concept of the *semantic web* in the early stages of the World Wide Web era. The primary objective of the semantic web was to facilitate knowledge sharing and enable the analysis and application of this knowledge. The information was expected to be human-readable while maintaining machine code to comprehend and process it. This led to the development of *semantic markups*. In 1999, the World Wide Web Consortium (W3C) recommended the use of the Resource Description Framework (RDF) as a standard for semantic markup. RDF is not a syntax but a data model designed to structure information. The syntax was XML, as depicted in the Semantic Web Stack in fig. 2.11. Today, this original combination is referred to as RDF/XML, given the emergence of various markups for RDF, such as Turtle, N3, Manchester Syntax, and others.

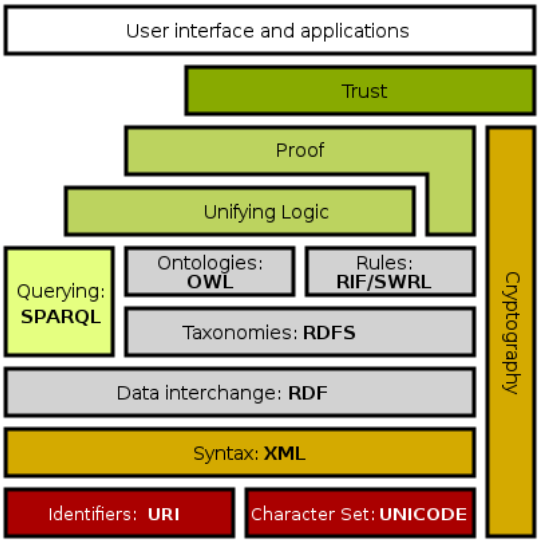


Figure 2.11: The Semantic Web

Annotation and meaning

An RDF triple, as defined by [30], takes the form of a subject, property, and object.

```
subject     property     object .
```

A collection of RDF triples is referred to as an *RDF Graph*. RDF could utilize XML to represent these triples, resulting in what is known as RDF/XML. The objects in the

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

ex:Dog rdf:type ex:Animal .
```

Listing 1: A simple RDF triple.

triple are typically Uniform Resource Identifiers (URIs), which identify resources on the web [15]. These URIs are frequently abbreviated with prefixes. For instance, as seen in listing 1, we declare that "a Dog is an Animal".

In the realm of ontologies, we have RDF Schema (RDFS), which introduces meta-properties to RDF to facilitate the representation of an ontology [51]. Some of the most notable metaproperties include rdfs:Resource, rdf:type, rdf:subClassOf, rdfs:Class, rdf:Property, rdfs:domain, and rdfs:range. These are interconnected via rdf:type and rdfs:subClassOf, as depicted in fig. 2.12, derived from W3C [13]. It is evident from fig. 2.12 that everything stems from rdfs:resource. The most prevalent meta-properties are briefly described in table 2.5. Furthermore, the RDF and RDFS can be explained by

- **Signature:** The RDF signature consists of the non-logical symbols used to denote entities, such as URIs and prefixes. For instance, in the RDF statement `ex:Dog rdf:type ex:Animal`, 'ex:Dog', 'rdf:type', and 'ex:Animal' are part of the signature, here 'ex' and 'rdf' are prefixes, while they also could use the complete URI `<http://www.w3.org/1999/02/22-rdf-syntax-ns>`;
- **Syntax:** The syntax of RDF/XML inherits XML syntax. It dictates the structure of well-formed triples, where a subject is linked to an object via a property. For instance, the triple "ex:Dog rdf:type ex:Animal" follows the syntax of RDF, where 'ex:Dog' (subject) is linked to 'ex:Animal' (object) via 'rdf:type' (property); and
- **Semantics:** The semantics of RDF provide real-world grounding for the triples. For instance, the triple "ex:Dog rdf:type ex:Animal" conveys the meaning that a Dog is an instance of an Animal in the real world. The interpretation function maps entities to the concepts they represent, creating a connection between the abstract RDF representation and the real-world concepts.

Meta Property	Description
rdfs:Resource	The foundational element in RDFS; all entities stem from it.
rdf:type	Designates an entity as an instance of a specific RDFS class or metaproperty.
rdfs:Class	Denotes that certain objects from the ontology can be classified as classes.
rdfs:domain	Restricts the subject of a triple to a specific class for a given property.
rdfs:range	Restricts the object of a triple to a specific class for a given property.
rdfs:subClassOf	Declares that the subject class is a subclass of the object class.
rdfs:subPropertyOf	Declares that the subject property is a sub-property of the object property.

Table 2.5: Key Meta Properties in RDF Schema (RDFS)

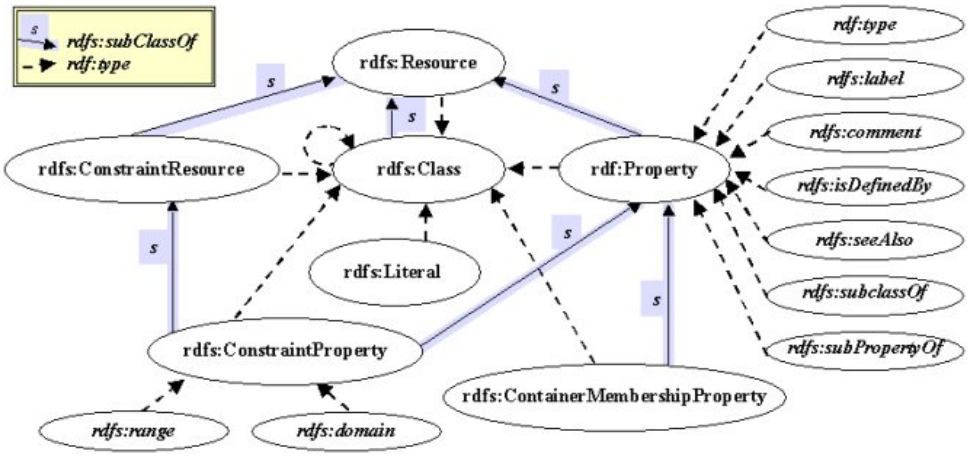


Figure 2.12: RDFS. Illustration from [13]

Listing 2 shows RDFS triples. Here we declare 'Dog' and 'Animal' as rdfs:Class, and 'hasPet' as a pdf:property. We then use rdfs:subClassOf to say that a 'Dog' is a subclass of 'Animal'. Lastly, we state that if hasPet is used in a triple, the object of that triple must be an 'Animal'. The last statement is achieved with rdfs:range.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
@prefix ex: <http://example.org/#>

ex:Dog rdf:type rdfs:Class .
ex:Animal rdf:type rdfs:Class .
ex:hasPet rdf:type rdf:Property .
ex:Dog rdfs:subClassOf ex:Animal .
ex:hasPet rdfs:range ex:Animal .
```

Listing 2: Example of RDFS triples

2.3.3 Exploring OWL

In the semantic stack, positioned above RDFS (refer to fig. 2.11), we find the OWL, an evolution of the original OWL. OWL extends RDF and RDFS, preserving similar syntax [51].

While the original version of OWL comprised three sublanguages: *OWL Full*, *OWL DL*, and *OWL Lite*, OWL 2 adopted a different approach. It introduced profiles, namely *OWL 2* $\mathcal{EL}$, *OWL 2* $\mathcal{QL}$, and *OWL 2* $\mathcal{RL}$, each designed to cater to specific use cases and reasoning requirements.

Among these profiles, OWL 2 $\mathcal{EL}$ is tailored explicitly for applications necessitating large ontologies, and it ensures polynomial-time reasoning complexity. The various profiles of OWL 2 each support a specific set of features and restrictions to cater to its target use cases. Generally, all OWL 2 profiles generally allow for the use of classes, properties (both object and data properties), and individuals, offering a more expressive vocabulary than RDF and RDFS. Our work primarily adopts OWL 2 $\mathcal{QL}$, as it maps nicely to queries and *DL-Lite*. It supports inclusion assertions using PerfectRef, which is often required in our context. OWL 2 $\mathcal{RL}$, by contrast, lack support for existential quantifiers on the right-hand side of an axiom [55]. Henceforth, whenever we refer to the acronym OWL, we are explicitly referencing W3C's OWL 2 $\mathcal{QL}$ Profile from December 2012 [55]. Any reference to another profile of OWL 2 or a previous version of OWL will be explicitly mentioned.

Our OWL profile, like other profiles, provides an enriched vocabulary, including the following:

- *Vocabulary Partitioning*:
  - *owl:Class*: An extension of rdfs:Class, in OWL it allows the definition of disjoint and equivalent classes within a new class instance using *owl:disjointWith* and *owl:equivalentClass*. Two predefined classes, *owl:Thing* and *owl:Nothing*, are always present in OWL 2.
  - *owl:DataProperty*: These properties pertain to datatype values.
  - *owl:ObjectProperty*: These properties relate to objects, or instances of classes. They can use rdfs:domain and rdfs:range, and may have inverse properties, *owl:inverseOf*, which are also ObjectProperties. Furthermore, OWL allows property restrictions via *owl:Restriction*.
- *Property Separation*: The sets of object properties and data properties must be disjoint.

In OWL, ontology definitions consist of axioms and facts. An *axiom* provides a statement that is considered to be true within the ontology. For instance, an OWL axiom could be:

```
#Example of an \gls{owl} Axiom
ex:hasPet rdfs:range ex:Animal .
```

On the other hand, *facts* provide specific details about an individual, including the classes the individual belongs to, and the properties and values associated with that individual [45]. Here is an example of an OWL fact:

```
#Example of an \gls{owl} Fact
ex:theo rdf:type ex:Dog .
```

This fact indicates that 'theo' is an instance of the class 'Dog'.

To illustrate the transition from RDFS to OWL, refer to fig. 2.13.

Overall, the introduction of OWL has further enhanced RDFS by offering more expressive capabilities. Different profiles catering to various use cases and reasoning requirements enable a more nuanced and powerful representation of ontologies. This, in turn, facilitates the creation of complex and meaningful semantic relationships.

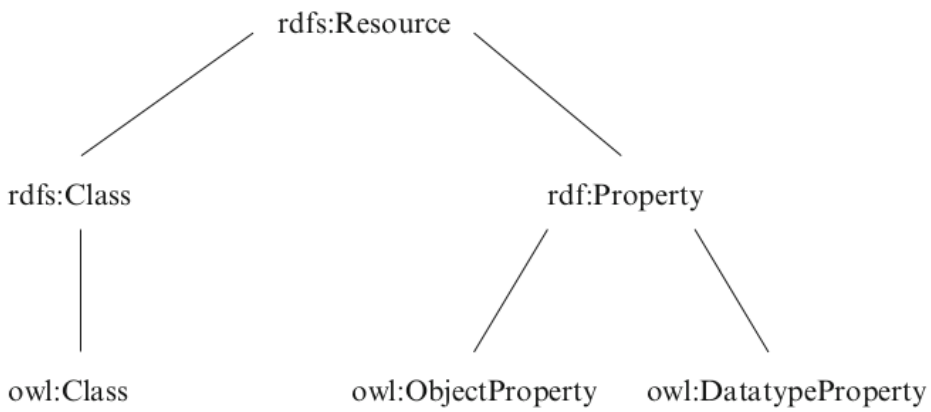


Figure 2.13: The Transition from RDFS to OWL. Source: [51]

2.3.4 Relationship between OWL and DL

The association between OWL variants and DLs, especially *DL-Lite*, constitutes a cornerstone of semantics within the Semantic Web. Grasping this correlation is crucial for performing entailment and deriving new statements or triples.

Within the DL context, the axioms set in OWL reflect the TBox, while the OWL's facts set aligns with the ABox in DL, which holds specific instances or facts about individuals.

A pivotal part of this relationship is the mapping of DL concepts and roles to OWL classes and properties. In DL, a concept parallels an OWL class, while a DL role corresponds to an OWL property.

When mapping DL to OWL, a DL concept becomes an instance of `rdf:type` in OWL. The triple's object should be the concept (or class) to which the subject belongs. For example, given a DL concept 'Animal' and 'Dog' as an instance of 'Animal', this relationship can be depicted in OWL as the triple: `ex:Dog rdf:type ex:Animal`.

Roles in DL map naturally to properties in OWL. For instance, if there is a role 'hasPet' in DL, it can be represented as an OWL property. If 'john' has a pet 'Dog', this relationship is expressed in OWL as: `ex:john ex:hasPet ex:Dog`.

For a more comprehensive look at the terminology mapping, refer to table 2.6. The mapping of assertions and descriptions are further illustrated in fig. 2.14 and fig. 2.15.

The synergy between OWL and DL, particularly *DL-Lite*, is advantageous as it integrates the expressivity of OWL with the reasoning capabilities of DL. *DL-Lite* is

Abstract Syntax	DL Syntax
owl:Thing	$\top$
owl:Nothing	$\bot$
intersectionOf(C_1 C_2 ...)	$C_1 \sqcap C_2$
unionOf(C_1 C_2 ...)	$C_1 \sqcup C_2$
complementOf(C)	$\neg C$
oneOf(o_1 ...)	$\{o_1, \ldots\}$
restriction(R someValuesFrom(C))	$\exists R.C$
restriction(R allValuesFrom(C))	$\forall R.C$
restriction(R hasValue(o))	$R : o$
restriction(R minCardinality(n))	$\geq n\, R$
restriction(R minCardinality(n))	$\leq n\, R$
restriction(U someValuesFrom(D))	$\exists U.D$
restriction(U allValuesFrom(D))	$\forall U.D$
restriction(U hasValue(v))	$U : v$
restriction(U minCardinality(n))	$\geq n\, U$
restriction(U maxCardinality(n))	$\leq n\, U$

Figure 2.14: OWL DL descriptions. Illustration from [51]

Abstract Syntax	DL Syntax
Class(A partial C_1 ...C_n)	$A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$
Class(A complete C_1 ...C_n)	$A = C_1 \sqcap \ldots \sqcap C_n$
EnumeratedClass(A o_1 ...o_n)	$A = \{o_1, \ldots, o_n\}$
SubClassOf(C_1 C_2)	$C_1 \sqsubseteq C_2$
EquivalentClasses(C_1 ...C_n)	$C_1 = \ldots = C_n$
DisjointClasses(C_1 ...C_n)	$C_i \sqcap C_j = \bot, i \neq j$
DatatypeProperty(U super(U_1)...super(U_n)	$U \sqsubseteq U_i$
domain(C_1) ...domain(C_m)	$\geq 1\, U \sqsubseteq C_i$
range(D_1) ...range(D_l)	$\top \sqsubseteq \forall U.D_i$
[Functional])	$\top \sqsubseteq \leq 1\, U$
SubPropertyOf(U_1 U_2)	$U_1 \sqsubseteq U_2$
EquivalentProperties(U_1 ...U_n)	$U_1 = \ldots = U_n$
ObjectProperty(R super(R_1)...super(R_n))	$R \sqsubseteq R_i$
domain(C_1) ...domain(C_m)	$\geq 1\, R \sqsubseteq C_i$
range(C_1) ...range(C_l)	$\top \sqsubseteq \forall R.C_i$
[inverseOf(R_0]	$R = ({}^{-} R_0)$
[Symmetric]	$R = ({}^{-} R)$
[Functional]	$\top \sqsubseteq \leq 1\, R$
[InverseFunctional]	$\top \sqsubseteq \leq 1\, R^{-}$
[Transitive])	$Tr(R)$
SubPropertyOf(R_1 R_2)	$R_1 \sqsubseteq R_2$
EquivalentProperties(R_1 ...R_n)	$R_1 = \ldots = R_n$

Figure 2.15: OWL Axioms and facts. Illustration from [51]

FOL	**DL**	**OWL**
Constant	Individual/Constant	Individual
Unary Predicate	Concept	Class
Binary Predicate	Role	Property
	Concept and role assertions	Axioms
	Membership assertion	Fact / instance of class
	TBox	Set of axioms
	ABox	Set of facts
	Knowledge base = TBox + ABox	Knowledge base = Ontology

Table 2.6: Relationship between terms in FOL, OWL and DL

especially suitable for this purpose, as it strikes an optimal balance between expressiveness and computational efficiency, essential for large-scale KBs typical of the Semantic Web. We can execute complex inferencing tasks more efficiently by employing the *DL-Lite* framework along with OWL. This relationship between DL and OWL is essential for addressing complex query answering. Together, they form the query rewriting foundation we use to query the KGEs.

The following chapter will explore how various elements - logic, knowledge graph embeddings, and semantic web aspects - interweave to form a comprehensive strategy for complex query answering.

Chapter 3

Combining Query Rewriting with Knowledge Graph Embeddings for Complex Query Answering

This chapter delves into integrating query rewriting and knowledge graph embeddings, exploring the potential of combining these two approaches to enhance complex query answering tasks. We begin by outlining the strategy behind our approach and explaining the rationale and benefits of combining query rewriting with knowledge graph embeddings. Next, we provide a detailed description of our pipeline design, illustrating the components and processes involved in the integration. By the end of this chapter, we will have implicitly addressed our first research question, shedding light on how query rewriting can be effectively integrated with knowledge graph embeddings to improve complex query answering performance.

3.1 Strategy: Why It Works

The primary motivation for introducing query rewriting into complex query answering is its ability to efficiently handle TBox information without resorting to computationally expensive methods, such as full ontology entailment to achieve completeness. By leveraging query rewriting, we can obtain the same query answers without having to entail new statements in the ontology beforehand, leading to more efficient and scalable solutions for query answering.

One of the critical benefits of query rewriting is that it enables us to handle concepts in the KB effectively. In a standard KG, triples representing concepts are rare, as they typically require the property to be rdf:type and the object to be a concept. However, with a well-defined TBox containing domain and range constraints, we can easily query for concepts, which are then rewritten into roles. Roles in *DL-Lite* have a straightforward mapping to the relations in the set of triples the KGE is trained upon, making roles more applicable to a conventional KG.

Query rewriting also complements KGEs by enhancing link prediction capabilities. As we train KGEs models on a set of triples, they inherently excel at predicting missing information in an incomplete KG. By combining query rewriting with KGEs, we can ensure that we cover all possible answers by expanding our query over the TBox while also benefiting from the link prediction capabilities of KGEs.

However, one limitation of our approach lies in the potential for too many rewritten queries when dealing with extensive TBoxes. As the output of query rewriting can be enormous for large ontologies, this may result in highly time-consuming predictions, as each rewritten query must be parsed. Consequently, the current approach is better suited for smaller TBoxes where the number of rewritten queries is 'manageable'.

For instance, in the case of the 'Human' concept in WikiData, the concept could be rewritten to thousands of other subclasses, leading to an overwhelming number of queries to process. This limitation should be considered when applying our method to large-scale knowledge graphs with complex and extensive TBoxes.

In summary, our approach efficiently handles TBox information through query rewriting and combines it with the link prediction strengths of KGEs. Query rewriting allows us to avoid expensive entailment processes to answer a query. It ensures that we cover all possible answers while making the most of KGEs' capacity to predict missing information in the KG. By integrating query rewriting and KGEs, we create a powerful and efficient solution for complex query answering tasks that address the challenges of incomplete KGs and effectively handle concepts and roles within the TBox. However, it is vital to keep in mind the limitations of our approach when dealing with large TBoxes, as the number of rewritten queries may become a bottleneck for performance.

3.2 Pipeline: How it works

In this section, we will delve into the details of our proposed method, providing a comprehensive explanation of how the different components of our pipeline work together. We will discuss the implementation of PerfectRef, our query generator, and the complex query answering pipeline, which collectively contribute to the effectiveness of our approach. To illustrate the interconnectivity of these components, we will walk through some example queries, demonstrating how the queries are processed and transformed at each step of the pipeline. By the end of this section, we will gain a deeper understanding of the inner workings of our method and how it successfully integrates query rewriting with KGEs. As a visual aid, we have prepared an abstract schematic representation of the entire pipeline, depicted in fig. 3.1.

3.2.1 Our Implementation of PerfectRef

As mentioned in section 2.1.3, PerfectRef is an algorithm designed for query reformulation. In this thesis, we have implemented this algorithm in Python, making it an integral part of our pipeline[1]. Before delving into the specifics of our implementation, its features, and limitations, we will first provide an overview of its dependency library, owlready2 [35].

It is important to note that while discussing our Python implementation, we may encounter terminological overlaps with previously defined terms. In object-oriented programming, the terms *Classes* and *objects* are commonly used. To avoid confusion, we will use the term *Object* when referring to classes in object-oriented programming. In contrast, *class* will retain its RDFS definition and remain synonymous with concepts in DLs.

owlready2

The Python library *owlready2* was developed in 2017 by Lamy [34], following his paper "Owlready: Ontology-oriented programming in Python with automatic classification and high-level constructs for biomedical ontologies" [34]. Our PerfectRef implementation relies on this library for three primary purposes:

[1] `https://github.com/AImenes/perfectref`

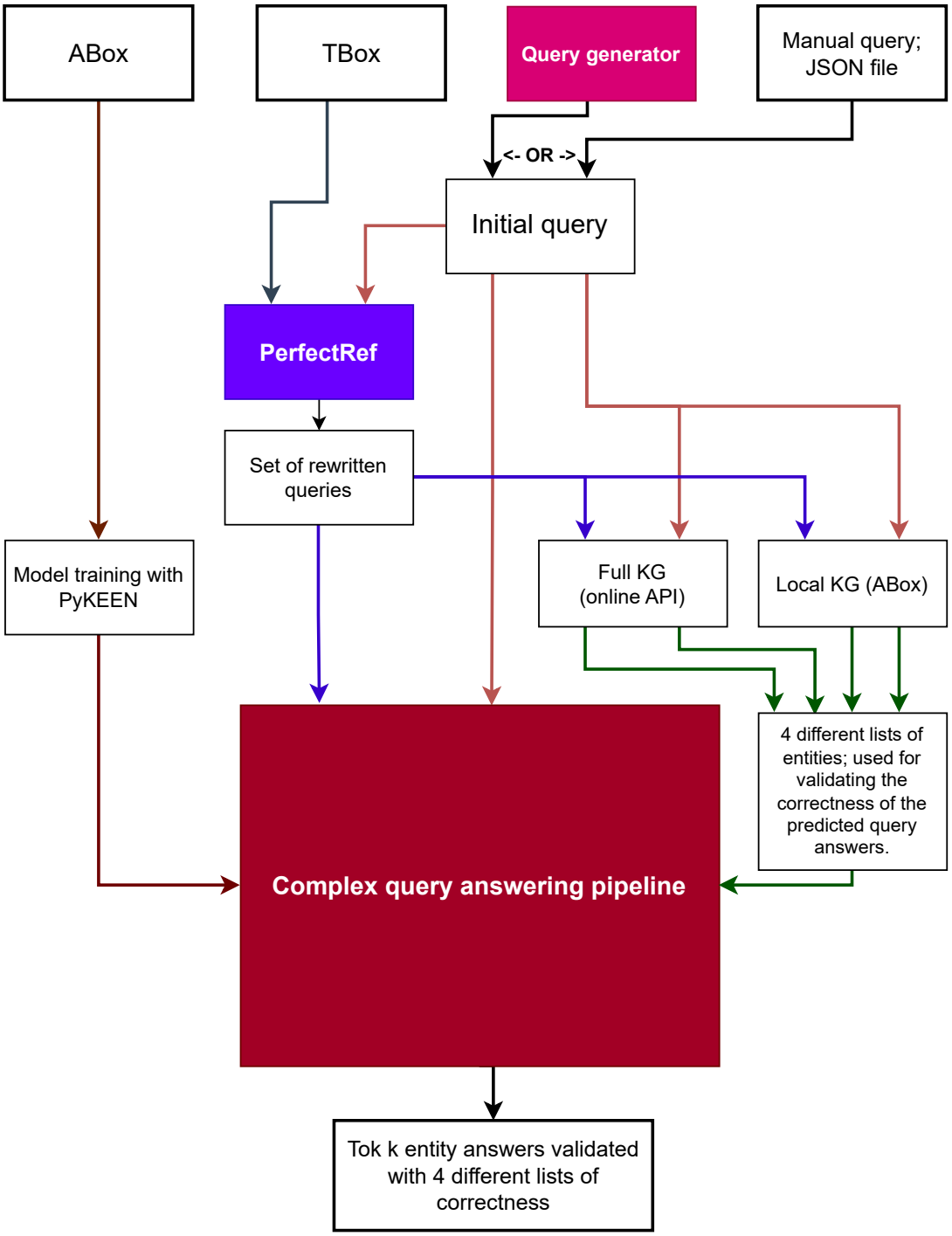


Figure 3.1: Our pipeline for integrating query rewriting with complex query answering using knowledge graph embeddings

1. Importing an OWL ontology;
2. Extracting axioms from the ontology; and
3. Utilizing its objects for the atoms in the axioms.

Importing an OWL ontology *owlready2* is capable of importing ontologies in RDF/XML, OWL/XML, or NTriples formats, and it automatically detects the format during import [34]. If an ontology is written in different syntax, it can be serialized to a compatible format using ontology software like Protégé[2] or frameworks such as OWLAPI[3] (Java) or RDFLib[4] (Python). However, this would have to be a manual process.

Extracting axioms After importing an ontology, *owlready2* allows access to all concepts and roles (referred to as classes and properties in the owlready2 documentation [36]). Iterating through these elements provides the following information:

For a specific atomic concept A, it yields

- a set of superclasses;
- a set of subclasses;
- a set of restrictions; and
- a set of equivalences.

The restriction pertinent to our study is known as the *SOME*-restriction [37]. This restriction implies that an atomic concept A is subsumed by an atomic role P, denoted as $A \sqsubseteq \exists P$ or $A \sqsubseteq \exists P^-$.

For a specific atomic role P (property), it yields

- a set of superproperties;
- a set of subproperties;
- a set of domains; and
- a set of ranges.

While the Owlready2 library produces a broader range of axioms beyond those mentioned, we have chosen to mention those particularly pertinent to our use case.

[2] `https://protege.stanford.edu/`
[3] `https://github.com/owlcs/owlapi`
[4] `https://github.com/RDFLib/rdflib`

Objects PerfectRef utilizes several objects from owlready2, including 'ThingClass', 'ObjectPropertyClass', 'Inverse', and 'Restriction'. While owlready2 contains other objects, such as 'EquivalentClass' and 'NotClass', these objects are not implemented in our implementation of PerfectRef.

Structure of PerfectRef

Before executing the PerfectRef algorithm, we must prepare and format the data required for its operation. This preparation process consists of four steps.

1. *Importing an ontology*: In this step, we import the ontology used as a basis for query rewriting.

2. *Extracting iterable axioms (TBox)*: After importing the ontology, we extract a set of iterable axioms (i.e., the TBox) from it. PerfectRef will use these axioms during the query rewriting process.

3. *Query parsing*: In this step, we analyze the input query to identify its atoms and determine the state of the variables (i.e., whether they are distinguished, shared, bound, or unbound).

4. *Synchronizing query with the ontology*: Once the query has been parsed, it is synchronized with the ontology by incorporating the relevant URIs into the parsed query.

Importing the ontology and extracting axioms First, we import the ontology using owlready2's owl parser, as shown in listing 3 on the second line. Next, we extract the axioms from the ontology with the *getaxioms* method, which iterates over every class and property in the OWL ontology. For a specific class (atomic concept) A, it extracts the following axioms:

- a set of Superclasses A_{super}. This yields axioms $A \sqsubseteq A_s$, where $A_s \in A_{super}$;
- a set of Subclasses A_{sub}. This yields axioms $A_s \sqsubseteq A$, where $A_s \in A_{sub}$; and
- a set of restrictions. The SOME-restriction [37] on an atomic role P yields $A \sqsubseteq \exists P$ or $A \sqsubseteq \exists P^-$.

For a specific property (atomic role) P,

```
def get_entailed_queries(ontology, string):
    onto = import_ontology(ontology)
    t_box = get_axioms(onto, True)

    q = parse_query(string)
    q_head = q.head
    q_body = q.body

    # get IRI and namespace
    get_iri_and_namespace(q,onto)

    PR = perfectref(q_body, t_box)

    #Exporting the results
    print_query(PR, string, q_head)
    return PR
```

Listing 3: The main method in the PerfectRef Python Library

- a set of Superproperties P_{super}. This yields role inclusion axioms $P \sqsubseteq P_s$, where $P_s \in P_{super}$;
- a set of Subproperties P_{sub}. This yields role inclusion axioms $P_s \sqsubseteq P$, where $P_s \in P_{sub}$;
- a set of domains A_{domain}. This yields axioms $\exists P \sqsubseteq A_d$, where $A_d \in A_{domain}$; and
- a set of ranges A_{range}. This yields axioms $\exists P^- \sqsubseteq A_r$, where $A_r \in A_{range}$.

We define an object, 'LogicalAxiom', which has variables *left-hand-side* and *right-hand-side*. These variables correspond to the two sides of the inclusion assertion $B \sqsubseteq C$. The axioms extracted, as mentioned earlier, are instances of this object. The method then returns this set of axioms.

The Query Parser and URI mapping The query parser expects a string as input, fulfilling the following rules:

- Each variable should start with the symbol '?';
- The circumflex symbol should separate each atom in the query, '^'; and
- The head and body of the query should be separated by ':-'.

We show some examples of parsable queries below.

$$q(?x) \text{ :- } Student(?x)$$
$$q(?x) \text{ :- } Student(?x) \cap hasTutor(?x, ?y)$$
$$q(?x) \text{ :- } Student(?x) \cap hasTutor(?x, ?\_) \cap hasTutor(?x, ?\_)$$

The parser will recurse down a chain of parsers until it reaches each variable in each atom and, in the end, return a complete query where the state of each variable is identified. The structure of objects in this method is shown in fig. 3.2.

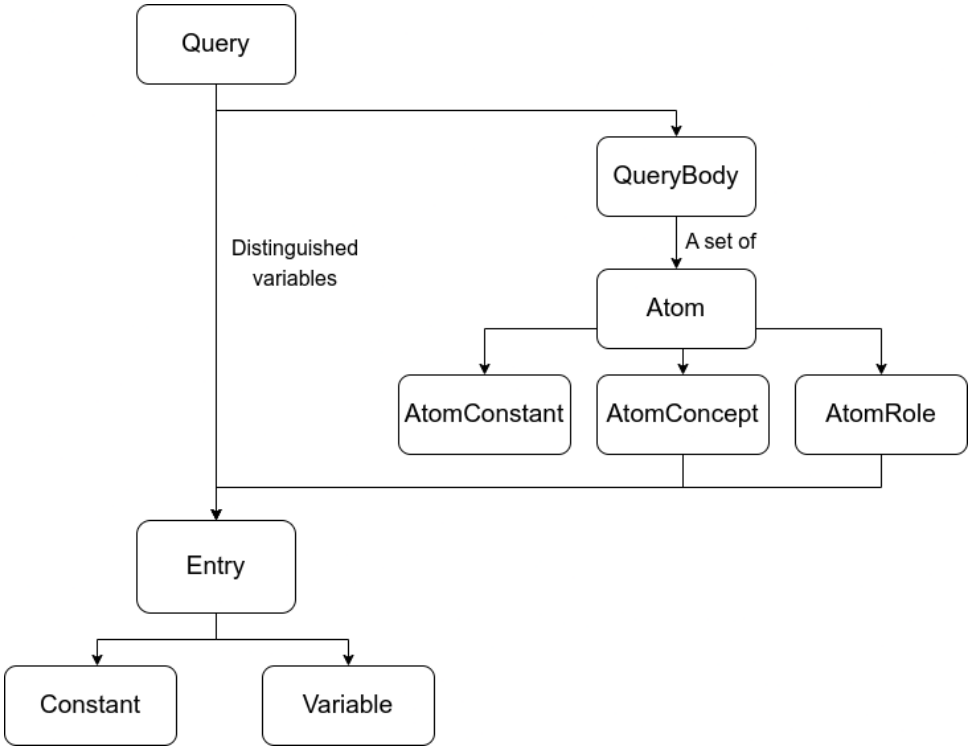


Figure 3.2: The objects in our PerfectRef query parser

Synching atoms and URIs After the query has been parsed, obtaining the URIs is initialized. This maps the written atoms to actual concepts and roles in the KB. If there are several concepts or roles with the same predicate, it asks from which namespace we refer. Now, we have the tools to run the actual algorithm.

The Core Logic of PerfectRef Implementation

The primary execution loop of our PerfectRef implementation strictly adheres to the workflow drafted in section 2.1.3. This subsection clarifies the variations from the main

algorithmic loop as laid out in the pseudo-code. It is crucial to remember that the algorithm identifies a query rewriting if an atom within the query is applicable to the TBox, or if any group of atoms in the query is reducible, as depicted in fig. 3.3.

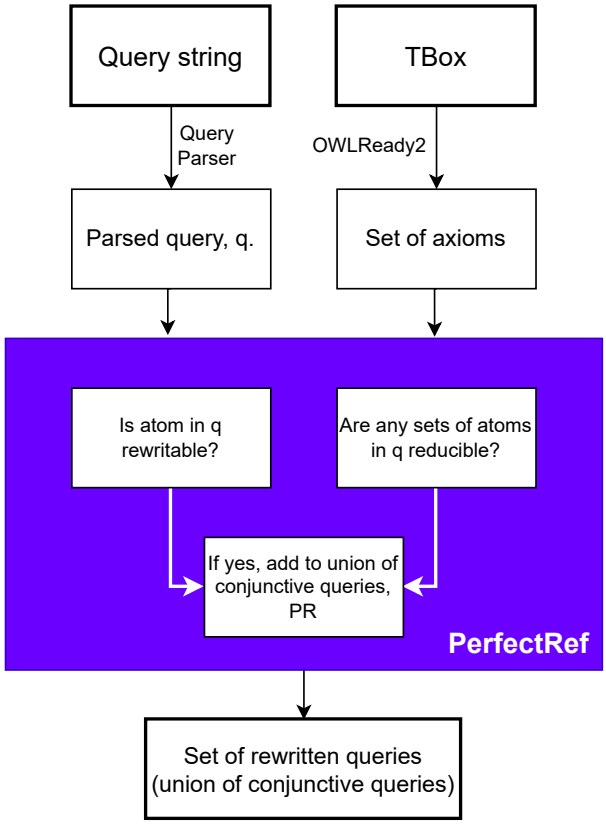


Figure 3.3: PerfectRef Implementation simplified overview

Our implementation introduces several enhancements to optimize the algorithm's execution time. An example is including a check to determine if a CQ has been processed previously. This check, illustrated in listing 4, prevents the algorithm from redundantly re-evaluating the same CQ each time PerfectRef reiterates its execution.

Moreover, we have implemented a mechanism to ascertain whether a newly entailed (in this context, reformulated) CQ already exists within PerfectRef, as shown in listing 5. This mechanism effectively eliminates duplicates from the resulting union of CQs.

In section 2.1.3, we introduced the method τ, which updates the variables post-execution of the *reduce* method. In our implementation, the τ function is incorporated within the 'reduce' method, streamlining the process.

Finally, to further optimize the performance, we have integrated an optional restriction that sets an upper limit on the number of rewritings.

```
1
2  [...]
3
4  #For every query in the list
5  for q in PR_prime:
6
7      #Check if the algorithm already processes the query
8      if not q.is_processed():
9
10 [...]
11
```

Listing 4: Snippet from the implementation of PerfectRef where the atom checks if is already processed

```
1
2  [...]
3
4  #Construct the new query
5  new_q = new_query(q, g, PI)
6
7  #If the query is not already entailed from previous processes, nor it is None
8  if not (new_q in PR or new_q is None):
9
10    # Add a new query to PR
11    PR.append(new_q)
12
13 [...]
14
```

Listing 5: Snippet from the implementation of PerfectRef where duplicates are identified

For a comprehensive technical exploration of our PerfectRef implementation, please refer to the GitHub repository[5]. The running time of our implementation is explained in appendix B.

3.2.2 Query answering from a Knowledge Graph Embedding

Although PerfectRef can handle query reformulations, a pipeline for query answering is still required. Therefore, we have developed a framework that can answer concepts and roles based on a specific set of parameters that we will introduce in the following section.

Training embeddings; PyKEEN

PyKEEN (Python KnowlEdge EmbeddiNgs) is a Python library for training and evaluating KGE models, developed by Ali et al. [3]. It comprehensively implements various embedding techniques and allows easy experimentation and model comparison. PyKEEN includes a variety of pre-implemented embedding models, including all five models we utilize in this work (TransE, DistMult, BoxE, RotatE, CompGCN). It also includes a range of evaluation metrics, like MR, MRR, AMRI and Hits@k. PyKEEN is built on top of the PyTorch deep learning framework [44] and is designed to be scalable and efficient. It also includes tools for hyperparameter tuning, early stopping, and saving and loading models.

Answering a CQ within Our Framework

To effectively handle and answer CQs within our framework, we have developed a dedicated pipeline that classifies queries based on their structure and a variable hierarchy.

[5] `https://github.com/AImenes/perfectref`

Classification of Query Structures A query within this framework can be categorized as a projection, intersection, union, or combination, namely a EPFO query. Recall that PerfectRef only handles CQs. Therefore, this framework will reformulate disjunctive queries to a union of CQs, which PerfectRef understand.

We specifically employ nine standardized query structures that have been defined in related complex query answering literature [49, 48, 23, 4, 6].

These nine distinct query structures are graphically represented in fig. 3.4. Each structure is denoted using a shorthand notation involving letters and a number. The letters represent the type of operation — (p)rojection, (i)ntersection, or (u)nion — while the number signifies the count of atoms in the query. For instance, '2p' signifies a projection query containing two atoms.

It should be noted that '1p' is a distinct case. Although it technically denotes a projection operation, it effectively involves just one atom and, thus, applies equally to intersection and union operations. It implies a single-atom prediction.

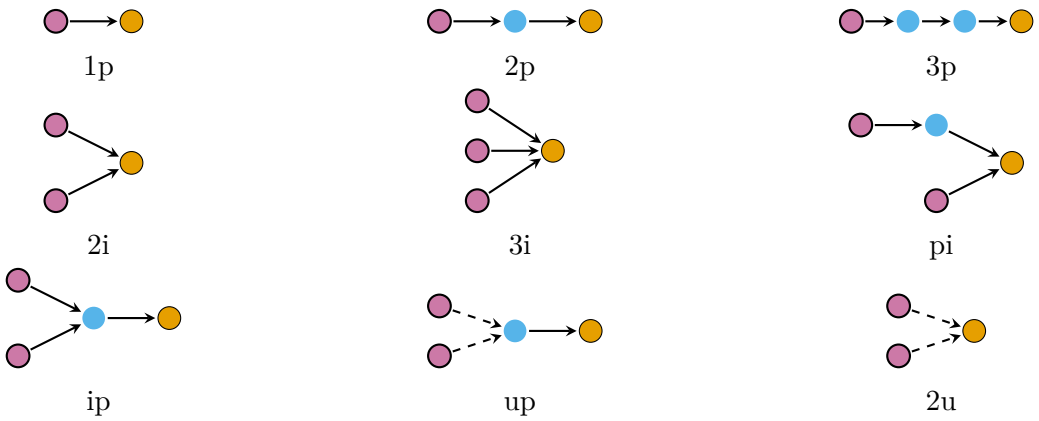


Figure 3.4: Queries using (p)rojection, (i)ntersection, and (u)nion

For efficiency, we have established specific terms and practices within our framework. We have restricted our attention to variables only, excluding constants. This stipulation allows many predictions for different query structures to commence with the same prediction. For example, consider the '2p' and '3p' queries:

$$2p : q(?w) : -hasFather(?x, ?y) \cap\ hasMother(?y, ?w) \tag{3.1}$$

$$3p : q(?w) : -hasFather(?x, ?y) \cap\ hasFather(?y, ?z) \cap\ hasSibling(?z, ?w) \tag{3.2}$$

The first atom prediction for both queries would be identical in this scenario. We term such instances as *base cases*. By our definition, a *base case* is either the first occurring role in a query (meaning it should have an unbound variable), where either the head or tail is the target, or it is a concept prediction.

Our pipeline initially identifies all base cases and carries out these predictions. Subsequently, it performs projection, intersection, or union (disjunction) operations using base case predictions. This approach substantially reduces time complexity by avoiding a multitude of overlapping predictions. However, this method is not applicable if the query contains constants.

In the following paragraphs, we elaborate on the detailed pipeline for handling projection, intersection, and disjunction operations.

Determining the Variable Hierarchy The previous section introduced nine standard query structures. However, it is essential to note that our framework is not confined to these nine structures. It is possible to execute more complex structures, such as '9p' or 'pipi', and so forth. However, for the sake of the thesis, we are only handling the previously defined structures.

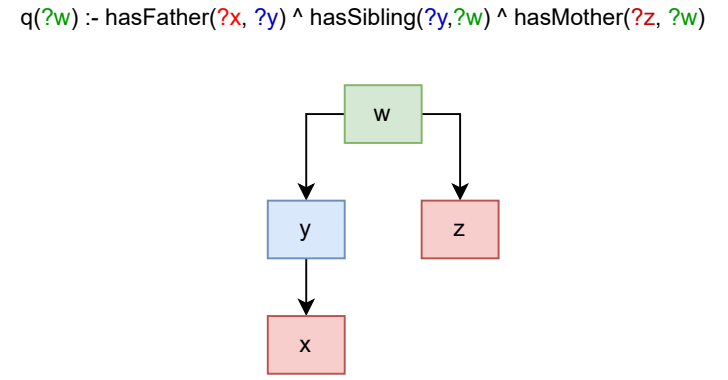


Figure 3.5: Variable hierarchy in a 'pi'-query

Before launching any prediction pipelines or identifying base case predictions, our system establishes the variables' hierarchy within the query. This process enables the identification of the base cases relevant to the current query.

If we represent the variables in a query as a tree—where the root node is the distinguished variable and the leaves are the unbound variables, as the 'pi'-query illustrates in red in fig. 3.5—a base case is identified as any atom role containing an unbound variable (colored red) or any concept that houses this variable.

Without an unbound variable within the query, the system iterates strictly based on depth.

We construct a tree structure for each query for the variables and store the depth alongside each variable. The distinguished variable at the root position is assigned a depth of zero.

This method determines the order in which predictions will be made. This implies the query does not need to be pre-ordered, as this hierarchy will identify the prediction sequence. Taking the example in fig. 3.5, the system will first predict 'hasFather' with the tail as the target and 'hasMother' with the tail as the target, as they contain a leaf variable (unbound) and are classified as base cases. Subsequently, it will predict the next atom with the 'highest' depth (i.e. farthest from the distinguished variable; the higher the number, the greater the depth). This logic ensures that our final prediction will always be the concluding prediction, regardless of whether the last step is an intersection, projection, or union.

This approach allows us to handle any query structure. We always identify the base cases (leaves) and work our way up according to the depth until we reach the distinguished variable. The specifics of how intersections, projections, and disjunctions will be processed along this journey will be discussed in the following sections.

Base Case Prediction Pipeline We define a *base case* in our context as either:

1. a concept; or
2. a role featuring exactly one unbound variable.

We employ a global set of base cases, denoted as B. This set is saved to a pickle file in Python, eliminating the need to recompute base case predictions each time the framework is run. We use the variable hierarchy to identify the base case atoms in the current query. If these predictions exist in B, the corresponding entity outputs and their scores are retrieved from B. If they do not exist, we generate this base case prediction and store the result in B. The base case prediction can either be a concept or a role:

Concepts A concept prediction is only feasible in our framework if the ABox also includes the property 'rdf:type'. Although having triples with 'rdf:type' in the ABox is not typically the case, with query rewriting, our queries can include concepts: the queries can be rewritten into a role prediction by leveraging domain and range axioms, and use the role pipeline described in the subsequent paragraphs.

In the rare case of an ABox that includes the RDF Schema (usually, the TBox and ABox are defined as distinct sets), our framework handles it as follows:

1. We first establish set X as a collection of instances associated with the desired atomic concept A.
2. If $X = \emptyset$, no prediction is made for the desired concept, and we return empty.
3. Otherwise, we proceed to define:

$$X_{head} = \{\ (h, r, t) \quad | \quad (h, r, t) \in W \mid\ h \in X\ \}$$
$$X_{tail} = \{\ (h, r, t) \quad | \quad (h, r, t) \in W \mid\ t \in X\ \}$$

4. For Heads,
   - 4.1. We define $R_{head} = \{\ \ r \quad | \quad (h, r, t) \in X_{head}\ \ \}$.
   - 4.2. We execute a top-k-link prediction using the embedding, with the tail as the target. The ensuing set is denoted as P. That is, $P = pred(X_{head}, R_{head}, \_\ )$.
   - 4.3. Leveraging the instances in P and relations R_{head}, we perform a top-k-link prediction to derive a set of heads H, i.e., $H = pred(\ \_\ , R_{head}, P)$.
5. For Tails,
   - 5.1. We define $R_{tail} = \{\ \ r \quad | \quad (h, r, t) \in X_{tail}\ \ \}$.
   - 5.2. We execute a top-k-link prediction using the embedding, with the head as the target. The ensuing set is denoted as P. That is, $P = pred(\ \_\ , R_{tail}, X_{tail})$.
   - 5.3. Leveraging the instances in P and relations R_{tail}, we perform a top-k-link prediction to derive a set of tails T, i.e., $H = pred(P, R_{tail}, \_\ )$.
6. As a form of verification (or filter), we conduct a final triple-scoring prediction to obtain scores on the already constructed triples. We denote $Z = (H \cup T)$. In practice, we construct triples in the format $(h, \texttt{rdf:type}, A) \quad | \quad h \in Z$, predict their score and sort them accordingly.

7. The resulting entities and their corresponding scores are stored in the base-case python dictionary.

Roles A base case role prediction occurs when it contains an unbound variable or an atom with a variable with the maximum depth value (i.e., farthest from the distinguished variable).

1. We define the current role atom, $r = P$.
2. Using r, we construct the sets:

$$X_{head} = \{ \ h \ \mid \ (h, r, t) \in W \mid \ r = P \ \}$$
$$X_{tail} = \{ \ t \ \mid \ (h, r, t) \in W \mid \ r = P \ \}$$

3. For roles $\exists x \mid P(x, \_)$,
   3.1. We execute a top-k-link prediction $H = pred(\ \_\ , r, X_{tail})$.
   3.2. The top-k entities and their scores are stored in the base case set, B.
4. For roles $\exists x \mid P(\_, x)$,
   4.1. We execute a top-k-link prediction $T = pred(X_{head}, r, \ \_\ )$.
   4.2. The top-k entities and their scores are stored in the base case set, B.
5. For roles $\exists x \mid (\exists y(P(x, y)))$,
   5.1. We perform both steps above, merging the results and scores.
   5.2. The top-k entities and their scores are stored in the base case set, B.

It is noteworthy that a role base case can exist in two distinct states. It can either have the head as the target or the tail. Each of these instances is a separate prediction, meaning a base case prediction for a role could have multiple states in the global base case set, B.

General Remarks In the global base case set, B, merely storing the prediction results is insufficient. Technically, this set operates as a dictionary where the key is the comprehensive parameter data for the current prediction. In other words, the model and its parameters, the cut-off prediction value, and the target in the triple serve as the key, whereas the base case prediction entities and their corresponding scores are the values of that instance in the dictionary. This distinction is crucial because scoring varies across different models, and scores are not directly comparable between models. The scores only provide valid comparisons within predictions executed with identical configurations.

Projection Pipeline For each query, we construct a new Python dictionary. The keys of this dictionary are the variables, and their associated values are the entities and their scores. After predicting or extracting all the base cases in the current query from the base case set, B, we map these results to their corresponding variable names in the dictionary. As we traverse the atoms in the query, this dictionary allows us to identify when we are performing a projection:

1. If the current atom is a role and both variables are bound, we execute a projection. We consult the variable hierarchy to determine which variable is closest to the distinguished variable, thereby establishing the target variable for this specific atom.
2. With this information, we can ascertain the input for the prediction. As all base cases are covered, if we query the input variable in the dictionary, it should yield a set of entities. We use this set of entities as the input for predicting the target variable.
3. We store the top-k predicted entities in the dictionary, using the target variable as the key. Once the newly predicted entities are identified, we update their scores by applying a T-norm to the score of the input entity and the new prediction. This operation involves multiplying the scores, as they always lie between 0 and 1 due to applying the Sigmoid function for score calibration.

Note: If the current atom is a concept, it cannot be used for projections.

Intersection Pipeline When we encounter an atom with a target variable for which entities already exist in the dictionary, we need to perform an intersection:

1. We first identify the entities associated with the variable and outputs of the current atom prediction.

2. For entities that exist in both, we compute the T-norm between them, which involves multiplying their scores in our framework.

3. The scores are calibrated using the Sigmoid function, ensuring they always lie between 0 and 1. Entities not in the new prediction results and the dictionary are discarded (as per definition, they do not intersect).

Disjunction Pipeline For disjunctions, the crucial steps occur prior to the prediction pipeline, making a disjunctive query into a union of glsplcq:

1. Each disjunctive query, like 'up' and '2u', can be rewritten into a union of CQs. Therefore, we identify a query as a disjunction and then rewrite it as a union of CQs involving only projections and intersections.

2. Finally, we merge the results from these CQs by computing the T-conorm between the entities and their scores.

3. If we have overlapping entities output from the CQs, the T-conorm preserves the one with the highest score, our chosen T-conorm.

Validation Procedure After prediction for all the atoms in a query is finished, we reference our dictionary for the distinguished variable. The linked entities and their corresponding scores provide the query response.

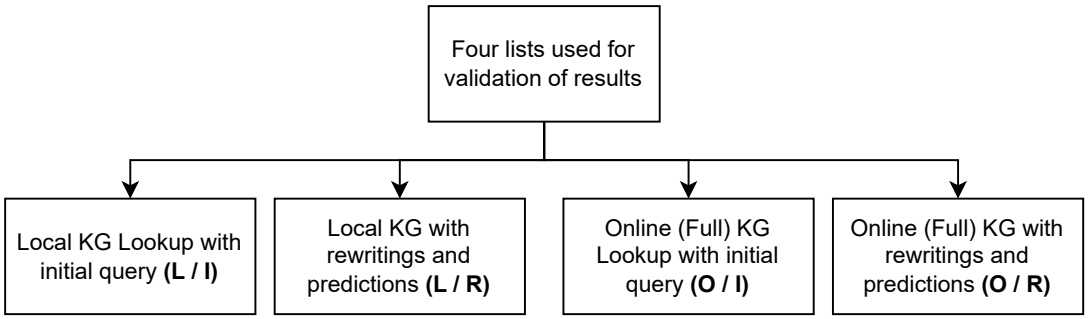


Figure 3.6: The four different lists used for validation of the correctness of predicted entities

Our processing pipeline consistently outputs predictions yielding the top k entity answers. To assess the accuracy and validate the correctness of these predictions, we

rely on four distinct lists of entities achieved from standard KG for our respective query. These lists are

- entity answers using our initial query to query the dataset our embeddings is trained upon (L/I);
- entity answers using our union of rewritten queries to query the dataset our embeddings is trained upon (L/R);
- entities obtained from a KG lookup to the full, online dataset, unfamiliar to the embedding models, using the initial query (O/I); and
- entities sourced from a KG lookup to the full, online dataset, unfamiliar to the embedding models, using the rewritten queries (O/R).

The four lists, depicted in fig. 3.6, allow us to verify if the predicted entities appear in these lists. If so, the prediction is validated correctly for that entity. When mentioning *validation* of the results using these lists, the term *validation* should not be confused with the validation phase used for finding our preferred parameters for our KGE models.

In this work, we used the entirety of our datasets for model training; thus, we use the online versions of our datasets to assess prediction accuracy. When we use our local version for validation (L/I and L/R), we essentially compare our results against facts known to the model during training. This intentional choice helps answer our research question regarding how our approach fares against a standard KG lookup. A standard KG lookup generates a list of entities, making it challenging to compare against a framework incorporating query rewriting and KGE predictions. Hence, when evaluating results with L/I, we establish a *baseline score* for a specific query response. In other words, we have verified the correctness of the prediction against the entities achieved from a standard KG lookup. To assess the impact of rewriting, we will contrast the scores obtained using L/R for validation against the results evaluated using L/I.

Similarly, we can compare the online KG results by juxtaposing O/R with O/I to perceive the influence of rewriting. When gauging the impact of KGE predictions, we compare online KG validation against our local dataset validations, i.e., L/I against O/I and L/R against O/R. As our embedding models do not see the online KG during training, they are used to ascertain if any predicted entities are accurately predicted without being recognized as correct by our local datasets. This is an alternate approach to the more traditional approach in complex query answering training; training our model on a subset of the test set. A vital advantage of this approach is the maximization of dataset utilization during training.

The predicted entities are ranked according to their final score, adjusted through T-norms and T-conorms throughout the pipeline. With a quantifiable baseline score, we can technically compare O/R against L/I to ascertain any improvements. Similarly, we can compare L/I to L/R and O/I to O/R to determine the effect of rewriting and L/I to O/I and L/R to O/R to evaluate the impact of predictions.

The validation serves a CWA approach. If the online KGs do not acknowledge an entity prediction, it will be validated as false. However, one could also treat the results with an OWA approach, but then it would be more challenging to measure the improvements by introducing query rewriting.

Final Analysis Each pipeline processes a single query. Therefore, if a query has three rewritings, the entire procedure is executed four times - once for the original and three times for the rewritings. Upon completing all rewriting predictions, their respective output prediction data frames are consolidated (i.e. unified) into a single data frame. In cases where several rewritings have predicted an entity, only the highest score (T-conorm) is retained. Remember that for the written results data frame, we utilize lists L/R and O/R for validation, while for the initial query results, we employ lists L/I and O/I.

Each query iteration records metrics for L/I, L/R, O/I, and O/R. Additionally, we calculate the Hits@k and o-MRR for all four entity lists used for validation.

For queries that do not offer any possible rewritings, the rewritten validation lists will equate to the list from the initial query. Conversely, when potential rewritings exist, these lists may be more extensive, and the final consolidated list for validation can pinpoint additional entities. This feature is the core goal of this work. The data from the initial query is also found in the rewritten list of entities; thus, ideally, the rewritten entity list used for validation should always be equal to or larger than the set from rewritings, and it should ideally not contain any entities that the rewritten set does not encompass.

One of the principal strengths of this framework lies in PerfectRef's capability to simplify a query in accordance with the TBox. For instance, a '3i' query could be reduced to a '2i' query if permitted by the TBox. The results would still contribute to the '3i' score in such instances, as the initial query is a '3i' structure. This ability to employ the TBox to decomplexify a query arguably is one of the most significant advantages of the framework.

The Query Generator

The query generator is the last essential component of our implementation. This generator produces random queries to facilitate a fair framework assessment, making cherry-picking queries impossible. However, generating these random queries is more complex than it may appear due to the inherent structure and distribution of entities and relations in the ABox. Most ontologies do not exhibit a uniform distribution of entities and relations, leading to the strength of the vector representations of relationships and entities varying in the embedded space. Furthermore, the ABox may not contain corresponding data for specific queries, rendering them non-answerable. Our query generator employs a frequency-based approach to be more fair towards the ABox.

The generator starts by conducting a frequency count across the ABox. These frequencies serve as probability weights (a general example in fig. 3.7) in subsequent query generation, ensuring the queries reflect the actual content of the ABox accurately. In cases where the TBox contains additional roles and concepts not present in the ABox, the generator assigns them a frequency count of one, such that they still can be randomly picked upon generation. This designation guarantees their potential inclusion in the generated queries while still giving preference to more frequently occurring entities and relations in the ontology.

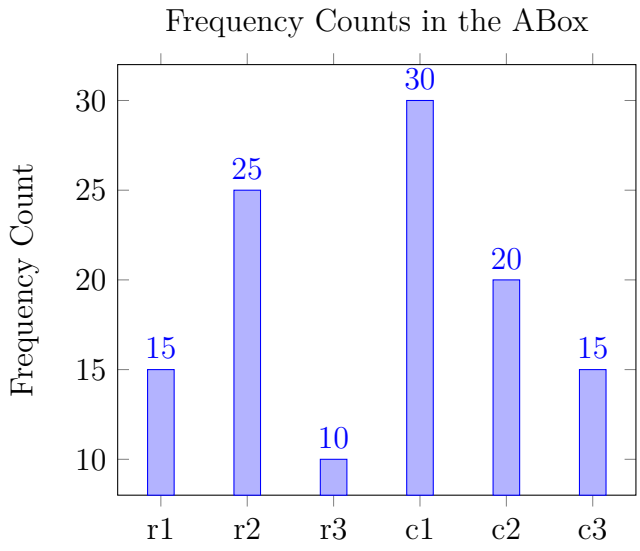


Figure 3.7: A general example of a frequency count of 3 relations and 3 concepts.

A key point to note with our generator is excluding RDFS meta properties from the query generation process. The rationale for this is twofold. Firstly, using these properties would require the query object to be a concept or another restricted rather

than an individual, complicating the query generation process. Secondly, these meta properties often do not contribute meaningfully to the query, as they could be a meta property for comments, aliases or other such properties.

The decision to develop our query generator stems from limitations we identified in existing generators, such as the one used by Query2Box[6]. This generator is designed to work with individuals, not concepts, and does not consider the TBox in query generation. As a result, it falls short in scenarios where the TBox plays a crucial role. However, this query generator will ensure it does not generate a query with no answers, which is a favourable attribute that might be our query generators biggest weakness.

It is crucial to acknowledge that while our query generator addresses the limitations mentioned earlier and offers the advantage of impartiality, it has drawbacks. The most significant challenge arises from the random selection of atoms based on their weights, which can lead to generating nonsensical, or *degenerated*, queries, particularly for longer queries.

For example, consider the following query

$$q(?w) \text{:-} Plant(?w) \cap\ hasChild(?x, ?w).$$

This query could have been generated due to a purely random selection using our weighted distribution. Although syntactically valid, it may not yield local and online KG hits. This query is an example of a degenerated query that combines unrelated concepts or roles, resulting in no corresponding data in the ABox. Unfortunately, our query generator lacks a mechanism to identify and prevent the generation of such unanswerable or degenerated queries, a feature found in the generator used by Query2Box. This limitation was detected in the final stages of the thesis timeline, leaving no room for the development of a viable solution.

Despite these limitations, our query generator serves as a valuable and unbiased tool for generating a diverse range of queries that accurately reflect the content of the ABox. It is designed to produce as many queries as the user desires for each predefined structure, providing significant flexibility and adaptability in query generation. However, it is essential to note that the performance of the query generator is highly dependent on

[6] `https://github.com/snap-stanford/KGReasoning`

the nature of the ontology it is applied. While it may perform efficiently with smaller TBoxes, it might struggle with larger ones as the probability of degeneration increases. Additionally, the structure of the ontology can also influence the performance, with more 'clique-like' ontologies likely reducing the generation of degenerated queries.

Therefore, it is fair to assume that the query generator's performance will vary based on the ontology used, and detailed performance analysis should consider this factor. Nonetheless, the generator's ability to produce a wide range of queries, coupled with its unbiased approach, makes it a significant contribution to the work and will still assist in answering our research questions unbiasedly.

3.2.3 Comprehensive Examples

To offer a holistic understanding of our implementation, we have chosen a subset of WikiData that focuses explicitly on family relations. Further details regarding this dataset will be explored in the forthcoming result chapter.

Our objective here is to highlight the dual advantages offered by the processes of rewriting and knowledge graph embedding predictions.

The first example demonstrates a scenario where rewriting improves the result. In contrast, the second instance illustrates how new facts can be discerned using KGE models. Subsequently, we present an example where both techniques - rewriting and prediction - work in unison to discover new entities.

Illustrating benefit of rewriting

We select an intersection query of two atoms for this demonstration, represented by the structure '2i'. Test case extracted from GitHub[7].

`q(?w) :- Father(?w) ∩ hasChild(?w,?x)`

We base our expectations for this run on an analysis of the ontology. We note that the concept *Father* only exists in the TBox and not in the ABox, while the property hasFather resides in the ABox. The TBox contains an 'rdfs:range' statement:

[7] `https://github.com/AImenes/query-answering-and-embeddings/blob/main/testcases/001/queries/family/k-1/every_structure/family-RotatE-dim192-epoch24-k100-2i-1.json`

`hasFather rdfs:range Father`. We anticipate that the rewriting process will yield a result considering this information.

This CQ could have been manually entered or generated using the query generator, as shown in fig. 3.1. We initiate the experiment by inserting the query into a JSON file encoded for WikiData, selecting the manual approach.

```
{
  "2i": [
  "q(?w) :- <https://www.wikidata.org/wiki/Q7565>(?w) ^
  ↪  <https://www.wikidata.org/prop/P40>(?w,?y)"
  ]
}
```

Our framework then performs a query (KG lookup) towards our local dataset using the initial query to obtain the list of entities that are valid responses (L/I). As anticipated, it returns an empty set since Q7565 (Father concept) is absent in our local WikiData subset ontology.

The next step involves rewriting using our PerfectRef implementation [8]. As expected, the rewriting process substitutes the Father concept with the hasFather role using the 'rdfs:range' axiom from the TBox. The rewriting results are as follows:

```
{
  "original":
  "q(?w) :- Q7565(?w)^P40(?w,?x)",
  "entailed":
  "q(?w) :- P22(?_,?w)^P40(?w,?x)"
}
```

Before proceeding, the query may seem redundant. If the first atom was Parent instead of Father, it could have also been reduced by our TBox, eliminating the need for the second atom. However, in the context of our TBox for this example, the redundancy is not detectable due to insufficient description. This missed reduction highlights the importance of a well-designed TBox.

[8] `https://github.com/AImenes/PerfectRef`

Following the rewriting step, each CQ in the union of CQs undergoes a KG lookup (in this case, only one), generating a consolidated list of entities where the entities satisfy at least one (union) of the rewritten CQs. Our framework produces a list of entities by taking the intersection between the output of hasFather (where the tail is the target) and hasChild (where the head is the target). This result forms the list of entities that will be used for validating the rewritten queries (L/R). We now have L/I and L/R validation lists.

We then transition to our complex query answering pipeline for predicting entities on the queries using KGEs. For this example, we employ a trained RotatE model on this ABox. The pipeline allows for 'in-situ' training if we lack a trained model.

In this example, the initial query returns an empty set as it lacks any candidates to predict the concept since our example ontology does not contain any RDFS meta-properties in the ABox, a common trait in most ontologies. However, we establish the variable hierarchy with the rewritten query, illustrated in fig. 3.8.

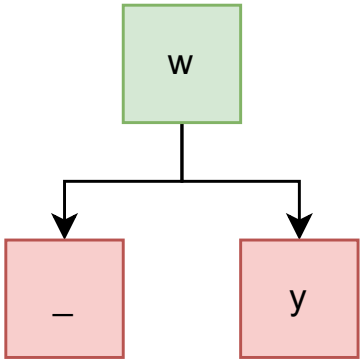


Figure 3.8: Variable hierarchy for the rewritten query

Both atoms are defined as base cases because they contain an unbound variable. The system predicts the 'P22' property with the tail as the target and then predicts 'P22' with the head as the target, storing both results in our base case dictionary. Since both atoms have the same depth to the variable farthest from the distinguished variable, the order in the query remains unchanged.

The pipeline activates, selecting 'P22' with the target tail and checking in the base case dictionary for its existence. It does exist, and these entities are stored in a separate dictionary mapped to the distinguished variable '?w'. The next atom is selected and found in the base case dictionary. These entities are selected and stored in the dictionary mapped to ?w. However, entities from the previous atom already exist here, triggering the intersection pipeline. Both lists of entities are compared, and identical entities are

identified and stored. The T-norm (multiplication in this case) is performed on the respective scores.

With all atoms traversed, we focus on the dictionary mapping variables to entities, selecting the Python DataFrame for the distinguished variable. We validate these final prediction entities with L/I and L/R. If the predicted entity is in the list, it returns true for that validation column in the DataFrame. The L/I validation yields our baseline score. Subsequently, we turn to the online KG version of WikiData. The framework automatically generates a SPARQL query based on the initial query and the set of rewritten queries. We receive two lists of entities from the online KG, which we can use to validate novel predictions unknown to our local datasets and embedding models. These lists are O/I and O/R, respectively. If the predicted entities exist in these lists, it returns true for that entity in their validation columns. This process repeats for every entity in the prediction, and we add a corresponding boolean column to the predicted entities. In the end, each entity has been validated by the four lists. The validation with L/I forms our baseline score, which we will compare with the other validation methods later.

This process is repeated for every CQ in our union of CQs (i.e. the set of rewritten queries). In this example, there is only one rewritten query. The final entity result for this initial query is a data frame merged result for all rewritings, including the initial query. The final entity prediction results are shown in table 3.1. Note that the Origin column identifies from which query (initial or rewritten) the entity was derived. If an entity has been the answer to several rewritings, it only stores the highest-scoring one. The score column is the T-norm between the scores of each atom in this specific case.

Interestingly, the initial query also returns false on the online KG (O/I). This is confirmed by querying the online API[9] for `?x rdf:type wd:Q7565`, which only returns four entities across the entire WikiData KG, being

- wd:Q30639365;
- wd:Q117228538;
- wd:Q117711034; and
- wd:Q118267469.

Therefore, all the validation using O/I yields no hits.

[9] `https://query.wikidata.org/`

Entity Label	Origin	Score	L/I	L/R	O/I	O/R
Q22020049	q(?w) :- P22(?x,?w) ∩ P40(?w,?y)	0.3275	false	true	false	true
Q280794	q(?w) :- P22(?x,?w) ∩ P40(?w,?y)	0.3268	false	true	false	true
Q6680740	q(?w) :- P22(?x,?w) ∩ P40(?w,?y)	0.3161	false	true	false	true
Q2426687	q(?w) :- P22(?x,?w) ∩ P40(?w,?y)	0.3128	false	true	false	true
Q934662	q(?w) :- P22(?x,?w) ∩ P40(?w,?y)	0.3078	false	true	false	true
Q637214	q(?w) :- P22(?x,?w) ∩ P40(?w,?y)	0.3078	false	true	false	true
Q723703	q(?w) :- P22(?x,?w) ∩ P40(?w,?y)	0.3029	false	true	false	true
Q4797766	q(?w) :- P22(?x,?w) ∩ P40(?w,?y)	0.2983	false	true	false	true
Q1136276	q(?w) :- P22(?x,?w) ∩ P40(?w,?y)	0.2971	false	true	false	true
Q3157533	q(?w) :- P22(?x,?w) ∩ P40(?w,?y)	0.2967	false	true	false	true
Q2714574	q(?w) :- P22(?x,?w) ∩ P40(?w,?y)	0.2965	false	true	false	true
Q721491	q(?w) :- P22(?x,?w) ∩ P40(?w,?y)	0.2954	false	true	false	true
Q488312	q(?w) :- P22(?x,?w) ∩ P40(?w,?y)	0.2951	false	true	false	true
Q12886175	q(?w) :- P22(?x,?w) ∩ P40(?w,?y)	0.2920	false	true	false	true
Q827448	q(?w) :- P22(?x,?w) ∩ P40(?w,?y)	0.2873	false	true	false	true
Q21062428	q(?w) :- P22(?x,?w) ∩ P40(?w,?y)	0.2861	false	true	false	true
Q3434614	q(?w) :- P22(?x,?w) ∩ P40(?w,?y)	0.2860	false	true	false	true
Q2381218	q(?w) :- P22(?x,?w) ∩ P40(?w,?y)	0.2859	false	true	false	true

Table 3.1: The entity output of the query example: Q7565(?w) ∩ P40(?w,?y). Results from GitHub.

Finally, we calculate Hits@k and the o-MRR using our four validation lists: L/I, L/R, O/I, and O/R. For this demonstration, we will use Hits@3. As we have a hit in the first position, the results are as follows:


```
{
    "hits@3":
    {
        "local_kg_hit_rewriting": 1.0,
        "online_kg_hit_rewriting": 1.0,
        "local_kg_hit_original": 0.0,
        "online_kg_hit_original": 0.0
    },
    "mrr":
    {
        "local_kg_hit_rewriting": 1.0,
        "online_kg_hit_rewriting": 1.0,
        "local_kg_hit_original": 0,
        "online_kg_hit_original": 0
```

```
    }
}
```

Our hypothesis was correct: the framework could not answer the initial query could, but with query rewriting, we obtained correct prediction results validated by L/R and the list of entities from the online version of WikiData O/R using the rewritten queries.

It is important to note that our metrics are not typically calculated for a single query, as we have done here. These metrics are averaged across every initial query in a test case to achieve an o-MRR where the denominator is more significant than one. The same argument applies to Hits@k. This approach ensures that our evaluation is robust, considering the system's performance on a range of queries rather than a single instance.

Illustrating benefit of KGE predictions

The previous example demonstrated how rewriting aids in discovering entities that satisfy the initial query. In this section, we illustrate an instance where the KGE predicts entities that may not be accurate based on the local KG lookup but are correct when validated with the online KG. Recall that using the list of entities achieved from the online KG is our method for verifying novel predictions by our embedding models.

To maintain simplicity, we have chosen a 1-predicate ('1p') query structure:
`q(?w) :- hasMother(?x,?w)`

This query is one of the 25 queries automatically generated by our query generator, which we will further discuss in the following chapter. The mentioned results are a subset extracted from these '1p' queries.

```
{
  "1p": [
  ...
  "q(?w) :- <https://www.wikidata.org/prop/P25>(?x,?w)",
  ...
  ]
}
```

In this scenario, the query has a singular rewriting to the concept `Mother(?w)` (corresponding to WikiData id: `Q7560`). Notably, no entities in the ABox are bound to this concept, implying that the rewriting step does not contribute to finding the answer.

The example undergoes the same pipeline process as the previous comprehensive example. Summarized, we perform rewriting, do our four KG lookups to achieve our list of entities used for validation, and match these against the prediction done by our embedding models. If an entity is present in a list from the online KG (O/I and O/R) and not in our local KG (L/I and L/R), then we have successfully pinpointed a correct entity prediction which the model was previously unaware of being true. As illustrated in table 3.3, we find four entities in the output list for this particular query. Complete query prediction results for this query can be found in our GitHub repository[10].

Entity Label	Origin	Score	L/I	L/R	O/I	O/R
Q1040807	q(?w) :- P25(?x,?w)	0.0058709825	false	false	true	true
Q104772	q(?w) :- P25(?x,?w)	0.0058709825	false	false	true	true
Q10315513	q(?w) :- P25(?x,?w)	0.0058709825	false	false	true	true
Q1010767	q(?w) :- P25(?x,?w)	0.0058709825	false	false	true	true

Table 3.3: The entity output of the query example: q(?w) :- P25(?x,?w)

Illustrating benefit of both

In this example, we examine a query where both prediction and rewriting are instrumental in identifying new entity answers to a query. This example is drawn from one of the 25 queries that will be discussed in the forthcoming test results[11].

The automatically generated query in question is denoted by

```
{
  "1p": [
  ...
  "q(?w) :- <https://www.wikidata.org/wiki/Q7566>(?w)",
  ...
  ]
}
```

[10] https://github.com/AImenes/query-answering-and-embeddings/blob/main/testcases/001/queries/family/k-25/every_structure/family-TransE-dim192-epoch24-k100-1p-7.json

[11] https://github.com/AImenes/query-answering-and-embeddings/blob/main/testcases/001/queries/family/k-25/every_structure/family-TransE-dim192-epoch24-k100-1p-8.json

which is mapped to `Parent(?w)`.

Initially, PerfectRef rewrites this query into four distinct CQs:

```
{
  "original":
    "q(?w) :- Q7566(?w)",
  "entailed":
    "q(?w) :- Q7565(?w)",
    "q(?w) :- Q7560(?w)",
    "q(?w) :- P22(?_,?w)",
    "q(?w) :- P25(?_,?w)",
}
```

Two rewritings pertain to Q7560 (Mother) and Q7565 (Father). These concepts do not have any 'rdf:type' in our ABox, so their predictions will invariably return empty. Nonetheless, utilizing the *rdfs:range* axioms in our TBox, we obtain rewriting to hasFather (P22) and hasMother(P25).

Each of the rewritten queries is sequentially processed through the framework. For the initial query, no entities are found in L/I. The online KG only identifies a single entity of type 'Parent', as depicted in fig. 3.9, yielding a very short O/I list.

In the final two rewrites, denoted by 'P22' and 'P25', we obtain hits from both the L/R and the O/R lists. Shown in table 3.4, 11 of those entities from the query prediction are uniquely recognized solely by the online KG via the rewritten queries (O/R). This recognization signifies the simultaneous utilization of prediction and rewriting, thus exemplifying the advantages of our approach.

Figure 3.9: Using the API to query for a parent. It returns one entity.

Entity Label	Origin	Score	L/I	L/R	O/I	O/R
Q1012807	q(?w) :- P22(?_,?w)	0.0125520453	false	false	false	true
Q101731	q(?w) :- P22(?_,?w)	0.0125520453	false	false	false	true
Q1014890	q(?w) :- P22(?_,?w)	0.0125520453	false	false	false	true
Q10066	q(?w) :- P22(?_,?w)	0.0125520453	false	false	false	true
Q101137	q(?w) :- P22(?_,?w)	0.0125520453	false	false	false	true
Q1000957	q(?w) :- P22(?_,?w)	0.0125520453	false	false	false	true
Q100246	q(?w) :- P22(?_,?w)	0.0125520453	false	false	false	true
Q1040807	q(?w) :- P25(?_,?w)	0.0058709825	false	false	false	true
Q104772	q(?w) :- P25(?_,?w)	0.0058709825	false	false	false	true
Q10305226	q(?w) :- P25(?_,?w)	0.0058709825	false	false	false	true
Q10315513	q(?w) :- P25(?_,?w)	0.0058709825	false	false	false	true

Table 3.4: Updated entity output of the query example: q(?w) :- Q7566(?w).

Chapter 4

Results

This chapter will present the experimental results and address the research questions raised in the introduction chapter.

4.1 Datasets

This study employs two distinct datasets containing ontologies. During the selection phase, our choices were constrained to datasets that explicitly included an ABox and a TBox. Having a TBox is a crucial requirement to utilize the PerfectRef tool effectively. We elected to incorporate an extract from DBPedia, named DBPedia15k, and an extract focused on family relations from WikiData. Both ontologies fulfil the prerequisite of containing clearly defined ABoxes and TBoxes, aligning with the structural requirements of our study. However, despite satisfying the basic requirements, these two chosen sets exhibit significant differences, which we will analyze and discuss in the following sections. This comparative analysis aims to elucidate the specific characteristics of each dataset and how these differences may influence the outcomes of our study.

For validation purposes in this study, we employ local and online KGs. The term *local* refers to the DBPedia15k and the Family dataset. These are the specific datasets on which our models have been trained. On the other hand, by 'online KGs', we refer to the full-scale DBPedia[1] and Wikidata[2] resources, both of which are readily accessible for SPARQL queries through their respective APIs. Notably, these online KGs are not utilized for training the models. Instead, they serve a crucial role in the validation phase, where they help determine the ability of our models to predict novel facts.

[1] {https://dbpedia.org/sparql

[2] https://query.wikidata.org/

4.1.1 DBPedia15k: Overview and Characteristics

The first dataset employed in this study is DBPedia15k, a subset of the expansive DBPedia knowledge graph, extracted from the work "Injecting Background Knowledge into Embedding Models for Predictive Tasks on Knowledge Graphs" [18] and its accompanying Git repository[3] [47]. The aforementioned is a slightly modified version compared to the original DBPedia15k. The original DBPedia15k was introduced by Liu et al. [39] and can be found on GitHub[4]. The changes in the modified version resemble the addition of TBox information [18].

The full DBPedia KG assimilates data from Wikipedia into an impressive 27 million triples. The online API is used for validation alongside DBPedia15k as our local KG.

DBPedia15k was curated as a manageable sample of the larger DBPedia, encompassing 12 800 entities, 1178 concepts and 278 roles, thereby resulting in a total of 180,000 triples. Uniquely, this dataset's TBox is separated into separate text files, providing ease of access and organization. We manually converted these files into an OWL ontology for the Python library *owlready2* to parse.

Unique Features and Challenges

An intriguing feature of the DBPedia15k dataset is the inclusion of 'rdf:type' triples within the ABox, which has implications for model training. The KGEs, in this case, treats the meta-properties in the ABox like any other property, thereby blurring the lines between standard properties and meta-properties in the embedded space. As a result, the 'rdf:type' meta-property occurs in more than half of the triple set in the DBPedia15k dataset, as seen in table 4.1.

The DBPedia15k ontology's TBox comprises 1178 concepts and 279 roles, but their distribution is not uniform. We observe that 110 out of 279 relations connect to fewer than ten triples, while the remaining relations appear thousands of times (refer table 4.1). This skewed distribution hampers the learning phase for the embedded vectors of many relations in the ontology, reducing their significance during backpropagation and weight optimization in model training.

[3] `https://github.com/Keehl-Mihael/TransROWL-HRS`
[4] `https://github.com/mniepert/mmkb/tree/master/DB15K`

Table 4.1: The most used properties in the family knowledge graph

IRI of properties	Frequency count
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	97,231
http://dbpedia.org/ontology/starring	7,122
http://dbpedia.org/ontology/birthPlace	6,117
http://dbpedia.org/ontology/genre	4,968
http://www.w3.org/2000/01/rdf-schema#seeAlso	4,321
http://dbpedia.org/ontology/country	2,890
http://dbpedia.org/ontology/occupation	2,805
http://dbpedia.org/ontology/associatedMusicalArtist	2,791
http://dbpedia.org/ontology/associatedBand	2,791
http://dbpedia.org/ontology/recordLabel	2,704
http://dbpedia.org/ontology/instrument	2,704
http://dbpedia.org/ontology/distributor	2,156
http://dbpedia.org/ontology/type	2,131
…	…

Concerning concepts, only 29 out of the 1178 TBox concepts are in use (see table 4.2), leaving 1149 concepts untrained in the selected prediction model. Interestingly, several concepts share entities, suggesting that rewriting might have a limited impact on this dataset. Moreover, it may rewrite into other concepts that lack an embedded representation.

Limitations

Certain limitations regarding the DBPedia15k dataset should be noted, which contains several logical inconsistencies. Examples include erroneous triples such as (Manchester United, rdf:type, (Economist), and (DiscoveryChannel, rdf:type, Island). These inaccuracies improperly categorize Manchester United FC as an economist and the Discovery Channel as an island. Such instances are quite prevalent throughout the dataset. It is crucial to underscore that these inconsistencies were not introduced in the modified version of the DBPedia15k but existed within the original DBPedia15k itself. The root cause behind these logical errors remains unclear, and we did not identify them until the analysis of our results began. The study also utilized the online DBPedia API[5] for validation of predictions, which is the entire DBPedia version. This version is free from these inconsistencies, as both the aforementioned erroneous examples return false through the KG API. Therefore, the disparities between the DBPedia15k dataset and the extended DBPedia version available online should be considered when evaluating our findings.

[5] https://dbpedia.org/sparql

Table 4.2: Frequency count of concepts in the knowledge graph

IRI of concepts	**Frequency count**
http://dbpedia.org/ontology/Island	15990
http://dbpedia.org/ontology/Place	7638
http://dbpedia.org/ontology/PopulatedPlace	7638
http://www.wikidata.org/entity/Q486972	7638
http://schema.org/Place	7638
http://dbpedia.org/ontology/Settlement	7638
http://dbpedia.org/ontology/Location	7638
NaturalPerson	2976
Agent	2976
http://xmlns.com/foaf/0.1/Person	2976
http://www.wikidata.org/entity/Q5	2976
http://www.wikidata.org/entity/Q215627	2976
http://dbpedia.org/ontology/Person	2976
http://schema.org/Person	2976
http://dbpedia.org/ontology/Agent	2976
http://dbpedia.org/ontology/Economist	2896
http://dbpedia.org/ontology/Governor	2472
http://dbpedia.org/ontology/Politician	1148
http://www.wikidata.org/entity/Q82955	1148
http://dbpedia.org/ontology/Species	1040
http://dbpedia.org/ontology/Planet	972
http://dbpedia.org/ontology/Architect	560
http://dbpedia.org/ontology/CelestialBody	458
http://dbpedia.org/ontology/Actor	246
http://dbpedia.org/ontology/Member_of_Congress	226
http://dbpedia.org/ontology/Astronaut	144
http://www.wikidata.org/entity/Q483501	122
http://dbpedia.org/ontology/Artist	122
http://dbpedia.org/ontology/Athlete	51

Advantages and Selection Rationale

Despite its flaws, DBPedia15k has particular strengths. It utilizes a wide TBox, enabling the exploration of PerfectRef's potential. Several queries are reducible and rewritable, demonstrating the reality of large ontologies where numerous URIs represent the same concept (synonyms). Moreover, its associated online API provides us access to the total DBPedia KG, which we use to validate our predictions' correctness in DBPedia15k.

The reason DBPedia15k was selected was its potential to train over TBoxes within a reasonable time frame. However, its distribution weakness and triple statement errors were undetected during selection. While considering these limitations when analyzing the results is critical, exploring this dataset remains a valuable learning experience.

4.1.2 The Family Dataset: A Wikidata5M Extract

The second dataset utilized in this study referred to as "The Family Dataset", was introduced by Jøsang et al. in a research work on rule mining in KGs [27]. This dataset represents a selection from the Wikidata5M KG, which initially comprises over 20 million triples, 5 million entities, and 800 properties.

Unique Features and Challenges

In contrast to DBPedia15k, The Family Dataset does not include any RDFS or OWL statements within its ABox. The distribution of triples, as demonstrated in table 4.4, is more evenly spread across this dataset. The extraction contains approximately 183,000 triples across seven properties. All URIs in the Wikidata extract are codified, with the corresponding mapping provided in table 4.3.

In addition to the local Family Dataset, this study also interfaces with the total WikiData Online KG via their API[6]. While the Family Dataset is a valid subset of the online KG, it does not include as many triples as the online KG, as will be illustrated in the results section.

[6] `https://query.wikidata.org/`

Table 4.3: TBox mapping in the Family Dataset

	IRI IDs	Mapping
Concept	https://www.wikidata.org/wiki/Q5	Human
	https://www.wikidata.org/wiki/Q1196129	Spouse
	https://www.wikidata.org/wiki/Q31184	Sibling
	https://www.wikidata.org/wiki/Q10861465	Brother
	https://www.wikidata.org/wiki/Q595094	Sister
	https://www.wikidata.org/wiki/Q7566	Parent
	https://www.wikidata.org/wiki/Q7560	Mother
	https://www.wikidata.org/wiki/Q7565	Father
	https://www.wikidata.org/wiki/Q7569	Child
	https://www.wikidata.org/wiki/Q177232	Son
	https://www.wikidata.org/wiki/Q308194	Daughter
Role	https://www.wikidata.org/prop/P1038	relative
	https://www.wikidata.org/prop/P26	spouse
	https://www.wikidata.org/prop/P8810	hasParent
	https://www.wikidata.org/prop/P22	hasFather
	https://www.wikidata.org/prop/P25	hasMother
	https://www.wikidata.org/prop/P40	hasChild
	https://www.wikidata.org/prop/P3373	hasSibling

Table 4.4: The most used properties in Family Dataset

IRI of properties	Frequency count
https://www.wikidata.org/prop/P3373	75654
https://www.wikidata.org/prop/P40	64983
https://www.wikidata.org/prop/P22	49973
https://www.wikidata.org/prop/P26	46561
https://www.wikidata.org/prop/P25	16408
https://www.wikidata.org/prop/P1038	4656

Limitations

The Family Dataset includes only six properties related to family relations, encompassing 116,000 distinct entities. Approximately 90,000 of these entities appear in fewer than five triples. In contrast, the entity "Chulalongkorn, King of Siam", is featured in over 150 triples due to his 76 children and 68 wives.

Advantages and Selection Rationale

Each property in this dataset has been trained with thousands of entities. The dataset maintains a high correlation between the local and online WikiData versions. The TBox presented in table 4.3, a subset of the WikiData TBox, includes some family relations. In contrast to DBPedia15k, this dataset is more 'clique-like', meaning that the random queries generated have a much higher chance of validity.

We selected this dataset because of the convenience of handling family relations and the strong assumption that this dataset could demonstrate the benefit of query rewriting.

4.2 Research questions

Until now, we have delineated the details of our framework and the datasets we utilized, and now we present extensive testing results on the framework. We have examined whether query rewriting enhances the process of answering complex queries and improves results compared with a standard KG lookup. The conclusions drawn from these investigations will be leveraged in this section to respond to the research questions raised in the introduction chapter. We will present and succinctly interpret the results to answer these research questions thoroughly.

4.2.1 Question 1: How can we integrate query rewriting with KGEs to enhance complex query answering?

We provide a detailed answer to this research question by considering chapter 3 as a whole, which thoroughly examines the methodologies for effectively combining query rewriting and KGEs to enhance complex query answering. We depicted a schematic representation of this approach in fig. 3.1. A summarization of the framework follows.

Our framework utilizes the *DL-Lite* algorithm, PerfectRef, that employs a TBox to elaborate our initial query into a union of CQs, designated as PR. Subsequently, we compile four lists of entities for validation purposes: the entities from the initial query (L/I), those from all rewritten queries (L/R), the entities from the initial query fetched from the online API (O/I), and finally, the entities from the rewritten queries fetched from the online API (O/R).

We process each rewritten query from PR and the initial query through our complex query answering pipeline. This process produces link predictions as potential answers to the queries, and performs intersections and projections according to the query structures. Our KGE models complete these predictions, trained on the Family dataset and DBPedia15k for this research - both subsets of larger KGs. To validate the accuracy of the predicted entity answers and calculate metrics, we validate our prediction results with the previously mentioned four entity lists. An improved score with the rewritten set suggests that the rewriting has enhanced the results. If the online list of entities yields a better score than the local list, our model has correctly predicted novel facts. We make this inference because the online versions contain the complete KG of WikiData and DBPedia. The initial query (L/I) validation forms the baseline score. We use the

other three lists of entities to validate the predictions, and we can identify the impact of rewriting and predictions by comparing its score to the baseline by tracking any improvements or deteriorations in the scores. By having these four validation outcomes, we can answer whether query rewriting with KGE predictions improves the results.

In this study, we have chosen a simple yet pragmatic approach to construct a complex query pipeline. We aim to demonstrate the potential of merging query rewriting with KGEs for complex query answering. To the best of our knowledge, no other research has investigated this unique combination for complex query answering at the time of this study's publication.

4.2.2 Experiment introduction

This section outlines the experimental setup designed to address the remaining research questions. The crux of this research lies in evaluating the efficacy of our novel approach, blending query rewriting and knowledge graph embeddings (KGEs), compared to a standard KG lookup. Our complex query answering framework and PerfectRef implementation are available on GitHub[7].

We begin by investigating the influence of various KGE models on the results, pinpointing the most efficient model for each dataset. Following this, we tackle the primary research question of this thesis: does our approach improve upon a standard KG lookup? Subsequently, we deliberate on how to interpret the results objectively. A hypothesis complements each research question, followed by a presentation of results and a discussion.

Using our tailored query generator, we spawned 25 distinct EPFO queries for each query structure, illustrated in fig. 3.4. We employed PerfectRef to rewrite each query, capping the number of rewritten variants at 500 per query. This limit was necessitated by the potential of some lengthier and broader queries in DBPedia to be restructured into more than 10,000 distinct variations.

We processed each of the 25 queries through our pipeline and calculated Hits@3 and o-MRR metrics for each instance using our four entity lists. Their application in similar research guided our choice of these metrics [49, 48, 23, 4, 6]. Specifically, we selected Hits@3 due to its immunity to fluctuations in ontology size, providing a stable

[7] `https://github.com/AImenes/`

performance measure. On the other hand, including the o-MRR metric offers insights into performance scenarios where hits fall outside the top three positions, even though it may vary with the size of the ontology. We then computed the average for these metrics across all 25 queries for each query structure. The evaluation utilized the top configurations of all five models across both datasets, yielding ten results. We present a comprehensive overview of these outcomes in appendix D. Detailed information regarding specific entity outcomes for each query, along with their respective scores, can be accessed on our GitHub repository[8].

4.2.3 Question 2: How are the results affected by different KGEs?

Following the training phase, we selected our five (BoxE, CompGCN, DistMult, RotatE, and TransE) model configurations using the AMRI metric. This evaluation aimed to discern the optimal configuration for each model across both datasets (refer to appendix C for further details on training results). BoxE is trained with NSSALoss as the objective function, while the four other models used the MRL objective function. We trained nine distinct configurations for each model (comprising three varying epoch lengths and three different embedding dimensions), culminating in 45 models for each dataset. We then assessed these models on our validation set (not to be confused with the four lists we use for query answering validation). Remember that we leverage the online versions of the KGs to validate the accuracy of the final predictions, which only necessitated partitioning these datasets into a training set and a validation set to select the optimal configuration from the nine available options for each model. The configuration that consistently excelled on the validation set for each model was one with 192 dimensions and 24 epochs, with one exception: DistMult on the family dataset performed optimally with 192 dimensions and 16 epochs.

With five models, we get ten pipeline iterations for this experiment (consisting of five model configurations and two datasets), where we define one iteration as the evaluation metrics averaged over 25 initial queries and their corresponding sets of rewritten queries across all nine query structures.

In this research question, we strive to identify the model that best facilitates query answering and examine the impact of diverse models on the results and the query

[8] `https://github.com/AImenes/query-answering-and-embeddings`

structures. Our next research question will then conclusively determine whether our approach successfully bolsters the efficacy compared to conventional KG lookups.

Concerning the influence of different KGE models, our hypothesis stipulates that no single model will consistently outperform the others across all query structures. We expect that unique designs of different models will interpret and react to patterns within query structures differently. In particular, the TransE model, due to its simplicity and its limitation in capturing symmetry [46], may not fare as well, especially when dealing with the family dataset, which heavily features relational structures such as hasSpouse and hasSibling. In addition to this, recall that DistMult cannot capture asymmetric relations due to design, and might also suffer on the family dataset.

An extensive analysis of the results is accessible in appendix D. However, in this section, we condense these results into three concise tables for each dataset. The first table showcases the highest scores for rewritten values in general, providing insights into which model offers the best general prediction for the dataset. The final two tables will underscore the influence of rewriting and KGE link predictions, achieved by calculating the differences in scores when a standard KG lookup serves as validation. Our intention is twofold: to identify the best-performing model overall and to highlight those models that gain the most from rewriting and prediction. We will utilize the best-performing model overall in the next research question.

Experiment results

Family Dataset In the following tables, we commence with table 4.5, which underscores the top-performing models for every query structure, independent of the extent of improvement over initial queries. Subsequently, table 4.6 elucidates the enhancements yielded by query rewriting across all models. Ultimately, table 4.7 presents the efficacy of KGE predictions in novel facts that were unknown to the models.

DBPedia15k Dataset In the following tables, we commence with table 4.8, which underscores the top-performing models for every query structure, independent of the extent of improvement over initial queries. Subsequently, table 4.9 elucidates the enhancements yielded by query rewriting across all models. Ultimately, table 4.10 presents the efficacy of KGE predictions in unearthing novel facts that were hitherto unknown to the models.

Table 4.5: Best o-MRR and Hits@3 values on local rewriting validation (L/R) result comparison of different models for each query structure on the family dataset.

Structure	BoxE	CompGCN	DistMult	RotatE	TransE
o-MRR Local KG Validation					
1p	**0.6800**	**0.6800**	**0.6800**	**0.6800**	0.4757
2p	0.9617	0.3856	0.8100	**0.9700**	0.5590
3p	0.9333	0.2168	0.4971	**1**	0.4457
2i	0.3600	0.5600	0.3680	0.3200	**0.5924**
3i	0.0400	0.1600	0.2800	0.0400	**0.4733**
pi	0.4229	0.3119	0.3636	0.5200	**0.7901**
ip	0.2800	0.0508	0.3057	0.2400	**0.3792**
up	**0.9000**	0.3841	0.4902	0.8900	0.6181
2u	**0.8000**	**0.8000**	0.7800	**0.8000**	0.6391
Hits@3 Local KG Validation					
1p	**0.6800**	**0.6800**	**0.6800**	**0.6800**	0.4267
2p	0.7733	0.3067	0.7200	**0.9600**	0.3733
3p	0.8133	0.1599	0.3867	**0.9733**	0.3467
2i	0.3467	0.4267	0.3600	0.3200	**0.4533**
3i	0.0400	0.1067	0.1333	0.0400	**0.2533**
pi	0.3600	0.2133	0.2667	**0.5200**	0.4533
ip	**0.2266**	0.0400	0.2000	0.2133	**0.2267**
up	0.8000	0.2533	0.3733	**0.8267**	0.3733
2u	**0.8000**	**0.8000**	0.7867	**0.8000**	0.6133
o-MRR Online KG Validation					
1p	**0.6800**	**0.6800**	**0.6800**	**0.6800**	0.4106
2p	**1**	0.3919	0.7949	0.9500	0.5307
3p	0.8656	0.1672	0.6056	**0.9700**	0.5623
2i	0.3600	**0.5600**	0.4080	0.3200	0.5567
3i	0.0400	0.2000	0.3000	0.0400	**0.4533**
pi	0.3433	0.2319	0.4080	0.4600	**0.6778**
ip	0.3600	0.1209	0.3557	0.3000	**0.4853**
up	**0.9000**	0.3881	0.4769	0.8500	0.5540
2u	**0.8000**	0.7400	0.7800	**0.8000**	0.6223
Hits@3 Online KG Validation					
1p	0.6800	0.6800	0.6800	0.6800	0.3733
2p	0.8000	0.2667	0.7067	**0.9333**	0.3467
3p	0.7867	0.0800	0.4667	**0.9200**	0.4133
2i	0.3467	**0.4400**	0.3867	0.3200	0.4267
3i	0.0400	0.1067	0.1333	0.0400	**0.2400**
pi	0.2933	0.1867	0.3067	0.4267	**0.3867**
ip	0.2933	0.0667	0.2133	0.2667	**0.3467**
up	0.7466	0.2133	0.3867	**0.8000**	0.4267
2u	**0.8000**	0.7467	0.7867	**0.8000**	0.6000

Table 4.6: Local Delta values (Δ) (Rewritten scores vs initial scores) with comparison of different models for each query structure on the family dataset.

Structure	BoxE	CompGCN	DistMult	RotatE	TransE
o-MRR Δ Local KG Validation					
1p	**0.2400**	**0.2400**	**0.2400**	**0.2400**	0.1450
2p	0.0985	0.0568	0.1019	0.0550	**0.2164**
3p	0.1548	**0.1998**	0.0576	0.1143	0.1282
2i	0.0800	0.1200	0.0800	0.0400	**0.1257**
3i	0.0400	0.0800	0.1200	0.0400	**0.1733**
pi	0.1096	**0.1872**	0.1229	0.1000	0.1788
ip	0	0	0	**0.0200**	0
up	0.0593	0.0301	0.0487	0.0369	**0.1053**
2u	0.2400	**0.3126**	0.2587	0.3097	0.1355
Hits@3 Δ Local KG Validation					
1p	**0.2400**	**0.2400**	**0.2400**	**0.2400**	0.1200
2p	0.1200	0.0667	0.0800	0.0667	**0.1466**
3p	0.1600	0.1599	0.0667	0.1200	**0.1734**
2i	0.0800	0.0934	0.0800	0.0400	**0.1333**
3i	0.0400	0.0534	**0.1266**	0.0400	0.0933
pi	0.1067	0.1066	0.1067	0.1067	**0.1333**
ip	0	0	0	**0.0133**	0
up	**0.1067**	0.0400	0.0266	0.0400	**0.1067**
2u	0.2667	**0.3067**	**0.3067**	**0.3067**	0.1600
o-MRR Δ Online KG Validation					
1p	**0.2400**	**0.2400**	**0.2400**	**0.2400**	0.0923
2p	**0.0397**	0.0031	0.0329	0.0200	0.0390
3p	**0.0397**	0.0040	0.0126	0.0320	0.0114
2i	0.0400	**0.0800**	**0.0800**	0	0.0500
3i	0.0400	**0.1200**	0.0600	0.0400	0.0933
pi	-0.0024	0.0192	0.0333	**0.0400**	0.0182
ip	**0.0400**	0.0214	0.0057	**0.0400**	0.0142
up	**0.2793**	0.0801	0.1279	0.2320	0.1587
2u	**0.2000**	0.1400	0.1600	**0.2000**	0.0423
Hits@3 Δ Online KG Validation					
1p	**0.2400**	**0.2400**	**0.2400**	**0.2400**	0.0800
2p	**0.0400**	0	**0.0400**	0.0266	**0.0400**
3p	0.0134	0	0.0134	**0.0400**	0.0133
2i	0.0400	**0.0667**	**0.0667**	0	0.0400
3i	0.0400	**0.0534**	0.0400	0.0400	**0.0533**
pi	0	0.0134	**0.0400**	0.0134	0
ip	0.0267	0.0134	0	**0.0400**	-0.0133
up	0.2267	0.0533	0.1200	**0.2533**	0.1467
2u	0.2000	**0.2534**	0.2000	0.1867	0.0400

Table 4.7: Delta values (Δ) for predictions (Online KG vs local KG) with comparison of different models for each query structure on the family dataset.

Structure	BoxE	CompGCN	DistMult	RotatE	TransE
Hits@3 Δ Original query					
1p	0	0	0	0	-0.0134
2p	**0.1067**	0.0267	0.0267	0.0134	0.0800
3p	0.1200	0.0800	0.1333	0.0267	**0.2267**
2i	0.0400	0.0400	0.0400	0.0400	**0.0667**
3i	0	0	**0.0866**	0	0.0267
pi	0.0400	0.0666	**0.1067**	0	0.0667
ip	0.0400	0.0133	0.0133	0.0267	**0.1333**
up	-0.1734	-0.0533	-0.0800	-0.2400	**0.0133**
2u	0.0667	0	0.1067	**0.1200**	0.1067
o-MRR Δ Original query					
1p	0	0	0	0	-0.0124
2p	0.0971	0.0600	0.0548	0.0150	**0.1491**
3p	0.0476	0.1462	0.1535	0.0523	**0.2334**
2i	**0.0400**	**0.0400**	**0.0400**	**0.0400**	**0.0400**
3i	0	0	**0.0800**	0	0.0600
pi	0.0324	0.0880	**0.1340**	0	0.0483
ip	0.0400	0.0487	0.0433	0.0400	**0.0919**
up	-0.2200	-0.0460	-0.0925	-0.2351	-0.1175
2u	0.0400	**0.1126**	0.0987	0.1097	0.0764
Hits@3 Δ Rewritten queries					
1p	0	0	0	0	-0.0534
2p	**0.0267**	-0.0400	-0.0133	-0.0267	-0.0266
3p	-0.0266	-0.0799	**0.0800**	-0.0533	0.0666
2i	0	0.0133	**0.0267**	0	-0.0266
3i	0	0	0	0	-0.0133
pi	-0.0667	-0.0266	**0.0400**	-0.0933	-0.0666
ip	0.0667	0.0134	0.0133	0.0534	**0.1200**
up	-0.0534	-0.0400	0.0134	-0.0267	**0.0534**
2u	0	-0.0533	0	0	-0.0133
o-MRR Δ Rewritten queries					
1p	0	0	0	0	-0.0651
2p	**0.0384**	0.0063	-0.0151	-0.0200	-0.0283
3p	-0.0675	-0.0496	0.1085	-0.0300	**0.1166**
2i	0	0	**0.0400**	0	-0.0357
3i	0	**0.0400**	0.0200	0	-0.0200
pi	-0.0796	-0.0800	0.0444	-0.0600	-0.1123
ip	0.0800	0.0701	0.0500	0.0600	**0.1061**
up	0	**0.0040**	-0.0133	-0.0400	-0.0641
2u	0	-0.0600	0	0	-0.0168

Table 4.8: Best o-MRR and Hits@3 values on Local KG with rewritings (L/R) comparison of different models for each query structure on the DBPedia15k dataset.

Structure	BoxE	CompGCN	DistMult	RotatE	TransE
o-MRR Local KG Validation					
1p	0.5342	0.6355	0.6457	0.5697	**0.6854**
2p	0.0536	0.0153	**0.0555**	0.0066	0.0295
3p	0.0040	0	0.0027	0	**0.0041**
2i	0.1120	0.2124	0.1536	0.1096	**0.2283**
3i	0.0400	0.1733	0.0800	0.0527	**0.1800**
pi	0	**0.0033**	0.0027	0	0
ip	**0.0181**	0.0032	0.0041	0.0055	0.0162
up	0.1198	0.0835	**0.1469**	0.1283	0.1357
2u	0.5636	0.6079	0.5988	0.5586	**0.7630**
Hits@3 Local KG Validation					
1p	0.4267	0.4667	0.5867	0.3867	**0.6000**
2p	0.0133	0	**0.0267**	0	0
3p	0	0	0	0	0
2i	0.0800	0.1200	0.1200	0.0933	**0.1733**
3i	0.0267	0.0933	0.0400	0.0267	**0.1600**
pi	0	0	0	0	0
ip	**0.0133**	0	0	0	0
up	0.0400	0.0133	0.0800	0.0400	**0.1067**
2u	0.3867	0.4533	0.5333	0.3733	**0.6933**
o-MRR Online KG Validation					
1p	0.2293	0.2124	0.1855	**0.3382**	0.2306
2p	**0.1049**	0.0186	0.0614	0.0116	0.0530
3p	0.0014	0.0009	0.0017	0	**0.0408**
2i	0.0024	**0.0644**	0.0021	0.0235	0.0571
3i	0.0008	**0.0200**	0	0	0
pi	0	**0.0025**	0.0013	0	0
ip	**0.0133**	0.0013	0.0010	0.0037	0.0057
up	0.0531	0.0366	0.0657	**0.0864**	0.0390
2u	0.2214	0.2591	0.2462	0.2889	**0.4075**
Hits@3 Online KG Validation					
1p	0.1200	0.1200	0.0533	0.1200	**0.2000**
2p	**0.0400**	0	0.0267	0	0.0267
3p	0	0	0	0	**0.0133**
2i	0	**0.0400**	0	0.0133	0.0267
3i	0	**0.0133**	0	0	0
pi	0	0	0	0	0
ip	**0.0133**	0	0	0	0
up	0.0133	0.0133	**0.0400**	0.0267	0.0267
2u	0.1067	0.1333	0.1200	0.1067	**0.3067**

Table 4.9: Local Delta values (Δ) (Rewritten scores vs initial scores) with comparison of different models for each query structure on the DBPedia15k dataset.

Structure	BoxE	CompGCN	DistMult	RotatE	TransE
o-MRR Δ Local KG Validation					
1p	0.1433	0.1585	0.1430	0.0953	**0.2163**
2p	0	0	0	0	0
3p	0	0	0	0	0
2i	0.0896	0.0760	0.0208	0.0060	**0.1200**
3i	0.0200	0.1333	0.0400	0.0327	**0.1600**
pi	0	0	0	0	0
ip	0	0	0	**0.055**	0.0162
up	0	0	0.	0	0
2u	0.0800	0.0434	**0.2666**	0.0200	0.0933
Hits@3 Δ Local KG Validation					
1p	0.1200	0.1067	0.1334	0.0667	**0.3467**
2p	0	0	0	0	0
3p	0	0	0	0	0
2i	0	0.0400	0	0	**0.0933**
3i	0	0.0533	0	0	**0.1340**
pi	0	0	0	0	0
ip	0	0	0	0	0
up	0	0	0	0	0
2u	0.0667	0.0666	0.0666	0.0133	**0.1066**
o-MRR Δ Online KG Validation					
1p	-0.0137	-0.1336	-0.2172	**0.0643**	-0.2361
2p	0	0	0	0	0
3p	0	0	0	0	**0.0367**
2i	-0.0076	0.0350	-0.0049	-0.0143	**0.0135**
3i	0.0008	**0.0200**	0	0	0
pi	0	-0.0008	-0.0014	0	0
ip	0	-0.0019	-0.0031	0	-0.005
up	**0.0157**	0.0022	-0.0013	0.0010	0.0138
2u	-0.0237	-0.1469	-0.1398	-0.0210	-0.1056
Hits@3 Δ Online KG Validation					
1p	-0.0133	-0.0933	-0.1734	**0.0133**	-0.1600
2p	0	0	0	0	**0.0267**
3p	0	0	0	0	**0.133**
2i	0	**0.0133**	0	-0.0134	**0.0133**
3i	0	**0.0133**	0	0	0
pi	0	0	0	0	0
ip	0	0	0	0	0
up	0	-0.0134	0	0	0
2u	**0.0134**	-0.12	-0.1067	-0.0666	-0.1200

Table 4.10: Delta values (Δ) for predictions (Online KG vs local KG) with comparison of different models for each query structure on the DBPedia15k dataset.

Structure	BoxE	CompGCN	DistMult	RotatE	TransE
Hits@3 Δ Original query					
1p	-0.1734	-0.1467	-0.2266	-0.2133	**0.1067**
2p	**0.0267**	0	0	0	**0.0267**
3p	0	0	0	0	**0.0133**
2i	-0.0800	-0.0533	-0.1200	-0.0666	-0.0667
3i	-0.0267	-0.0400	-0.0400	-0.0267	-0.0260
pi	0	0	0	0	0
ip	0	0	0	0	0
up	-0.0267	0.0134	-0.0400	-0.0133	-0.0800
2u	-0.2267	-0.1334	-0.2400	-0.1867	-0.1600
o-MRR Δ Original query					
1p	-0.1479	-0.1300	-0.1000	-0.2005	-0.0024
2p	0.0513	0.0033	**0.0059**	0.0050	0.00267
3p	-0.0026	0.0009	-0.0010	0	**0.0367**
2i	-0.0924	-0.1070	-0.1258	-0.0658	-0.0647
3i	-0.0200	-0.0400	-0.0400	-0.0200	-0.0200
pi	0	-0.0008	-0.0014	0	0
ip	-0.0048	-0.0019	-0.0031	-0.0018	-0.0105
up	-0.0824	-0.0491	-0.0799	-0.0429	-0.1105
2u	-0.2385	-0.1585	-0.1862	-0.2287	-0.1566
Hits@3 Δ Rewritten queries					
1p	-0.3067	-0.3467	-0.5334	-0.2667	-0.4000
2p	**0.0267**	0	0	0	**0.0267**
3p	0	0	0	0	0.0133
2i	-0.0800	-0.0800	-0.1200	-0.0800	-0.1466
3i	-0.0267	-0.0800	-0.0400	-0.0267	-0.1600
pi	0	0	0	0	0
ip	0	0	0	0	0
up	-0.0267	0	-0.0400	-0.0133	-0.0800
2u	-0.2800	-0.3200	-0.4133	-0.2800	-0.3866
o-MRR Δ Rewritten queries					
1p	-0.3049	-0.4231	-0.4602	-0.2315	-0.4548
2p	0.0513	0.0033	**0.0059**	0.0050	0.0235
3p	-0.0026	0.0033	-0.0010	0	**0.0367**
2i	-0.1096	0.0009	-0.1515	-0.0861	-0.1712
3i	-0.0392	-0.1533	-0.0800	-0.0527	-0.1800
pi	0	-0.0008	-0.0014	0	0
ip	-0.0048	-0.0019	-0.0031	-0.0018	-0.0105
up	-0.0667	**0.0469**	-0.0812	-0.0419	-0.0967
2u	-0.3422	-0.3488	-0.3526	-0.2697	-0.3555

Experiment discussion

Best model for query answering We begin by examining table 4.5 for the Family dataset. This table, in its upper two sections, presents the o-MRR and Hits@3 results based on our prediction using a list of entities retrieved from a KG lookup across all rewritten queries (L/R). In doing so, we can discern how well various models handle general link prediction independently of the rewriting impact. To provide a comprehensive picture, we have included the same metrics in the lower two sections of the table, but here we use the list of entities retrieved from the online versions of the datasets.

To offer the most honest evaluation of the model, we primarily consider the validation performed using the same dataset employed for training (local KG). A notable observation is that for the '1p' structure, all models except TransE achieve an o-MRR score of 0.68, indicating that these models handle this query structure effectively. In KGE terminology, a '1p' query solely pertains to link prediction, and the 'p' for projection does not truly reflect projection in this scenario. It applies equally (or minimally) to the intersection and disjunction categories.

Considering pure projection query structures, '2p' and '3p', RotatE emerges as the top performer, achieving an impressive o-MRR score of 1 for '3p'—meaning, with the rewriting validation, the top-ranked entity is always present in that list, happening for all 25 initial queries. BoxE also exhibits commendable scores for projection. Despite their differing designs—RotatE performs prediction by considering rotations in the complex plane, while BoxE is based on a translational approach augmented with Box mechanics—both models demonstrate that one of the three categories introduced in the background does not necessarily outperform the others for projection.

Intriguingly, TransE outperforms the other four models for intersection queries—by a significant margin. With intersections, the entities outputted from each atom in the query must intersect for validation. Therefore, TransE's simplicity might cause it to predict more similar entities than the other models, as it employs simpler vector embeddings. Given that the cut-off value for this test case is set at 100 entities for each prediction, TransE may have more valid outputs from its prediction. At the same time, the other models might exhibit less intersection between the atoms due to their output entity sets being more disjunct.

TransE also stands out in the combination of projection and intersection queries. Despite its underperformance compared to the other models for projection, the combination

of the two query types still seems to favour TransE—the simplest translational-based model.

BoxE comes out on top for disjunction, followed closely by RotatE, with nearly identical results. In the 'up' structure, projection explains why BoxE and RotatE perform well. As for '2u', it essentially involves two '1p' predictions, with the union of the results obtained subsequently. This explains why the '2u' results closely mirror '1p' results regarding relative differences and scaling—with TransE performing the worst.

DistMult does not perform best on any structure (except for a shared first on '1p'). This could be because DistMult struggles with asymmetric relationships due to its design.

Turning our attention to the DBPedia15k dataset, we focus on local validation for this discussion, given its direct correlation with the training data. As shown in table 4.8, the scores present a different narrative—several markedly low, indicating a probable abundance of mispredicted entities.

However, the '1p' structure—commonly called the standard link prediction in KGE terminology—registers respectable scores. TransE achieves the highest score with an o-MRR value of 0.6854.

Contrary to the Family dataset, projection scores are substantially lower for DBPedia15k. The best score for '2p' is recorded by DistMult, with a score of 0.0555, while TransE tops '3p' with a modest 0.0041. All models struggle in this context.

When it comes to intersections, the results fare better. TransE is also better than the other models for this dataset, registering a score of 0.2283 for '2i' and 0.18 for '3i'.

Up until this point, TransE consistently emerges as the model of choice. However, on closer scrutiny, we note that while CompGCN never surpasses TransE, it does trail very closely. Upon encountering the 'pi' structure, CompGCN posts a result of 0.0033, outpacing TransE's zero scores. For 'ip', BoxE delivers the least dismal score of 0.0181, followed closely by TransE. Discerning a pattern for the best-performing model in terms of combined query structures ('ip' and 'pi') for DBPedia15k is challenging due to low scores.

Finally, regarding disjunction, DistMult leads the pack for 'up' structure, with TransE behind. For '2u', much like the Family dataset, the results align closely with the '1p' structure—TransE outperforms the others with a score of 0.7630.

In assessing these results, it is vital to bear in mind the inherent structures of the datasets. Remember that our query generator cannot predict whether a generated query has any possible answers. About the Family dataset, generating degenerate queries is inherently challenging, given the high likelihood of a previously predicted individual having another family connection (in other words, an additional atom in the query). However, this is more complex with DBPedia15k, as this subset encompasses numerous logical branches, such as 'places' and 'people,' and few plausible intersections and projections exist between these branches.

Despite these challenges, the results for the '1p' and '2u' structures remain commendably strong across both datasets.

Improvements using query rewriting In the preceding section, our discussion centred primarily on determining the optimal KGE model for our DBPedia15k and Family datasets. Now, our focus will shift towards understanding which KGE models accrue the most significant benefits from query rewriting in their prediction validations - in other words, which models experience the most substantial improvements with the implementation of query rewriting. We term these scores as *delta scores* (Δ), representing the difference in score between the rewritten and initial query validation cases.

For the Family dataset, our attention turns to table 4.6. At first glance, we notice only positive values for the validation using our local dataset. A slightly negative value is observed for the 'pi' structure when employing the online KG for validation. Given that we include the initial query lookup in the set of rewritten queries, negative values in the context of rewritings are anomalous. One reason may be that the API used to retrieve these entities occasionally returned an HTTP 206 *Partial Content* code, signifying that the complete list of answers exceeded the permitted payload size for the HTTP message.

A notable increase of 0.24 in the o-MRR score is evident for the local KG validation in the '1p' query structure. In line with the previous section, all models except TransE display these delta scores. For '3p', CompGCN benefits the most from rewritings, showing a significant positive increase of 0.1998. TransE and CompGCN demonstrate the most significant gains from query rewriting regarding o-MRR .

For DBPedia15k, as depicted in table 4.9, the results using our datasets for validation dramatically improve with query rewriting. Once again, TransE reaps the highest rewards from rewritings regarding the o-MRR score. The only notable exception is '2u', where DistMult sees an increase of 0.2666.

However, when evaluating the improvements achieved by rewriting in the context of the online KG, the term 'improvement' rapidly loses its accuracy. Here, we mainly witness declines. This unexpected observation likely stems from receiving fewer entities from the APIs on the online KGs due to oversized requests.

Nevertheless, we observe a clear and significant improvement with rewritings compared to initial KG lookup validation on our predictions for all models, but most significantly for TransE and CompGCN on our local validation. This observation suggests that query rewriting augments our results by expanding the list of entities used for validation. If both local and online KGs were complete, then rewriting to multiple rewritten queries would not introduce any new entities for validation. Hence, we have also identified that these datasets are not complete.

Improvements using predictions In this latter part of our dual-pronged approach, we focus on KGE predictions. When assessing the effectiveness of predictions in the context of complex query answering, it is customary to structure the training set as a subset of the validation set, which is, in turn, a subset of the test set. It is essential to remember that, in our case, the models have been trained and validated on our local datasets. We employ more complete online versions of DBPedia and WikiData for validations on correctness for our predictions.

Under the assumption that the online versions of our KGs have a higher rate of completeness than the local versions (which is the case for DBPedia and WikiData), we can use the online KG validation as compared to the local one to ascertain whether an entity predicted by the model to be an answer to a query is, indeed, a correct prediction, provided it is part of the online KGs. However, as noted in the rewriting section, the online KGs are also only partially complete, making it likely that we receive some entities that, according to our interpretation of the domain, would yield a correct result without us detecting it. These will not be identified as we have a CWA approach to our predictions. Nonetheless, we would need an effective verification method beyond manually examining each case, and that would not be a feasible approach.

We could add all the predicted entities not present in the KG lookup to the list of answers to that query, assuming that the online KG is not entirely complete (OWA approach). However, in this thesis, evaluating the actual rate of improvement from introducing embeddings becomes problematic in such cases, as it would include false positives. Therefore, we compare the initial query prediction with validation from the

local KG lookup and the online KG lookup for that respective query in the subsequent validation.

In table 4.7 for the Family dataset, we see no improvements for '1p' in terms of the o-MRR for the initial query. However, for '2p' and '3p' queries, the KGE model identifies entities present in the online KG, not in the local one, thereby achieving a correct prediction. We see improvements for all query structures with the introduction of embeddings, except for 'up'. Here, we observe negative values, indicating differences in the entity sets from local and online KGs, which, in an ideal scenario for our validation, should ideally be subsets of the online KGs.

Turning our attention to DBPedia15k, we notice a shift, with the improvements noted for the Family dataset now replaced by declines as shown in table 4.10. We see values close to or equal to 0 for projection cases due to degenerated queries and missing answers. Therefore, to begin with, the model scores for prediction are shallow. This information leads us to conclude that we cannot ascertain any substantial benefit from KGEs for DBPedia15k.

While we saw enhancements in the embedding results for the Family dataset, this was not the case for DBPedia15k. Contrary to the previous section on rewriting results, here, we are comparing results across two different validation origins. Therefore, negative values are primarily due to dataset differences. Despite these differences, we see an improvement and detection of new correctly labelled entities for the Family dataset, best shown by TransE.

Concluding Remarks and Model Selection We have compared various models and assessed their impact on the results. Contrary to our hypothesis, we found that TransE was the model that most effectively leveraged rewriting and KGE predictions. However, RotatE demonstrated superior results for the Family dataset for all projection-related queries. The neural network model, CompGCN, emerged as a strong contender for most structures in DBPedia15k, even surpassing TransE when considering online validation.

Interestingly, DistMult did not perform superior in any cases for the Family dataset, except for a tie for first place in the '1p' structures. One reason for this could be the lack in design for asymmetric relations, which we find in the family dataset. In the case of DBPedia15k, interpretation of the scores became more challenging due to query degeneration. This fact became apparent as it performed well on the '1p' and

'2u' structures, where the prediction involved just one atom (or two atoms where their respective answers are unified).

We observed significant disparities in scores between the two datasets, primarily attributed to query degeneration. Additionally, we noted negative values for our delta scores. One unverified hypothesis suggests that in the case of rewritings, we may not have received the complete list of answers from the online API. Our pipeline observations substantiate this hypothesis where some API requests returned with HTTP code 206, signifying partial content delivery. Regarding the impact of prediction, negative values indicate discrepancies between our datasets and the online KGs used for validation.

For addressing the following research question, in fairness, we will carry forward RotatE for its best overall performance from local evaluation, even though TransE had the best improvements (i.e. delta scores) for the Family dataset. For DBPedia15k, we will employ TransE, given its superior prediction performance and its achievement of the best improvements (or least degradation) according to our delta tables.

4.2.4 Question 3: How does our integrated query rewriting approach compare to standard KG lookups?

In this research question, we will use the two proposed models from the previous research question to answer more explicitly whether our approach does improve on a standard KG Lookup. We use RotatE for the family dataset and TransE for DBPedia15k. Before running the test case, our hypothesis proposed that incorporating query rewriting should improve the results across all nine query structures. Nevertheless, enhancements might not be as significant in DBPedia15k due to the increased probability of degenerate queries.

Experiment results

We first present the family dataset results, thereafter the DBPedia15k.

Table 4.11: Summary of results for different query structures for the dataset **family** and model **RotatE** (dim: 192, epoch: 24). All query structures have a count of 25 different queries. L/I: Local KG prediction using solely initial query, L/R: Local KG prediction using rewritten queries, O/I: Online KG prediction using solely initial query, O/R: Online KG prediction using rewritten queries.

	Hits@3				o-MRR			
Query Structure	L/I	L/R	O/I	O/R	L/I	L/R	O/I	O/R
1p	0.4400	0.6800	0.4400	0.6800	0.4400	0.6800	0.4400	0.6800
2p	0.8933	0.9600	0.9067	0.9333	0.9150	0.9700	0.9300	0.9500
3p	0.8533	0.9733	0.8800	0.9200	0.8857	1	0.9380	0.9700
2i	0.2800	0.3200	0.3200	0.3200	0.2800	0.3200	0.3200	0.3200
3i	0	0.0400	0	0.0400	0	0.0400	0	0.0400
pi	0.4133	0.5200	0.4133	0.4267	0.4200	0.5200	0.4200	0.4600
ip	0.2000	0.2133	0.2267	0.2667	0.2200	0.2400	0.2600	0.3000
up	0.7867	0.8267	0.5467	0.8000	0.8531	0.8900	0.6180	0.8500
2u	0.4933	0.8000	0.6133	0.8000	0.4903	0.8000	0.6000	0.8000

Structure	Δ Local o-MRR	Δ Online o-MRR	Δ Local Hits@3	Δ Online Hits@3
1p	0.2400	0.2400	0.2400	0.2400
2p	0.0550	0.0200	0.0667	0.0267
3p	0.1143	0.0320	0.1200	0.0400
2i	0.0400	0	0.0400	0
3i	0.0400	0.0400	0.0400	0.0400
pi	0.1000	0.0400	0.1067	0.0133
ip	0.0200	0.0400	0.0133	0.0400
up	0.0369	0.2320	0.0400	0.2533
2u	0.3097	0.2000	0.3067	0.1867

Table 4.12: Family Dataset with RotatE: Delta values (Δ) for rewriting. o-MRR and Hits@3 for each query structure for Local and Online KGs

Structure	Δ Original o-MRR	Δ Rewritten o-MRR	Δ Original Hits@3	Δ Rewritten Hits@3
1p	0	0	0	0
2p	0.0150	-0.02	0.0133	-0.0267
3p	0.0523	-0.03	0.0267	-0.0533
2i	0.04	0	0.04	0
3i	0	0	0	0
pi	0	-0.06	0	-0.0933
ip	0.04	0.06	0.0267	0.0533
up	-0.2351	-0.04	-0.24	-0.0267
2u	0.1097	0	0.12	0

Table 4.13: Family Dataset with RotatE: Delta values (Δ) for KGE predictions. o-MRR and Hits@3 for each query structure for Local and Online KGs

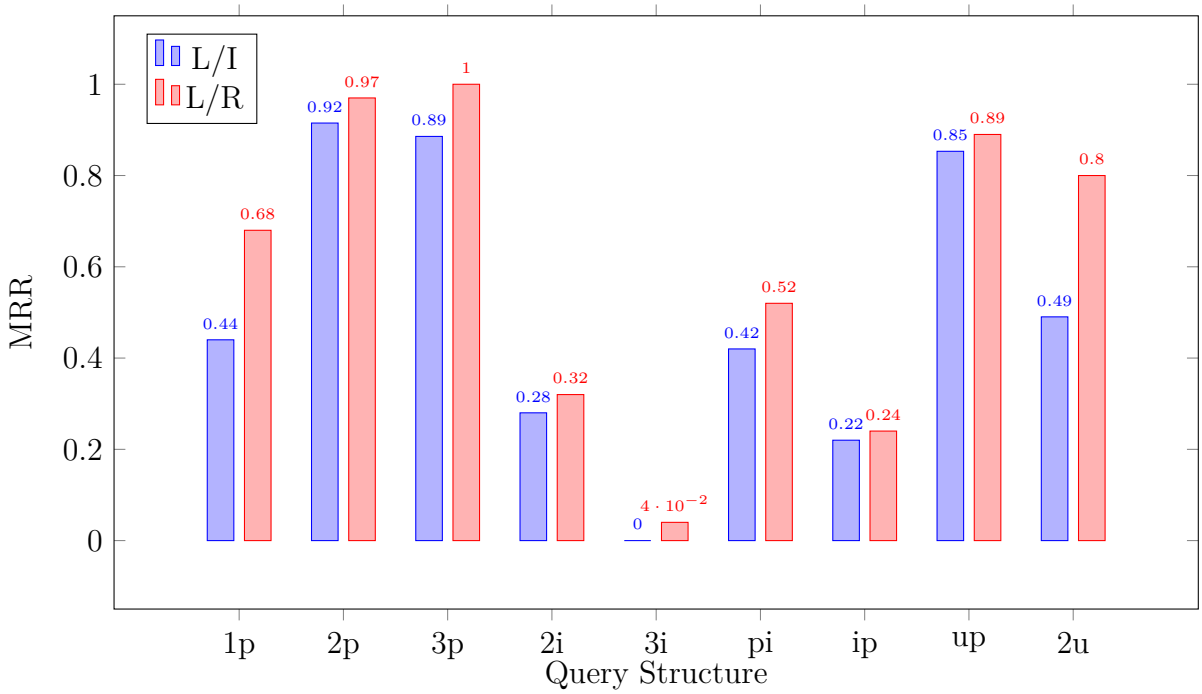


Figure 4.1: Family Dataset with RotatE, showing impact of **rewriting**: o-MRR for L/I and L/R

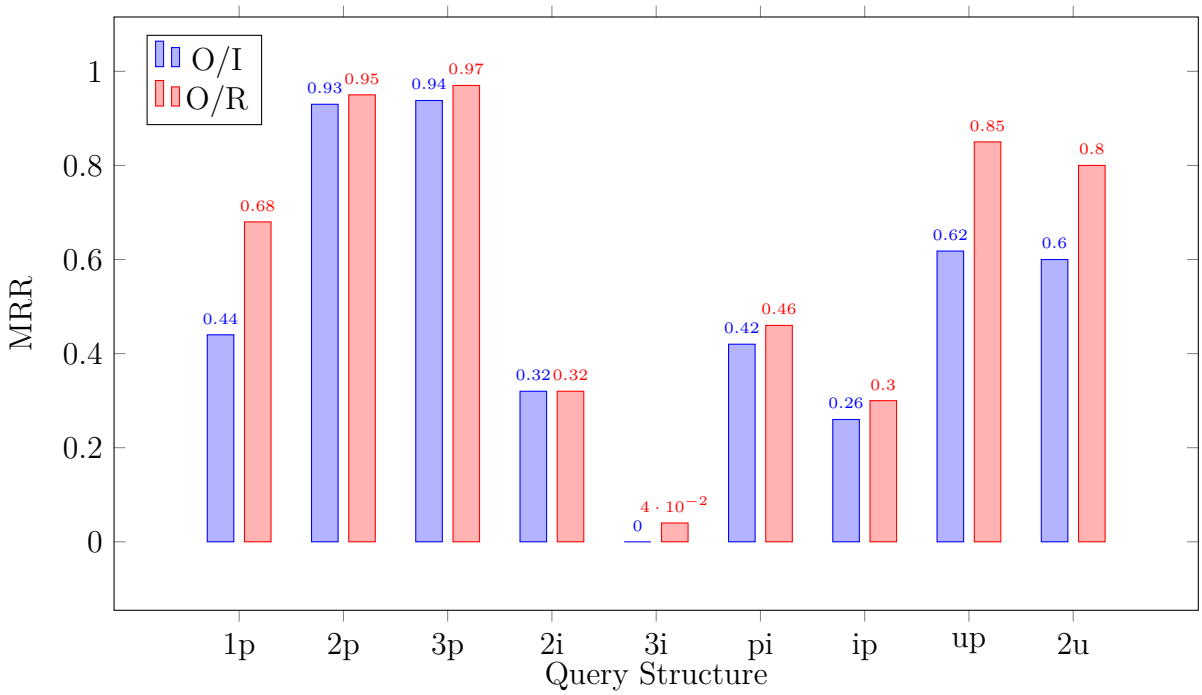


Figure 4.2: Family Dataset with RotatE, showing impact of **rewriting**: o-MRR for O/I and O/R

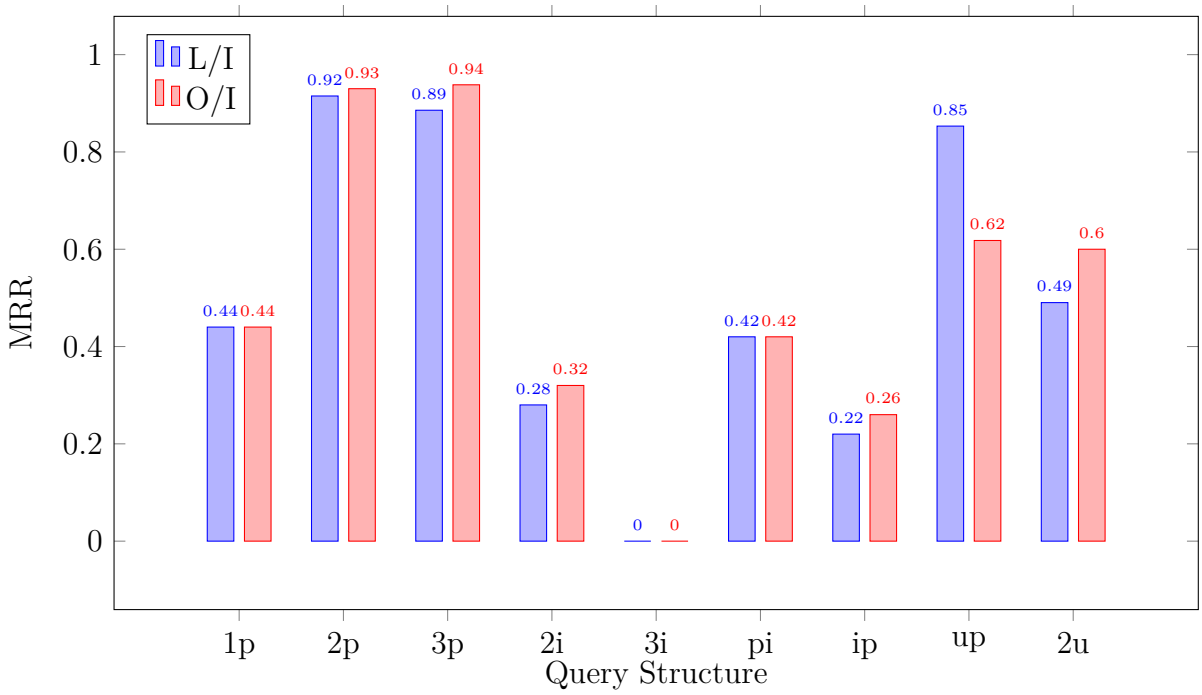


Figure 4.3: Family Dataset with RotatE, showing impact of **predictions**: o-MRR for L/I and O/I

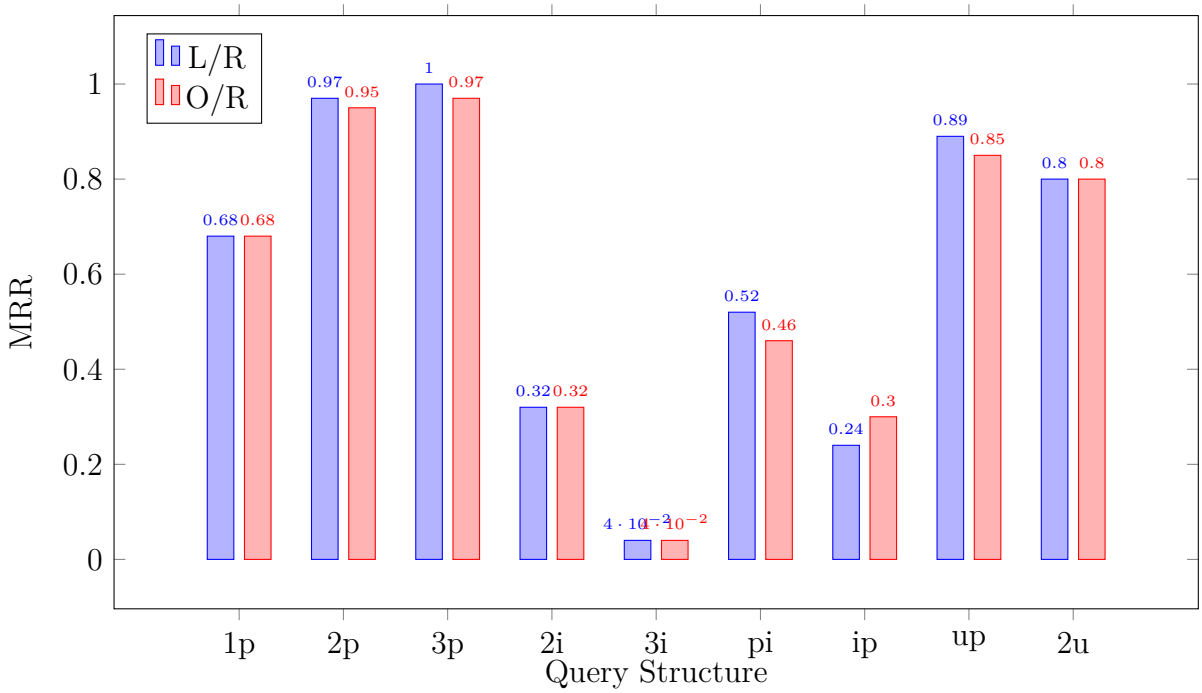


Figure 4.4: Family Dataset with RotatE, showing impact of **predictions**: o-MRR for L/R and O/R

Table 4.14: Summary of results for different query structures for the dataset **DBPedia15k** and model **TransE** (dim: 192, epoch: 24). All query structures have a count of 25 different queries. L/I: Local KG prediction using solely initial query, L/R: Local KG prediction using rewritten queries, O/I: Online KG prediction using solely initial query, O/R: Online KG prediction using rewritten queries.

	Hits@3				o-MRR			
Query Structure	L/I	L/R	O/I	O/R	L/I	L/R	O/I	O/R
1p	0.2533	0.6000	0.3600	0.2000	0.4691	0.6854	0.4667	0.2306
2p	0	0	0.0267	0.0267	0.0295	0.0295	0.0530	0.0530
3p	0	0	0.0133	0.0133	0.0041	0.0041	0.0408	0.0408
2i	0.0800	0.1733	0.0133	0.0267	0.1083	0.2283	0.0436	0.0571
3i	0.0260	0.1600	0	0	0.0200	0.1800	0	0
pi	0	0	0	0	0	0	0	0
ip	0	0	0	0	0.0162	0.0162	0.0057	0.0057
up	0.1067	0.1067	0.0267	0.0267	0.1357	0.1357	0.0252	0.0390
2u	0.5867	0.6933	0.4267	0.3067	0.6697	0.7630	0.5131	0.4075

Structure	Δ Local o-MRR	Δ Online o-MRR	Δ Local Hits@3	Δ Online Hits@3
1p	0.2163	-0.2361	0.3467	-0.1600
2p	0	0	0	0
3p	0	0	0	0
2i	0.1200	0.0135	0.0933	0.0134
3i	0.1600	0	0.1340	0
pi	0	0	0	0
ip	0	0	0	0
up	0	0.0138	0	0
2u	0.0933	-0.1056	0.1066	-0.1200

Table 4.15: Dbpedia Dataset with TransE: Delta values (Δ) for **rewriting**. o-MRR and Hits@3 for each query structure for Local and Online KGs

Structure	Δ Original o-MRR	Δ Rewritten o-MRR	Δ Original Hits@3	Δ Rewritten Hits@3
1p	-0.0024	-0.4548	0.1067	-0.4
2p	0.0235	0.0235	0.0267	0.0267
3p	0.0367	0.0367	0.0133	0.0133
2i	-0.0647	-0.1712	-0.0667	-0.1466
3i	-0.02	-0.18	-0.026	0
pi	0	0	0	0
ip	-0.0105	-0.0105	0	0
up	-0.1105	-0.0967	-0.08	-0.08
2u	-0.1566	-0.3555	-0.16	-0.3866

Table 4.16: DBPedia15k Dataset with TransE: Delta values (Δ) for KGE predictions. o-MRR and Hits@3 for each query structure for Local and Online KGs

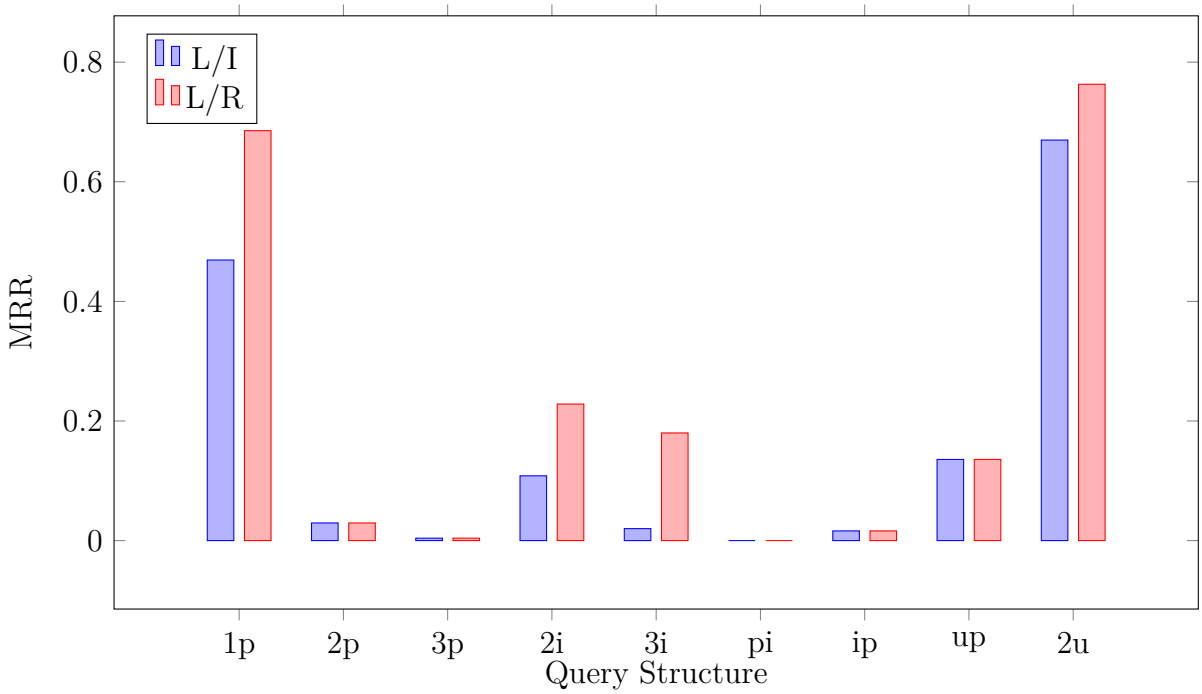


Figure 4.5: DBPedia15k Dataset with TransE, showing impact of **rewriting**: o-MRR for L/I and L/R

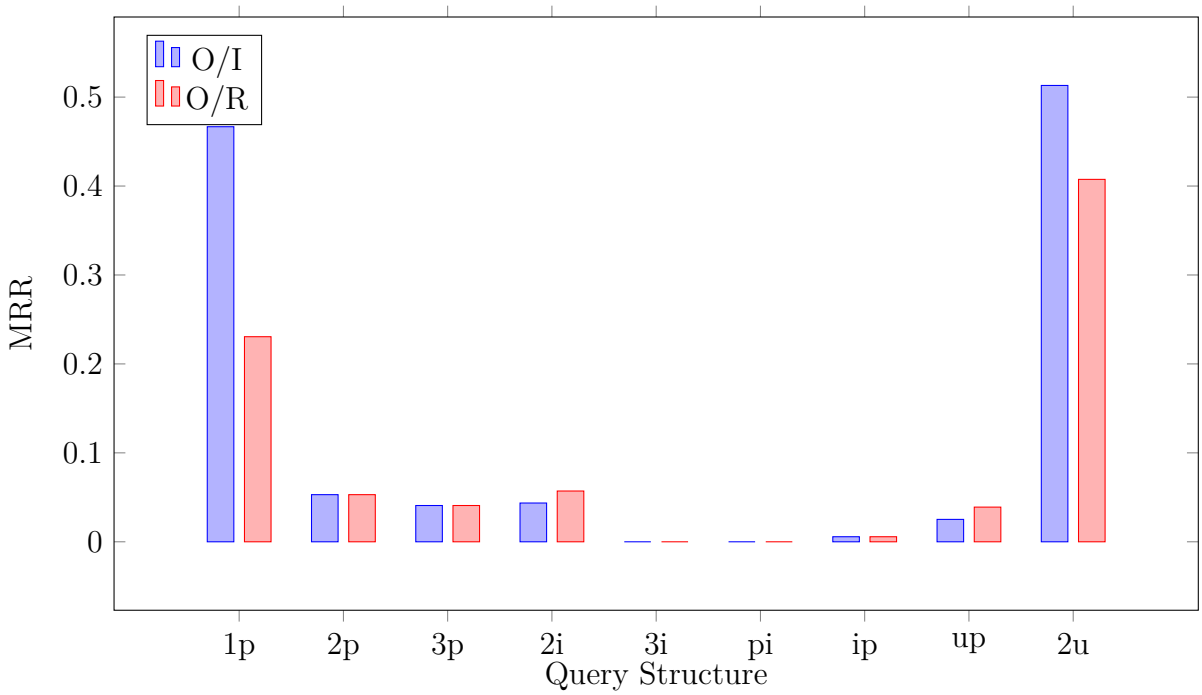


Figure 4.6: DBPedia15k Dataset with TransE, showing impact of **rewriting**: o-MRR for O/I and O/R

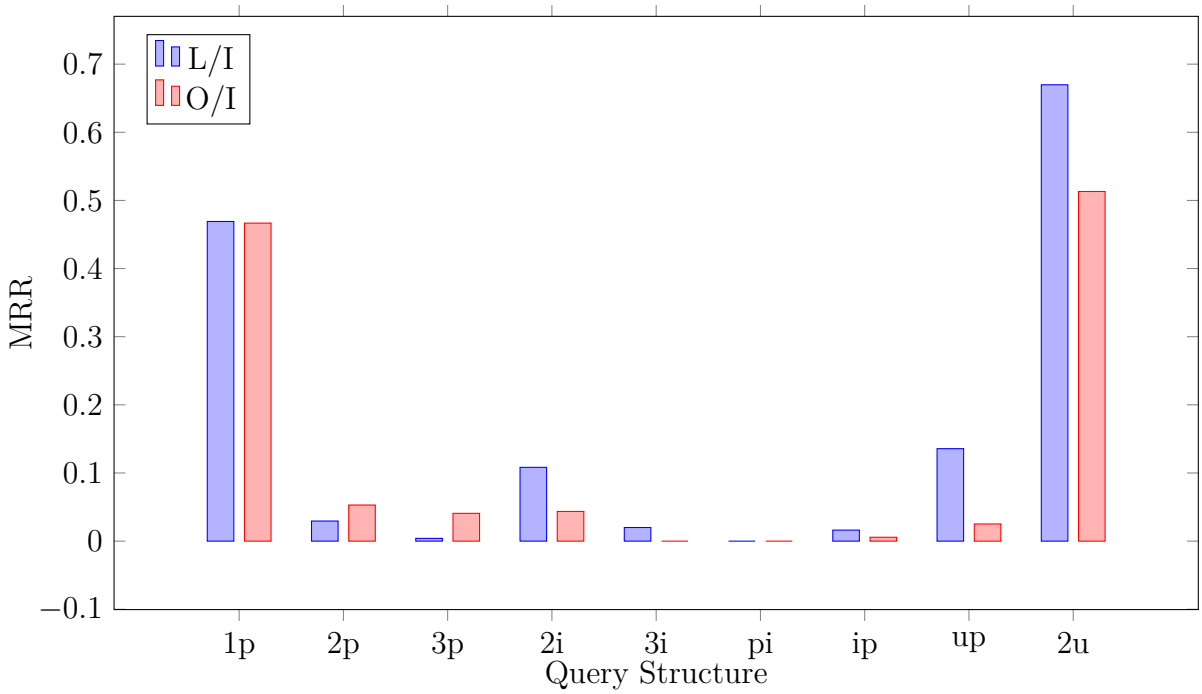


Figure 4.7: DBPedia15k Dataset with TransE, showing impact of **predictions**: o-MRR for L/I and O/I

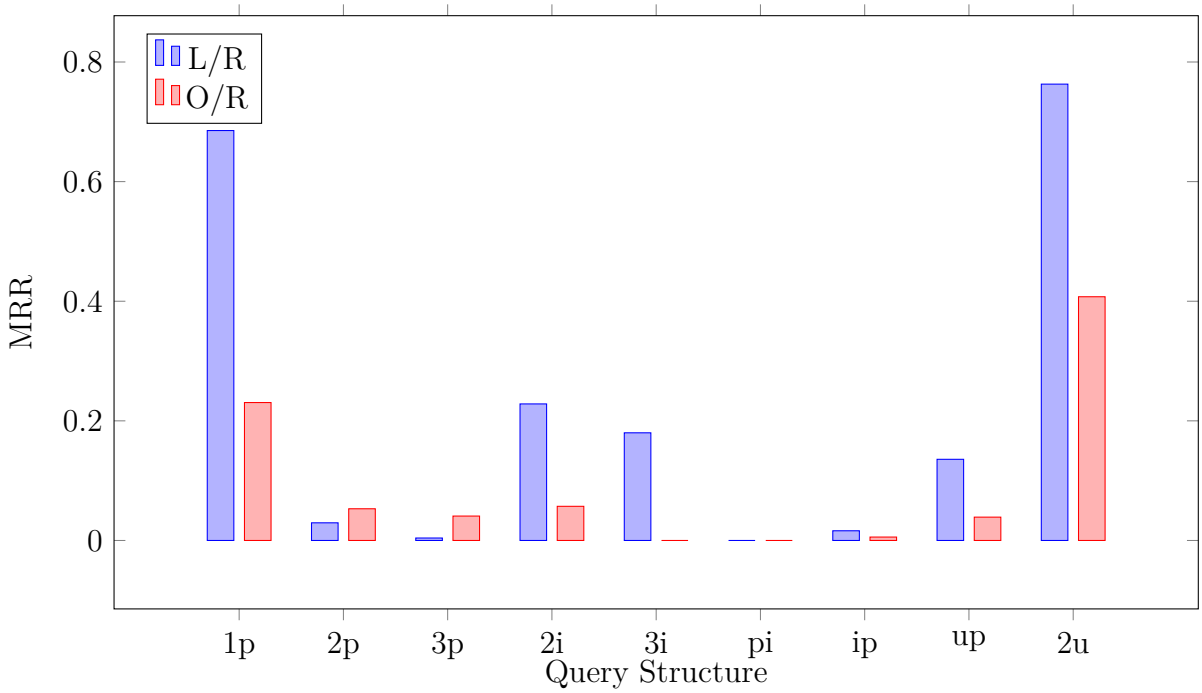


Figure 4.8: DBPedia15k Dataset with TransE, showing impact of **predictions**: o-MRR for L/R and O/R

Experiment discussion

The results for the family dataset, summarized in table 4.11, reveal intriguing patterns about the effects of different query structures and validation techniques on the embedding scores. These differences are vividly illustrated in the bar charts in figs. 4.1 to 4.4, while we detail the changes in o-MRR and Hits@3 scores in tables 4.12 and 4.13.

The query structure '3i' stands out with notably lower scores. A possible explanation might be that queries in '3i' can degenerate, for example, by inadvertently asking for cross-gender relationships. Such queries are less problematic for projection queries, where each query's target is another's input. While for intersection, we need to find entities that are answers to all atoms in that particular query.

When we consider the effect of rewriting, we notice a general increase in both o-MRR and Hits@3 scores across all query structures (table 4.12 and fig. 4.1). This observation suggests that rewriting the queries enhances the validation process by expanding the answer sets. We can see remarkable improvements for the '1p' and '2u' query structures, where the scores rise by at least 50%. Notably, this beneficial effect of rewriting persists even when we perform the validation using the online version of the dataset (fig. 4.1).

However, the benefits of making predictions are less pronounced and vary with the query structure. For instance, fig. 4.3 shows a minor increase in scores for all query structures except for 'up', where the scores decline significantly. The decline suggests that the entities retrieved from the local dataset might only partially align with those retrieved from the online KG. The differences are minimal when comparing the rewritten sets using both local and online versions (table 4.11). This minimal difference indicates a high level of similarity between the answer sets obtained through the two validation methods, resulting in comparative evaluations of the predicted entities.

The results for the DBPedia15k dataset with TransE, our top choice model, are depicted in table 4.14. We detail the changes in scores brought about by rewriting and the use of KGEs in tables 4.15 and 4.16 and illustrated in figs. 4.5 to 4.8. To improve readability, we omitted the numbers within the chart to prevent label overlaps.

Upon rewriting the queries, we see improvement when validating using the local dataset, as depicted in fig. 4.5. Notably, the scores are low for every query structure involving projection, which can be attributed to the generation of queries that exist in different branches of our TBox hierarchy and hence share no common entity answers. However, we still see an improvement compared to using the online KG for validation, as illustrated in fig. 4.6. Except for '1p' and '2u', results across all query structures are reasonably consistent. It is important to recall that '2u' essentially comprises two '1p' queries with unified results.

When considering KGE prediction, we observe a decline in performance when comparing the results validated against the online KG to those validated against the local KG. This decline is evident in table 4.16 and figs. 4.7 and 4.8. Here, both the initial and rewritten query sets perform worse with the online KG validation. The worsening suggests that the list of entities answering a particular query in the online KG differs from that in the local KG, supporting our hypothesis of KG discrepancies. As stated in the dataset introduction, our local version of DBPedia15k contains numerous erroneous triples inconsistent with the online KG. Unfortunately, this undermines the credibility of the results derived from this dataset.

Considering both rewriting and prediction aspects across both datasets, it is evident that rewriting has a clear advantage. Rewriting adheres to the TBox and is logically distinct from the embedding process, as it occurs before we infer the queries through our pipeline. The query rewriting implementation was primarily the aspect we aimed to explore in this work. We also note improvements in KGE predictions for all query

structures in the family dataset (except 'up' on the local KG), and for all structures in DBPedia15k, except for the 1p and 2u structures that experienced a significant decline (likely due to a partial validation list obtained from the API).

To answer the research question more specifically, we must remember that the prediction is made using our KGE models trained on our local KGs, namely the Family dataset and DBPedia15k. We generate the list of entities that answer our query using four different strategies - L/I, L/R, O/I and O/R. Here, L/I is the list of entities obtained from a standard KG lookup, i.e., directly querying the KG. Consequently, the L/I score forms the baseline, and other validation methods should be interpreted relative to this baseline score. Rewriting demonstrated an apparent increase for all local validation methods, and in most cases, also with online validation. The advantages of KGE predictions could have been more evident due to the reasons previously mentioned regarding discrepancies. When we combine the aspects of query rewriting and KGE predictions, we arrive at the O/R score, which validates the predictions with rewritings and the online KG, assisting in discovering new accurately predicted entities. We present the final results for the best performing models in tables 4.17 and 4.18 and figs. 4.9 and 4.10.

In the case of the Family dataset, integrating query rewriting with KGEs resulted in an improvement over a standard KG lookup (table 4.17 and fig. 4.9). With DBPedia15k, there was an increase for '2p' and '3p', but other query structures experienced worse results; hence we cannot discern an overall improvement for this dataset using our framework. Therefore, while we observed a significant improvement with query rewriting across both datasets, we also saw that when considering the entire framework, the Family dataset shined and DBPedia15k suffered.

Table 4.17: Delta values (Δ) between Online KG prediction using rewritten queries (O/R) and Local KG prediction using solely initial query (L/I) for the dataset family and model **RotatE** (dim: 192, epoch: 24). Positive values indicate that O/R outperformed L/I.

Query Structure	Hits@3 Delta (O/R - L/I)	o-MRR Delta (O/R - L/I)
1p	0.2400	0.2400
2p	0.0400	0.0350
3p	0.0667	0.0843
2i	0.0400	0.0400
3i	0.0400	0.0400
pi	0.0134	0.0400
ip	0.0667	0.0800
up	0.0133	-0.0031
2u	0.3067	0.3097

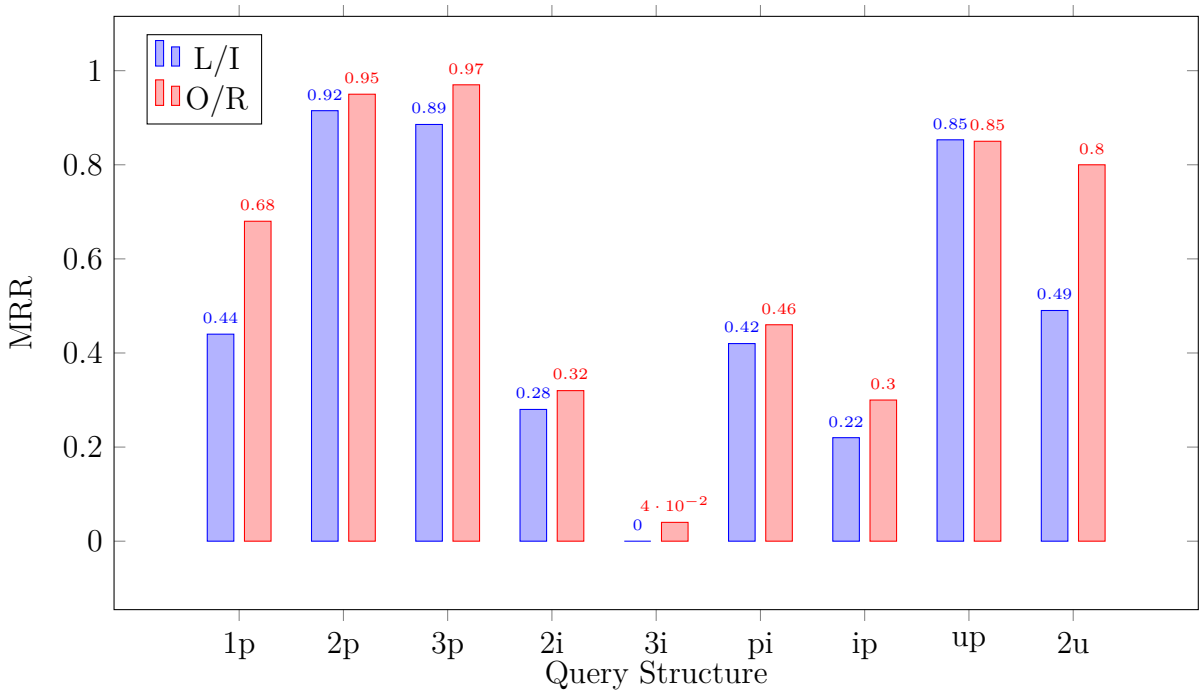


Figure 4.9: Family Dataset with RotatE, showing impact of **predictions**: o-MRR for L/I and O/R

Table 4.18: Delta values (Δ) between Online KG prediction using rewritten queries (O/R) and Local KG prediction using solely initial query (L/I) for the dataset DBPedia15k and model **TransE** (dim: 192, epoch: 24). Positive values indicate that O/R outperformed L/I.

Query Structure	Hits@3 Delta (O/R - L/I)	o-MRR Delta (O/R - L/I)
1p	-0.0533	-0.2385
2p	0.0267	0.0235
3p	0.0133	0.0367
2i	-0.0533	-0.0512
3i	-0.026	-0.02
pi	0	0
ip	0	-0.0105
up	-0.08	-0.0967
2u	-0.28	-0.2622

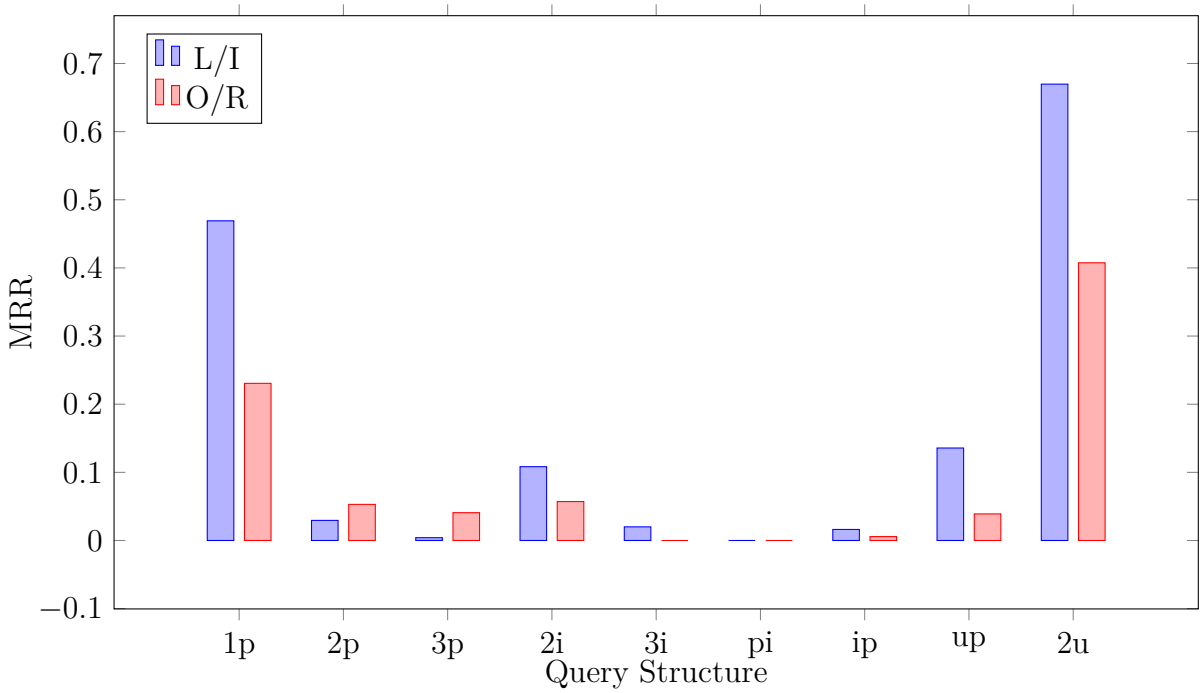


Figure 4.10: DBPedia15k Dataset with TransE, showing impact of **predictions**: o-MRR for L/I and O/R

4.2.5 Question 4: How do we interpret and compare our results fairly?

In this final research question, we delve into how to interpret and fairly compare our results.

Every list of entities answering a specific query has an associated score. For example, we extracted two entities for a specific query from the results. As shown in table 4.20, there is a column for scores. However, these scores lack a mapping to the probability or likelihood of accurate prediction. Remember that different models employ different evaluation functions; hence the scores are calculated diversely. Additionally, we perform T-norms for projection and intersection for queries with several atoms and T-conorms on these score values when performing unification (disjunction).

Entity Label	Origin	Score	L/I	L/R	O/I	O/R
Q100246	q(?w) :- P22(?_,?w)	0.0125520453	false	false	false	true
Q1040807	q(?w) :- P25(?_,?w)	0.0058709825	false	false	false	true

Table 4.20: Updated entity output of the query example: q(?w) :- Q7566(?w)

Although these scores do not map to the probability of their accuracy, we can compare the scores within each model environment to determine the best prediction, irrespective of the weight of that prediction. Consequently, we can calculate the top k entities from the prediction pipeline for each model configuration, but we cannot compare their prediction quality to our other models. Thus, to conduct this experiment, we had to make some assumptions.

Our first assumption was that all prediction outputs from all models should be considered equally likely to be correct, given that these models are often incomparable in general. We then allow the evaluation calculation of o-MRR and Hits@3 to attest to their quality ultimately. That said, for queries that contain multiple atoms, such as a '3p' atom, we cannot determine the exact path a correct entity has taken to end up correctly in our implementation. This indeterminacy is because we manage *a set* of entities when performing link predictions and receive *a set* of entities as output. However, we have a 'still_valid' boolean in our list of entity predictions, indicating that all intermediate predictions leading to the final one have been valid compared to the validation list of entities. Therefore, we could end up with correctly identified entities that have taken an incorrect intermediate prediction in the query answering process.

We have chosen to employ MRR and Hits@3, motivated by their frequent use in other complex query answering methodologies [49, 48, 23, 4, 6]. Nonetheless, it is essential to recognize that our query answering pipeline differs significantly, making direct comparisons to these complex query answering methods inappropriately. These methods incorporate queries during the model's training phase and utilize grounded queries for each of the nine query structures. They also employ a query generator that assures the generation of only those queries that yield answers. However, these approaches do not accommodate queries from a TBox, an element that we have intentionally included in our pipeline to underscore the influence of query rewriting.

We chose the o-MRR score to provide a meaningful evaluation measure when a correctly identified entity does not rank within the top 3. Despite the utility of these metrics, it is worth noting that they do not incorporate a comparison with expected values, a distinctive characteristic embodied by the AMRI metric. This metric offers a normalized measure over expected values, thereby illuminating the degree of performance enhancement over and above what can be attributed to mere chance.

The main reason we have chosen not to use the AMRI metric in the evaluation of the query answering process is to ensure the employment of metrics that are not only

intuitively comprehensible but also harmonize with the conventional understanding of query answering and the metrics typically used in this realm of research. In this context, o-MRR and Hits@3 serve this purpose well.

In conclusion, individual entity scores should not be compared across models, while we can indeed compare the Hits@3 and o-MRR scores. Our queries are also not grounded, which could yield correct entity results with intermediate false predictions while still giving us a correct entity. Despite this, our scores function in a manner that allows us to demonstrate the impact of query rewriting and KGE predictions.

Chapter 5

Related Work

Research in complex query answering using embeddings has advanced significantly over the past few years. Hamilton et al. [23] introduced a technique for making efficient predictions about conjunctive logical queries on incomplete knowledge graphs in 2018. They successfully embedded graph nodes into a low-dimensional space and implemented logical operations as learned geometric operations, an approach that is linear in time complexity with the number of query variables.

Subsequently, Ren and Leskovec [48] introduced BetaE, a probabilistic embedding framework capable of handling a complete set of first-order logic (FOL) operations: conjunction, disjunction, and negation. BetaE could embed queries and entities as distributions by using probabilistic distributions with bounded support, providing a natural way to model uncertainty. Around the same time, Ren et al. [49] introduced Query2box (Q2B), which utilized box embeddings to perform reasoning over knowledge graphs in vector space. Query2Box handled EPFO queries and was particularly adept at answering complex queries involving existential quantifiers and disjunctions.

When writing, Query Computation Tree Optimization (QTO) proposed by Bai et al. [6] stands as the state-of-the-art method for complex query answering. QTO efficiently finds the optimal solution for sizeable combinatorial search spaces by performing forward-backwards propagation on the query computation tree, achieving superior performance on complex query answering across multiple datasets. As a result, QTO achieves state-of-the-art performance on complex query answering across three datasets, shown in table 5.1.

However, these methods focused primarily on complex query answering without a TBox attached, making them different in approach and goal from the current work. While

Answering Complex Logical Queries on Knowledge Graphs via Query Computation Tree Optimization

Method	avg_p	avg_{ood}	avg_n	1p	2p	3p	2i	3i	pi	ip	2u	up	2in	3in	inp	pin	pni
FB15k																	
GQE	28.0	20.1	-	54.6	15.3	10.8	39.7	51.4	27.6	19.1	22.1	11.6	-	-	-	-	-
Query2Box	38.0	29.3	-	68.0	21.0	14.2	55.1	66.5	39.4	26.1	35.1	16.7	-	-	-	-	-
BetaE	41.6	34.3	11.8	65.1	25.7	24.7	55.8	66.5	43.9	28.1	40.1	25.2	14.3	14.7	11.5	6.5	12.4
CQD-CO	46.9	35.3	-	89.2	25.3	13.4	74.4	78.3	44.1	33.2	41.8	21.9	-	-	-	-	-
CQD-Beam	58.2	49.8	-	89.2	54.3	28.6	74.4	78.3	58.2	67.7	42.4	30.9	-	-	-	-	-
ConE	49.8	43.4	14.8	73.3	33.8	29.2	64.4	73.7	50.9	35.7	55.7	31.4	17.9	18.7	12.5	9.8	15.1
GNN-QE	72.8	68.9	38.6	88.5	**69.3**	58.7	79.7	83.5	69.9	70.4	74.1	61.0	44.7	41.7	42.0	30.1	**34.3**
QTO	**74.0**	**71.8**	**49.2**	**89.5**	67.4	**58.8**	**80.3**	**83.6**	**75.2**	**74.0**	**76.7**	**61.3**	**61.1**	**61.2**	**47.6**	**48.9**	27.5
FB15k-237																	
GQE	16.3	10.3	-	35.0	7.2	5.3	23.3	34.6	16.5	10.7	8.2	5.7	-	-	-	-	-
Query2Box	20.1	15.7	-	40.6	9.4	6.8	29.5	42.3	21.2	12.6	11.3	7.6	-	-	-	-	-
BetaE	20.9	14.3	5.5	39.0	10.9	10.0	28.8	42.5	22.4	12.6	12.4	9.7	5.1	7.9	7.4	3.5	3.4
CQD-CO	21.8	15.6	-	46.7	9.5	6.3	31.2	40.6	23.6	16.0	14.5	8.2	-	-	-	-	-
CQD-Beam	22.3	15.7	-	46.7	11.6	8.0	31.2	40.6	21.2	18.7	14.6	8.4	-	-	-	-	-
FuzzQE	24.0	17.4	7.8	42.8	12.9	10.3	33.3	46.9	26.9	17.8	14.6	10.3	8.5	11.6	7.8	5.2	5.8
ConE	23.4	16.2	5.9	41.8	12.8	11.0	32.6	47.3	25.5	14.0	14.5	10.8	5.4	8.6	7.8	4.0	3.6
GNN-QE	26.8	19.9	10.2	42.8	14.7	11.8	38.3	54.1	31.1	18.9	16.2	13.4	10.0	16.8	9.3	7.2	**7.8**
QTO	**33.5**	**27.6**	**15.5**	**49.0**	**21.4**	**21.2**	**43.1**	**56.8**	**38.1**	**28.0**	**22.7**	**21.4**	**16.8**	**26.7**	**15.1**	**13.6**	5.4
NELL995																	
GQE	18.6	12.5	-	32.8	11.9	9.6	27.5	35.2	18.4	14.4	8.5	8.8	-	-	-	-	-
Query2Box	22.9	15.2	-	42.2	14.0	11.2	33.3	44.5	22.4	16.8	11.3	10.3	-	-	-	-	-
BetaE	24.6	14.8	5.9	53.0	13.0	11.4	37.6	47.5	24.1	14.3	12.2	8.5	5.1	7.8	10.0	3.1	3.5
CQD-CO	28.8	20.7	-	60.4	17.8	12.7	39.3	46.6	30.1	22.0	17.3	13.2	-	-	-	-	-
CQD-Beam	28.6	19.8	-	60.4	20.6	11.6	39.3	46.6	25.4	23.9	17.5	12.2	-	-	-	-	-
FuzzQE	27.0	18.4	7.8	47.4	17.2	14.6	39.5	49.2	26.2	20.6	15.3	12.6	7.8	9.8	11.1	4.9	5.5
ConE	27.2	17.6	6.4	53.1	16.1	13.9	40.0	50.8	26.3	17.5	15.3	11.3	5.7	8.1	10.8	3.5	3.9
GNN-QE	28.9	19.6	9.7	53.3	18.9	14.9	42.4	**52.5**	30.8	18.9	15.9	12.6	9.9	14.6	11.4	6.3	**6.3**
QTO	**32.9**	**24.0**	**12.9**	**60.7**	**24.1**	**21.6**	**42.5**	50.6	**31.3**	**26.5**	**20.4**	**17.9**	**13.8**	**17.9**	**16.9**	**9.9**	5.9

Table 5.1: Table borrowed from [6]: "*Test MRR results on complex query answering across all query types. avg_p is the average on EPFO queries; avg_{ood} is the average on out-of-distribution (OOD) queries; avg_n is the average on queries with negation. Results on Hits@1 are in Appendix G.1.* "

they achieve remarkable results, these methods do not aim to improve query answering using a TBox, which is the core objective of our study.

In parallel to advancements in complex query answering, research on ontology representation has also made significant strides. The KG community have proposed various approaches to embed ontological knowledge into a low-dimensional vector space, from representing relations as regions within the vector space [22], embedding theories in the Description Logic $\mathcal{EL}$++ [32], to embedding ontologies articulated in the $\mathcal{ALC}$ description logic into a real-valued vector space [43].

Most recently, Xiong et al. [60] represented TBoxes in an embedded space where entities are points and concepts are boxes. Here, if an entity point resides within a concept box, it is considered an instance of that concept. This approach has interesting properties, such as the intersection of several concepts answering a conjunctive intersection query, illustrated in fig. 5.1.

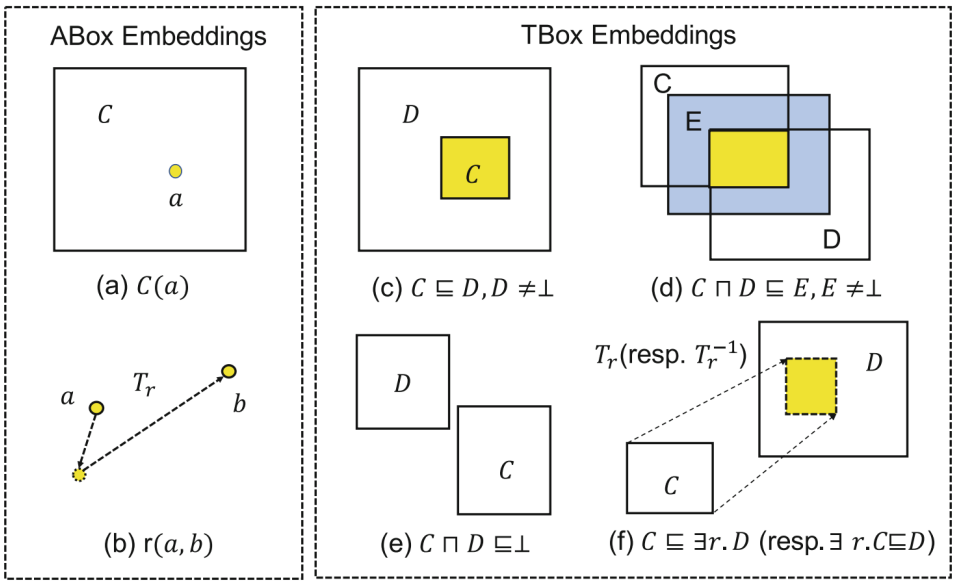


Figure 5.1: BoxEL Embedded Space. Illustration from [60]: *The geometric interpretation of logical statements in ABox (left) and TBox (right) expressed by DL EL++ with BoxEL embeddings.*

While the embedded representation of the TBox in the embedded space offers fascinating potential and has certain parallels to our work, it is not an exact match. The study on DeepProbLog by Manhaeve et al. [40] provides a compelling illustration of how logical rules and neural network models can be used in tandem while maintaining a distinct separation. They demonstrated how Problog rules could be entirely decoupled from low-level perception, exemplified in their case by a convolutional neural network

model predicting on the MNIST dataset. Following this, they applied these logical rules to execute numerical addition without meddling with the prediction labels, or in other words, without affecting the neural network model's learning of numbers.

The benefit of this approach is that the learned weights that comprise the CNN model do not have to accommodate the logic within the model's weights, enabling the model to concentrate purely on predicting the numbers. If the model had to learn addition and identify numbers simultaneously, the results would drastically deteriorate, as evidenced by a substantial increase in the prediction outcome numbers [40]. To draw a parallel with our case, the Problog rules in DeepProbLog would be abstractly equivalent to our ontology, and their CNN model for predicting MNIST numbers would correlate with our KGE models. However, unlike DeepProbLog, we do not incorporate our logic into the training phase; DeepProbLog includes logic within its pipeline towards the scoring function.

Nevertheless, intriguing intersections exist between complex query answering, TBox representation in the embedded space, and the integration of logic into the learning pipeline, demonstrating this field's rich tapestry of possibilities.

Chapter 6

Conclusion

The principal aim of this thesis was to assess the potential of the PerfectRef algorithm in enhancing the outcomes of complex query answering through query rewriting, utilizing knowledge graph embeddings (KGE). Four pivotal research questions guided our exploration. The first delved into the integration of query rewriting with knowledge graph embeddings. The second question focused on the impact of different models on the results achieved through this integration. The third examined the comparative performance of this approach against a conventional KG lookup—finally, the fourth question reflected on the interpretation and implications of our results.

Our initial focus was on integrating queries and embeddings for complex query answering. Even with various methods for complex query answering, we need to consider concepts from a TBox to our knowledge instead of solely utilizing the ABox. In our work, we leveraged two distinct datasets: DBPedia15k, which incorporated the TBox in the set of triples the model was trained on, and the Family dataset, which kept the TBox and ABox separate. In the latter case, predicting concepts became challenging due to the absence of a fair and unbiased method for selecting concept candidates. However, we converted most concepts into roles (properties) with query rewriting, enabling effective query answering. We showed how query rewriting was separate from the embeddings and how to amalgamate query rewriting and complex query answering.

Our experiments employed five distinct models: TransE, DistMult, BoxE, RotatE, and CompGCN. Utilizing the Hits@3 metric on the Family dataset revealed RotatE as the top performer across all projection query structures (1p, 2p, 3p, pi, ip, up). Unexpectedly, TransE outperformed the others on the intersection (2i, 3i) and the 2u

disjunction structure. In the case of DBPedia15k, TransE secured the highest scores across all query structures, except for 'pi', where CompGCN proved superior. TransE was also the model for DBPedia15k that benefitted the most from rewriting. However, no clear correlations emerged between model architectural designs and query structure scores.

To answer the third question, we created four lists of entities to validate the correctness of our predicted entities. The first list derived from a conventional KG Lookup, the second from the rewritten queries set on the lookup to our dataset, the third from the initial query to the full version of the ontology using their API, and the fourth from querying the online KG version with the union of rewritten queries. We observed improved results for all query structures in the Family dataset and DBPedia15k when validating our results using the list of entities from the rewritten sets using our local datasets. However, DBPedia15k decreased for '1p' and '2u' structures when using the full ontology API for validation, potentially explained by the receipt of some HTTP Code 206, Partial Content from our lookups.

For measuring the impact of KGE predictions, we leveraged the online KG to ascertain whether we had identified any new facts not already known to our local datasets, thus being able to correctly identify new facts unknown to our KGE model during training. However, the improvements were not clear. For the Family dataset, we saw a slight increase for all structures when evaluating the original query, while scores were equal primarily when using the rewritten sets. This observation suggests a high similarity between our local rewritten set and the online set for evaluation for the family dataset. For DBPedia15k, results significantly decreased, likely due to discrepancies between our local dataset and the online version and possible partial content delivery from API lookups. Consequently, our results did not demonstrate the advantage of KGE predictions.

When we combined the outcomes of query rewriting and KGE predictions, improvements (or equal results) were evident for every query structure in the Family dataset, while DBPedia15k mainly showed decreases. Given the credibility issues with DBPedia15k, we weighted the results from the Family dataset more, which allowed us to conclude that integrating query rewriting and KGE embeddings improved the results. If we solely consider query rewriting, it is clear that it improved our results for both datasets.

Our findings, while promising, highlight areas for improvement. The query generator currently does not prevent the generation of illogical queries and queries without answers. Also, a more reliable dataset than DBPedia15k would have benefitted the work by

fairly supplementing the family dataset. Despite these issues, our results from the Family dataset indicate that query rewriting enhances complex query answering. This enhancement is evident when datasets are incomplete - meaning it could entail new facts from the TBox. We hope this study fosters further exploration of query rewriting techniques in combination with KGEs. It also underscores the significance of considering the TBox in query answering tasks, contributing to improved accuracy and effectiveness of the query answering process.

6.1 Future Work

This thesis primarily examined the merits of query rewriting using a TBox to broaden the horizons for complex query answering tasks. As a result, our primary pursuit was not refining the query answering pipeline, leaving ample room for advancements and exploration. Our research highlighted the considerable potential of query rewriting, particularly in dealing with ontologies that lack full entailment (i.e., completeness).

In future studies, it would be intriguing to delve into the possibilities of integrating our query rewriting approach with the advancements in knowledge graph embeddings models that embody logic, as illustrated by BoxEL [60]. BoxEL offers a harmonized representation of both the ABox and TBox of an ontology within an embedded space. This approach could couple with existing complex query answering techniques such as Query2Box [49] and QTO [6] Further, an exciting prospect would be to examine the balance between integrating the TBox within the embedded space and keeping the logic entirely separate from the embedding, the latter demonstrated by DeepProbLog [40]. Assessing whether implementing TBoxes into the embedded space significantly complicates the models, thereby impacting the embeddings' general prediction abilities, could provide valuable insights into the trade-off between maintaining a clear separation between logic and embedding. We hypothesize that segregating the logic and TBox from the embedded space permits using a KGE model trained on a conventional dataset for prediction. At the same time, accommodating projection, intersection, and disjunction for query structures outside of the embedded space may allow for more effective generalization in query answering beyond predefined query structures.

As we build upon the foundation of this thesis, an exciting future endeavour could be to expand TBox representations, incorporating the strengths of our query rewriting methods. This approach could create a more robust and efficient knowledge graph-based

query answering system capable of managing a more comprehensive array of queries and providing even more accurate results. Our contribution could be significantly enhanced if we ground the queries, improve the complex query answering pipeline, and ensure that the foundational training of our models is valid subsets of the comprehensive online KGs utilized to evaluate novel facts. Nevertheless, query rewriting can be an essential tool in complex query answering, potentially enhancing the results when a TBox for the corresponding dataset is available. We hope this discovery yields value to the KG community.

List of Acronyms and Abbreviations

AMRI Adjusted Mean Rank Index.

API Application Programming Interface.

CQ Conjunctive Query.

CWA Closed-World Assumption.

DL Description Logic.

EPFO Existential Positive First-order.

FOL First Order Logic.

KB Knowledge Base.

KG Knowledge Graph.

KGE Knowledge Graph Embedding.

L/I List of entities from local KG with initial query.

L/R List of entities from local KG with rewritten queries.

ML Machine Learning.

MR Mean Rank.

MRL Margin Ranking Loss.

NI Negative Inclusion.

NSSALoss Self-adversarial negative sampling loss.

O/I List of entities from online KG with initial query.

O/R List of entities from online KG with rewritten queries.

o-MRR Optimistic Mean Reciprocal Rank.

OWA Open World Assumption.

OWL The W3C Web Ontology Language.

PI Positive Inclusion.

PL Propositional Logic.

RDF Resource Description Framework.

RDFS RDF Schema.

URI Uniform Resource Identifier.

VSM Vector Space Model.

Bibliography

[1] Ralph Abboud, İsmail İlkan Ceylan, Thomas Lukasiewicz, and Tommaso Salvatori. Boxe: A box embedding model for knowledge base completion. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

URL: `https://proceedings.neurips.cc/paper/2020/hash/6dbbe6abe5f14af882ff977fc3f35501-Abstract.html`.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. ISBN 0-201-53771-0.

URL: `http://webdam.inria.fr/Alice/`.

[3] Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Sahand Sharifzadeh, Volker Tresp, and Jens Lehmann. PyKEEN 1.0: A Python Library for Training and Evaluating Knowledge Graph Embeddings. *Journal of Machine Learning Research*, 22(82):1–6, 2021.

URL: `http://jmlr.org/papers/v22/20-825.html`.

[4] Erik Arakelyan, Daniel Daza, Pasquale Minervini, and Michael Cochez. Complex query answering with neural link predictors. *CoRR*, abs/2011.03459, 2020.

URL: `https://arxiv.org/abs/2011.03459`.

[5] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003. ISBN 0-521-78176-0.

[6] Yushi Bai, Xin Lv, Juanzi Li, and Lei Hou. Answering complex logical queries on knowledge graphs via query computation tree optimization. *CoRR*, abs/2212.09567,

2022. doi: 10.48550/arXiv.2212.09567.
URL: https://doi.org/10.48550/arXiv.2212.09567.

[7] Jørgen Bang-Jensen and Gregory Gutin. *Basic Terminology, Notation and Results*, pages 1–34. Springer International Publishing, Cham, 2018. ISBN 978-3-319-71840-8. doi: 10.1007/978-3-319-71840-8_1.
URL: https://doi.org/10.1007/978-3-319-71840-8_1.

[8] Tim Berners-Lee and Mark Fischetti. *Weaving the web - the original design and ultimate destiny of the World Wide Web by its inventor*. HarperBusiness, 2000. ISBN 978-0-06-251587-2.

[9] Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning structured embeddings of knowledge bases. In Wolfram Burgard and Dan Roth, editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press, 2011.
URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3659.

[10] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian 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.

[11] Aaron R. Bradley and Zohar Manna. *The calculus of computation - decision procedures with applications to verification*. Springer, 2007. doi: 10.1007/978-3-540-74113-8.
URL: https://doi.org/10.1007/978-3-540-74113-8.

[12] Tomaz Bratanic. Knowledge graph completion with pykeen and neo4j, Dec 2021.
URL: https://towardsdatascience.com/knowledge-graph-completion-with-pykeen-and-neo4j-6bca734edf43.

[13] Dan Brickley and R.V. Guha. Resource description framework (rdf) schema specification 1.0. W3C note, W3C, September 2001.
URL: https://www.w3.org/2001/sw/RDFCore/Schema/20010913/.

[14] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. Autom. Reason.*, 39(3):385–429, 2007. doi: 10.1007/s10817-007-9078-x.
URL: https://doi.org/10.1007/s10817-007-9078-x.

[15] Tony Coates, Dan Connolly, Diana Dack, Leslie Daigle, Ray Denenberg, Martin Dürst, Paul Grosso, Sandro Hawke, Renato Iannella, Graham Klyne, Larry Masinter, Michael Mealling, Mark Needleman, and Norman Walsh. URIs, URLs, and URNs: Clarifications and recommendations 1.0. W3C note, W3C, September 2001.
URL: https://www.w3.org/TR/2001/NOTE-uri-clarification-20010921/.

[16] Lito Perez Cruz. *Theoremus - A Student's Guide to Mathematical Proofs*. Springer, 2021. ISBN 978-3-030-68374-0. doi: 10.1007/978-3-030-68375-7.
URL: https://doi.org/10.1007/978-3-030-68375-7.

[17] Yuanfei Dai, Shiping Wang, Neal N. Xiong, and Wenzhong Guo. A survey on knowledge graph embedding: Approaches, applications and benchmarks. *Electronics*, 9(5), 2020. ISSN 2079-9292. doi: 10.3390/electronics9050750.
URL: https://www.mdpi.com/2079-9292/9/5/750.

[18] Claudia d'Amato, Nicola Flavio Quatraro, and Nicola Fanizzi. Injecting background knowledge into embedding models for predictive tasks on knowledge graphs. In Ruben Verborgh, Katja Hose, Heiko Paulheim, Pierre-Antoine Champin, Maria Maleshkova, Oscar Corcho, Petar Ristoski, and Mehwish Alam, editors, *The Semantic Web*, pages 441–457, Cham, 2021. Springer International Publishing. ISBN 978-3-030-77385-4.

[19] C. J. Date. *A Guide to the SQL Standard, Second Edition*. Addison-Wesley, 1989. ISBN 978-0-201-50209-1.

[20] Lisa Ehrlinger and Wolfram Wöß. Towards a definition of knowledge graphs. In Michael Martin, Martí Cuquet, and Erwin Folmer, editors, *Joint Proceedings of the Posters and Demos Track of the 12th International Conference on Semantic Systems - SEMANTiCS2016 and the 1st International Workshop on Semantic Change & Evolving Semantics (SuCCESS'16) co-located with the 12th International Conference on Semantic Systems (SEMANTiCS 2016), Leipzig, Germany, September 12-15, 2016*, volume 1695 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.
URL: http://ceur-ws.org/Vol-1695/paper4.pdf.

[21] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. Hermit: An owl 2 reasoner. *J. Autom. Reason.*, 53(3):245–269, 2014.
URL: http://dblp.uni-trier.de/db/journals/jar/jar53.html#GlimmHMSW14.

[22] Víctor Gutiérrez-Basulto and Steven Schockaert. From knowledge graph embedding to ontology embedding? an analysis of the compatibility between vector space representations and rules. In Michael Thielscher, Francesca Toni, and Frank Wolter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona, 30 October - 2 November 2018*, pages 379–388. AAAI Press, 2018.
URL: https://aaai.org/ocs/index.php/KR/KR18/paper/view/18013.

[23] William L. Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. Embedding logical queries on knowledge graphs. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 2030–2041, 2018.
URL: https://proceedings.neurips.cc/paper/2018/hash/ef50c335cca9f340bde656363ebd02fd-Abstract.html.

[24] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge graphs. *ACM Computing Surveys*, 54(4):1–37, jul 2021. doi: 10.1145/3447772.
URL: https://doi.org/10.1145%2F3447772.

[25] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge graphs. *ACM Comput. Surv.*, 54(4), jul 2021. ISSN 0360-0300. doi: 10.1145/3447772.
URL: https://doi.org/10.1145/3447772.

[26] Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun 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 (Volume 1: Long Papers), pages 687–696, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1067.
URL: https://aclanthology.org/P15-1067.

[27] Johanna Jøsang, Ricardo Guimarães, and Ana Ozaki. On the effectiveness of knowledge graph embeddings: a rule mining approach. *CoRR*, abs/2206.00983, 2022. doi: 10.48550/arXiv.2206.00983.
URL: https://doi.org/10.48550/arXiv.2206.00983.

[28] Mayank Kejriwal, Craig A. Knoblock, and Pedro Szekely. *Knowledge Graphs: Fundamentals, Techniques, and Applications*. The MIT Press, Cambridge, MA, 2021.

[29] Hans Kleine Büning and Theodor Lettmann. *Propositional logic - deduction and algorithms*, volume 48 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, 1999. ISBN 978-0-521-63017-7.

[30] Graham Klyne and Jeremy Carroll. Resource description framework (RDF): Concepts and abstract syntax. W3C recommendation, W3C, February 2004. https://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[31] Markus Krötzsch and Gerhard Weikum. Journal of web semantics. *Journal of Web Semantics*, 37-38:53–54, 2016. ISSN 1570-8268. doi: https://doi.org/10.1016/j.websem.2016.04.002.
URL: https://www.sciencedirect.com/science/article/pii/S1570826816300026.

[32] Maxat Kulmanov, Wang Liu-Wei, Yuan Yan, and Robert Hoehndorf. EL embeddings: Geometric construction of models for the description logic EL ++. *CoRR*, abs/1902.10499, 2019.
URL: http://arxiv.org/abs/1902.10499.

[33] Siddharth Krishna Kumar. On weight initialization in deep neural networks, 2017.

[34] Jean-Baptiste Lamy. Owlready: Ontology-oriented programming in python with automatic classification and high-level constructs for biomedical ontologies. *Artif. Intell. Medicine*, 80:11–28, 2017. doi: 10.1016/j.artmed.2017.07.002.
URL: https://doi.org/10.1016/j.artmed.2017.07.002.

[35] Jean-Baptiste Lamy. owlready2. https://bitbucket.org/jibalamy/owlready2/src/master/, 2020.

[36] Jean-Baptiste Lamy. Managing ontologies, 2023.
URL: https://owlready2.readthedocs.io/en/latest/onto.html.

[37] Jean-Baptiste Lamy. Class constructs, restrictions and logical operators, 2023.
URL: https://owlready2.readthedocs.io/en/latest/restriction.html.

[38] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In Blai Bonet and Sven 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.

[39] Ye Liu, Hui Li, Alberto García-Durán, Mathias Niepert, Daniel Oñoro-Rubio, and David S. Rosenblum. MMKG: multi-modal knowledge graphs. *CoRR*, abs/1903.05485, 2019.
URL: http://arxiv.org/abs/1903.05485.

[40] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *CoRR*, abs/1805.10872, 2018.
URL: http://arxiv.org/abs/1805.10872.

[41] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26, page 3111–3119. Curran Associates, Inc., 2013.
URL: https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

[42] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In Lise Getoor and Tobias 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.

[43] Özgür Lütfü Özçep, Mena Leemhuis, and Diedrich Wolter. Cone semantics for logics with negation. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages

1820–1826. ijcai.org, 2020. doi: 10.24963/ijcai.2020/252.
URL: https://doi.org/10.24963/ijcai.2020/252.

[44] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.
URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[45] Peter Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL web ontology language semantics and abstract syntax. W3C recommendation, W3C, February 2004. https://www.w3.org/TR/2004/REC-owl-semantics-20040210/.

[46] Aleksandar Pavlović and Emanuel Sallinger. Expressive: A spatio-functional embedding for knowledge graph completion. In *The Eleventh International Conference on Learning Representations*, 2023.
URL: https://openreview.net/forum?id=xkev3_np08z.

[47] Nicola Flavio Quatraro. Transrowl-hrs. https://github.com/Keehl-Mihael/TransROWL-HRS, 2020.

[48] Hongyu Ren and Jure Leskovec. Beta embeddings for multi-hop logical reasoning in knowledge graphs. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.
URL: https://proceedings.neurips.cc/paper/2020/hash/e43739bba7cdb577e9e3e4e42447f5a5-Abstract.html.

[49] Hongyu Ren, Weihua Hu, and Jure Leskovec. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.
URL: https://openreview.net/forum?id=BJgr4kSFDS.

[50] Baoxu Shi and Tim Weninger. Open-world knowledge graph completion. In Sheila A. McIlraith and Kilian 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 1957–1964. AAAI Press, 2018.
URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16055.

[51] Steffen Staab and Rudi Studer, editors. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, 2004. ISBN 3-540-40834-7.

[52] Rudi Studer, V. Richard Benjamins, and Dieter Fensel. Knowledge engineering: Principles and methods. *Data Knowl. Eng.*, 25(1-2):161–197, 1998. doi: 10.1016/S0169-023X(97)00056-6.
URL: https://doi.org/10.1016/S0169-023X(97)00056-6.

[53] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. 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=HkgEQnRqYQ.

[54] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha P. Talukdar. Composition-based multi-relational graph convolutional networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.
URL: https://openreview.net/forum?id=BylA_C4tPr.

[55] Evan Wallace and Christine Golbreich. OWL 2 web ontology language new features and rationale (second edition). W3C recommendation, W3C, December 2012. https://www.w3.org/TR/2012/REC-owl2-new-features-20121211/.

[56] Meihong Wang, Linling Qiu, and Xiaoli Wang. A survey on knowledge graph embeddings for link prediction. *Symmetry*, 13(3), 2021. ISSN 2073-8994. doi: 10.3390/sym13030485.
URL: https://www.mdpi.com/2073-8994/13/3/485.

[57] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In Carla E. Brodley and Peter 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.

[58] Wikipedia. Knowledge graph — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Knowledge%20graph&oldid=1138959356, 2023. [Online; accessed 18-February-2023].

[59] Paul Witherell, Sundar Krishnamurty, Ian Grosse, and Jack Wileden. Improved knowledge management through first-order logic in engineering design ontologies. *AI EDAM*, 24:245–257, 05 2010. doi: 10.1017/S0890060409990096.

[60] Bo Xiong, Nico Potyka, Trung-Kien Tran, Mojtaba Nayyeri, and Steffen Staab. Faithful embeddings for el++ knowledge bases. In Ulrike Sattler, Aidan Hogan, C. Maria Keet, Valentina Presutti, João Paulo A. Almeida, Hideaki Takeda, Pierre Monnin, Giuseppe Pirrò, and Claudia d'Amato, editors, *The Semantic Web - ISWC 2022 - 21st International Semantic Web Conference, Virtual Event, October 23-27, 2022, Proceedings*, volume 13489 of *Lecture Notes in Computer Science*, pages 22–38. Springer, 2022. doi: 10.1007/978-3-031-19433-7_2.
URL: https://doi.org/10.1007/978-3-031-19433-7_2.

[61] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.
URL: http://arxiv.org/abs/1412.6575.

[62] Amrapali Zaveri, Anisa Rula, Andrea Maurino, Ricardo Pietrobon, Jens Lehmann, and Sören Auer. Quality assessment for linked data: A survey. *Semantic Web*, 7(1): 63–93, 2016. doi: 10.3233/SW-150175.
URL: https://doi.org/10.3233/SW-150175.

Appendix A

Logic fundamentals

A.1 Logic

Logic is a ubiquitous term found in many different areas of knowledge - like philosophy, mathematics, reasoning, and computing [5]. As a matter of fact, it is a field of knowledge in itself. Logic immediately becomes relevant in artificial intelligence as the need for knowledge representation is highly evident.

A.1.1 Propositional Logic

In the realm of Propositional Logic (PL), statements are referred to as propositional formulas, and these encompass propositional variables known as atoms [29]. The values that these atoms can take are restricted to boolean expressions, specifically true or false. The convention for denoting atoms is to use capital letters such as A and B.

These atoms are linked together by a variety of logic connectives, as shown in table A.1. The connectives include conjunction, disjunction, negation, implication, and double implication (equivalency).

Each connective serves a specific function. For instance, the conjunction (AND) connective ensures that the combined proposition is true only if both A and B are true. In contrast, the disjunction (OR) connective makes the compound proposition true if either A or B, or both, are true.

Name	Symbol	Connection	Meaning
Negation (NOT)	$\neg$	$\neg A$	$(\neg A)$ is true if A is false.
Conjunction (AND)	$\wedge$	$A \wedge B$	$(A \wedge B)$ is true if both A and B are true; otherwise, false.
Disjunction (OR)	$\vee$	$A \vee B$	$(A \vee B)$ is true if either A or B is true, or both true, otherwise false.
Exclusive OR (XOR)	$\oplus$	$A \oplus B$	$(A \oplus B)$ is true if either A or B is true, otherwise false.
Implication	$\implies$	$A \implies B$	if A occur, then B occurs.
Double implication	$\Leftrightarrow$	$A \Leftrightarrow B$	A occurs if and only if B occurs

Table A.1: Operators in Propositional Logic

To illustrate this with an example in the context of PL, consider the following propositions:

$$A := \text{The temperature outside varies by the time of year } B \quad := \text{Norway is a country} \tag{A.1}$$

Here, we have two distinct propositions, A and B. The truth value of these propositions cannot be determined by the propositions themselves but depends on a given valuation (interpretation). In a common interpretation of these propositions, where A is generally accepted as true due to weather variations throughout the year, and B is accepted as true since Norway is recognized as a country, both A and B would be considered true. However, it is important to note that these truth values are context-dependent and not inherent properties of the propositions.

A.1.2 First Order Logic

PL exhibits certain limitations in terms of its expressivity. As an example, consider the logical progression of the following three statements:

1. All dogs like to walk
2. $Dog(Theo)$

---

3. $LikesWalking(Theo)$

In this progression, the first two statements logically lead to the third. However, such an expression of interconnectedness is beyond the expressive power of PL.

First Order Logic (FOL) enhances this expressiveness by introducing predicates, terms, and quantification [29]. *Predicates* in FOL are analogous to atoms in PL. The term *arity* (also known as valence) refers to the number of arguments or operands that a function or operation accepts, and in this context, it refers to the number of terms a predicate contains. A predicate with an arity of 0 is an atom in PL as it doesn't expect any terms.

Terms in FOL can be constants, variables, or functions.

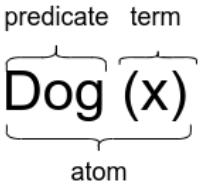


Figure A.1: FOL atom

In fig. A.1, the unary (one term) predicate *Dog* includes the variable x, forming an atom. When a constant, such as *Theo*, substitutes the variable x, the atom is said to be grounded [29]. In addition to n-ary predicates, an atom could be $\top$ (for *top*) or $\bot$ (for *bottom*). Literals are defined as atoms or their negated form.

FOL also introduces the universal and existential quantifiers, illustrated in Table A.2. The universal quantifier $\forall x$ means *for all* x, while the existential quantifier $\exists x$ implies *there exists some* x. For instance, to state in FOL that "all people are mortal", it would be formulated as $\forall x.Man(x) \rightarrow Mortal(x)$. Replacing the universal quantifier with the existential, $\exists x.Man(x) \wedge Mortal(x)$, changes the statement to "there exists a mortal person".

Name	**Symbol**	**Connection**	**Meaning**
Universal quantifier	$\forall$	$\forall x.P(x)$	For all x, $P(x)$ holds.
Existential quantifier	$\exists$	$\exists x.P(x)$	There exists an x such that $P(x)$ holds.

Table A.2: Quantification in First-order Logic

A FOL formula integrates literals and quantifiers with the logical connectives inherited from PL [11]. Following is a FOL formula for the sake of illustration,

$$\forall x.P(f(x),x) \rightarrow (\exists y.P(f(g(x,y)),g(x,y))) \wedge Q(x,f(x)).$$

This formula reads: for all x, if $P(f(x),x)$ holds, then there exists a y such that both $P(f(g(x,y)),g(x,y))$ and $Q(x,f(x))$ hold.

Conjunctive Queries

A critical aspect of FOL emphasized in this thesis is the concept of CQs. The essence of queries lies in their deep-rooted connection with logic. Every query essentially constitutes two components we've already encountered – variables and FOL formulas.

To construct the context of a query, we begin by defining it within the framework of FOL. This is done by selecting a variable of interest and linking it to a FOL formula. This can be formally represented as

$$\{\ \vec{x} \quad | \quad \phi(\vec{x})\ \}.$$

Here, $\vec{x}$ denotes the variable under query and $\phi(\vec{x})$ represents a FOL formula. The arity of the query is dictated by the length of $\vec{x}$.

The structure of a CQ is given by $\{\vec{x} \mid \exists \vec{y}.\text{conj}(\vec{x},\vec{y})\}$, where $\text{conj}(\vec{x},\vec{y})$ is a *conjunction* of atoms. Here, $\vec{x}$ and $\vec{y}$ are vectors of variables [14]. A further extension of this concept leads to a *union of conjunctive queries*, which essentially entails a set of CQs yielding a single collective response for the variables satisfying each CQ.

It is also common to use standard datalog notation [2] when working with CQs. Here, the query $\{\vec{x} \mid \exists \vec{y}.\text{conj}(\vec{x},\vec{y})\}$ is translated as $q(\vec{x'}) \leftarrow \text{conj}'(\vec{x'},\vec{y'})$. In this notation, the term $q(\vec{x'})$ is referred to as the *head* of the query, and $\text{conj}'(\vec{x'},\vec{y'})$ is known as the *body* of the query.

A significant aspect to note here is the presence of *distinguished* variables, denoted by $\vec{x'}$, in the head of the query [14]. These variables also appear in the body $\text{conj}'(\vec{x'},\vec{y'})$, influencing the variables responded to by the body (or FOL Formula). These variables are analogous to the terms following the $SELECT$ statement in SQL [19], meaning the variables we want answered. The variables represented by $\vec{y'}$ in $\text{conj}'(\vec{x'},\vec{y'})$ are termed as *non-distinguished* variables, as they only occur in the body of the query. In the context of FOL formulas, these terms correspond to *bound* variables and *free* variables [16].

Appendix B

PerfectRef implementation running time

B.1 Summation of all natural numbers

The formula for the sum of all natural numbers from 1 to d is given by:

$$S(d) = \frac{d(d+1)}{2}$$

Let's prove this formula using mathematical induction.

Base Case: When $d = 1$, we have

$$S(1) = \frac{1(1+1)}{2} = 1,$$

which is the first natural number.

Induction: We assume that the formula is true for some k. That is, we assume

$$S(k) = \frac{k(k+1)}{2}.$$

We need to prove that it holds for $k+1$. That is, we need to show that

$$S(k+1) = \frac{(k+1)((k+1)+1)}{2}.$$

The left side $S(k+1)$ can be written as $S(k)+(k+1)$, based on the definition of the summation. Substituting the induction hypothesis into this we get:

$$S(k+1) = \frac{k(k+1)}{2} + (k+1).$$

To simplify this, we can find a common denominator and combine terms:

$$S(k+1) = \frac{[k(k+1)+2(k+1)]}{2} = \frac{(k+1)(k+2)}{2}.$$

This completes the induction step and hence the formula

$$S(d) = \frac{d(d+1)}{2} \tag{B.1}$$

is proved to hold for all d.

B.2 Binomial coefficient

B.2.1 Correlation between binomial coefficient and triangular numbers

For the reduction in PerfectRef, we are going to test all combinations of pairs of atoms in our query. The binomial coefficient *d choose 2* (denoted as dC2) gives the sequence of *triangular numbers*: $1, 3, 6, 10, 15, 21, 28, \ldots$.

Here's why:

$$\text{d choose } 2 = \frac{d!}{[(d-2)!2!]}$$

As factorial is defined as the product of all positive integers up to a certain number, we have:

$$\frac{d(d-1)(d-2)!}{[(d-2)!2]}$$

The $(d-2)!$ terms cancel out, leaving us with:

$$\frac{d(d-1)}{2}$$

which is the same as if we change the ordering and add 1 to d

$$\frac{d(d+1)}{2}$$

This formula is precisely the one used to generate the sequence of triangular numbers. Therefore, the binomial coefficient *d choose 2* equals the (d-1)th triangular number.

The dth triangular number is given by the formula

$$T(d) = \frac{d(d+1)}{2} \tag{B.2}$$

B.2.2 Summation over the first d triangular numbers

If we want to sum the first d triangular numbers, we denote it as $S(d)$. The formula for $S(d)$ is given by:

$$S(d) = \frac{d(d+1)(d+2)}{6}$$

Let's show this holds by induction.

Base Case: When $d = 1$, we have

$$S(d) = \frac{1(1+1)(1+2)}{6} = 1,$$

which is the first triangular number.

Induction: We assume that the formula is true for some k. That is, we assume

$$S(k) = \frac{k(k+1)(k+2)}{6}.$$

We need to prove that it holds for $k+1$. That is, we need to show that

$$S(k+1) = \frac{(k+1)(k+2)(k+3)}{6}$$

$$S(k+1) = S(k) + T(k+1) = \frac{k(k+1)(k+2)}{6} + \frac{(k+1)(k+2)}{2}$$

Factoring $(k+1)(k+2)$ from both terms we get:

$$S(k+1) = (k+1)(k+2)[k/6 + 1/2] = \frac{(k+1)(k+2)(k+3)}{6}$$

This completes the induction step and so the formula

$$S(d) = \frac{d(d+1)(d+2)}{6} \tag{B.3}$$

holds for all d.

B.3 Running time PerfectRef

Let d be the number of atoms in the current query. Let t be the number of axioms in $\mathcal{T}$, where $\mathcal{T}$ is the TBox, and let n be the combined number of classes and properties in our ontology. Let T be a function of

In our model, we assume a hypothetical scenario where every class and property may be reformulated to each other, effectively creating a fully connected graph [7]. This means every atom in the query q applies to every axiom in $\mathcal{T}$.

For $d = 0$, we have the base case

$$T(0, t, n) = 0$$

For $d = 1$, we generate t more queries, while iterating through $\mathcal{T}$. Thereafter, each newly added conjunctive query (of t amount) will also run through $\mathcal{T}$ (consisting of t axioms), making this operation use t^2 iterations. This time, no new entailments occur. Since it is already just one atom, there is no reduction, resulting in a total running time of

$$T(1) = 1(t + t^2) + T(0, t, n)$$

When $d = 2$, we end up with $2t$ conjunctive queries, as they will run through $\mathcal{T}$ for every atom. Furthermore, we will also account for possible reductions where the atoms are equal; in the worst case, we will get n reductions, each running through $\mathcal{T}$ to check for further reformulations, yielding nt. Adding the running time from $d-1$, we arrive at

$$T(2) = 2(t + t^2) + nt + T(1, t, n)$$

For $d = 3$, the process yields $3t$ conjunctive queries, as they each run through the $\mathcal{T}$ for every atom. Similarly, reductions, where atoms are equal, are accounted for; in the worst case, $3n$ possible reductions occur, each running through $\mathcal{T}$. The number 3 is from the number of pairs it is possible to reduce, following the formula

$$C(d, r) = \frac{d!}{r!(d-r)!}$$

where d is the number of atoms and $r = 2$ in our case, since we want to test pairwise combinations. Adding the running time from $d - 1$, the total is $T(3) = 3(t + t^2) + C(3, 2)nt + T(2, t, n)$.

In general, for $d = k$ where $k \in \mathbb{N}$, we get kt conjunctive queries and $n(k - 1)$ reductions in the worst case. After this, we add the running time from $d-1$ until we our base case $d = 0$.

$$\begin{aligned}
T(0, t, n) &= 0 \\
T(1, t, n) &= 1(t + t^2) + T(0, t, n) \\
T(2, t, n) &= 2(t + t^2) + C(2, 2)nt + T(1, t, n) \\
T(3, t, n) &= 3(t + t^2) + C(3, 2)nt + T(2, t, n) \\
&\dots \\
T(k, t, n) &= k(t + t^2) + C(k, 2)nt + T(k - 1, t, n)
\end{aligned} \tag{B.4}$$

We see that k follow the natural number sequence, so we can remove the recursion from the equation, and add eq. (B.1). We also see that the other fraction follow the summation of triangular numbers, so we can substitute it with eq. (B.3).

We simplify to

$$T(k,t,n) = \frac{k(k+1)}{2}(t+t^2) + \frac{k(k+1)(k+2)}{6}nt \tag{B.5}$$

Considering the upper found, O, it is difficult to simplify due to several variables which can vary in size, as seen in eq. (B.4). We have the upper bound for our running time shown in eq. (B.6).

$$T(d,t,n) = O\left(\frac{d(d+1)}{2}(t+t^2) + \frac{d(d+1)(d+2)}{6}nt\right) \tag{B.6}$$

Thus, the running time of the PerfectRef implementation is $O(\frac{d(d+1)}{2}(t+t^2) + \frac{d(d+1)(d+2)}{6}nt)$, where d is the number of atoms in the query, t is the number of axioms in the TBox $\mathcal{T}$, and n is the combined number of classes and properties in the ontology. Hence, we see that the running time suffer the most from the length of the query.

Appendix C

Embedding results

Hits@1	*DBPedia*15*k*			*Familyontology*		
Epochs n	$n = 16$	$n = 20$	$n = 24$	$n = 16$	$n = 20$	$n = 24$
TransE						
$dim = 64$	0.0826	0.0731	0.0620	0	0	0
$dim = 128$	0.0148	0.0133	0.0125	0	0	0
$dim = 192$	0.0072	0.0082	0.0098	0	0	0
DistMult						
$dim = 64$	0.0250	0.0266	0.0273	0.4272	0.4473	0.4488
$dim = 128$	0.0377	0.0392	0.0410	0.4875	0.4911	0.4900
$dim = 192$	0.0410	0.0412	0.0426	0.4620	0.4605	0.4566
BoxE						
$dim = 64$	0.2084	0.2118	0.2129	0.3265	0.3710	0.4104
$dim = 128$	0.2149	0.2161	0.2208	0.4563	0.4848	0.4985
$dim = 192$	0.2187	0.2221	**0.2245**	0.4890	0.4950	0.4930
RotatE						
$dim = 64$	0.1327	0.1588	0.1798	0.6563	0.6767	0.6886
$dim = 128$	0.1822	0.2017	0.2139	0.6633	0.6772	0.6821
$dim = 192$	0.2028	0.2203	0.2263	0.6968	0.7112	**0.7181**
CompGCN						
$dim = 64$	0.1838	0.1844	0.1835	0.0646	0.0767	0.0873
$dim = 128$	0.1720	0.1735	0.1731	0.0938	0.1090	0.1224
$dim = 192$	0.1655	0.1680	0.1895	0.0973	0.1226	0.1297

Table C.1: Hits@1 results

Hits@3	*DBPedia*15*k*			*Familyontology*		
Epochs n	$n = 16$	$n = 20$	$n = 24$	$n = 16$	$n = 20$	$n = 24$
TransE						
$dim = 64$	0.2327	0.2349	0.2369	0.2285	0.2713	0.3067
$dim = 128$	0.2490	0.2499	0.2397	0.3765	0.4045	0.4266
$dim = 192$	0.2461	0.2454	0.2530	0.4321	0.4510	0.4637
DistMult						
$dim = 64$	0.0464	0.0518	0.0526	0.8142	0.8722	0.8798
$dim = 128$	0.0692	0.0714	0.0728	0.8912	0.8898	0.8886
$dim = 192$	0.0747	0.0764	0.0799	**0.8948**	0.8944	0.8898
BoxE						
$dim = 64$	0.3055	0.3083	0.3113	0.4609	0.5373	0.5924
$dim = 128$	0.3162	0.3167	0.3210	0.6414	0.6962	0.7140
$dim = 192$	0.3186	0.3234	**0.3265**	0.7285	0.7398	0.7420
RotatE						
$dim = 64$	0.2155	0.2452	0.2744	0.8146	0.8438	0.8610
$dim = 128$	0.2695	0.2952	0.3132	0.8574	0.8759	0.8830
$dim = 192$	0.2978	0.3173	0.3264	0.8631	0.8789	0.8877
CompGCN						
$dim = 64$	0.2722	0.2739	0.2734	0.1208	0.1418	0.1595
$dim = 128$	0.2713	0.2750	0.2762	0.1700	0.1973	0.2173
$dim = 192$	0.2614	0.2630	0.2880	0.1818	0.2202	0.2385

Table C.2: Hits@3 results

Hits@5	*DBPedia*15*k*			*Familyontology*		
Epochs n	$n = 16$	$n = 20$	$n = 24$	$n = 16$	$n = 20$	$n = 24$
TransE						
$dim = 64$	0.2752	0.2777	0.2801	0.3091	0.3588	0.4007
$dim = 128$	0.2900	0.2885	0.2853	0.4665	0.5033	0.5328
$dim = 192$	0.2903	0.2914	0.2969	0.5336	0.5620	0.5802
DistMult						
$dim = 64$	0.0543	0.0607	0.0634	0.8544	0.9095	0.9198
$dim = 128$	0.0817	0.0851	0.0869	0.9293	0.9294	0.9289
$dim = 192$	0.0886	0.0914	0.0952	**0.9347**	0.9332	0.9315
BoxE						
$dim = 64$	0.3275	0.3329	0.3337	0.5163	0.5995	0.6603
$dim = 128$	0.3378	0.3412	0.3441	0.7009	0.7634	0.7842
$dim = 192$	0.3415	0.3443	**0.3507**	0.8062	0.8211	0.8278
RotatE						
$dim = 64$	0.2420	0.2717	0.2982	0.8352	0.8634	0.8784
$dim = 128$	0.2914	0.3153	0.3304	0.8770	0.8936	0.9019
$dim = 192$	0.3162	0.3329	0.3422	0.8809	0.8954	0.9038
CompGCN						
$dim = 64$	0.2918	0.2953	0.2949	0.1488	0.1741	0.1915
$dim = 128$	0.2960	0.3008	0.3046	0.2012	0.2274	0.2489
$dim = 192$	0.2960	0.3056	0.3114	0.2126	0.2506	0.2698

Table C.3: Hits@5 results

Hits@10	*DBPedia*15*k*			*Familyontology*		
Epochs n	$n = 16$	$n = 20$	$n = 24$	$n = 16$	$n = 20$	$n = 24$
TransE						
$dim = 64$	0.3113	0.3167	0.3204	0.4040	0.4612	0.5069
$dim = 128$	0.3277	0.3286	0.3302	0.5538	0.5960	0.6289
$dim = 192$	0.3306	0.3352	0.3404	0.6227	0.6558	0.6797
DistMult						
$dim = 64$	0.0652	0.0734	0.0786	0.8800	0.9282	0.9375
$dim = 128$	0.0992	0.1039	0.1080	0.9470	0.9485	0.9497
$dim = 192$	0.1086	0.1138	0.1185	**0.9534**	0.9523	0.9513
BoxE						
$dim = 64$	0.3598	0.3636	0.3657	0.5809	0.6664	0.7258
$dim = 128$	0.3701	0.3740	0.3744	0.7535	0.8170	0.8404
$dim = 192$	0.3718	0.3774	**0.3852**	0.8642	0.8836	0.8929
RotatE						
$dim = 64$	0.2712	0.3036	0.3251	0.8499	0.8757	0.8892
$dim = 128$	0.3186	0.3398	0.3536	0.8886	0.9026	0.9110
$dim = 192$	0.3411	0.3571	0.3692	0.8919	0.9049	0.9119
CompGCN						
$dim = 64$	0.3160	0.3212	0.3227	0.1943	0.2214	0.2409
$dim = 128$	0.3225	0.3258	0.3325	0.2457	0.2727	0.2955
$dim = 192$	0.3260	0.3348	0.3381	0.2548	0.2930	0.3136

Table C.4: Hits@10 results

MR	*DBPedia15k*			*Familyontology*		
Epochs n	$n = 16$	$n = 20$	$n = 24$	$n = 16$	$n = 20$	$n = 24$
TransE						
$dim = 64$	960.7	867.0	809.8	3367.4	3115.3	2954.3
$dim = 128$	876.3	812.4	772.5	3274.2	3024.0	2913.4
$dim = 192$	849.1	808.1	778.9	3084.8	2936.3	2715.4
DistMult						
$dim = 64$	4859.0	4754.9	4537.3	1842.8	1553.3	1510.9
$dim = 128$	4318.9	4083.4	3748.0	1494.8	1468.2	1454.8
$dim = 192$	3971.2	3794.1	3713.8	1357.5	**1322.4**	1342.2
BoxE						
$dim = 64$	452.5	434.8	425.7	4472.9	4267.2	4136.3
$dim = 128$	418.5	414.2	406.0	4139.2	3918.1	3847.1
$dim = 192$	408.8	400.7	**396.0**	3436.2	3364.6	3252.6
RotatE						
$dim = 64$	1601.2	1440.3	1321.3	2622.0	2522.1	2480.4
$dim = 128$	1401.6	1259.9	1157.8	2468.6	2420.8	2376.3
$dim = 192$	1260.4	1143.0	1061.3	3223.7	3281.4	3233.2
CompGCN						
$dim = 64$	930.4	866.7	831.9	3399.5	3088.9	2893.9
$dim = 128$	944.4	886.5	843.7	3084.1	2844.8	2667.8
$dim = 192$	984.9	914.1	856.3	3224.9	2967.5	2767.9

Table C.5: MR results. DBPEDIA15k Triples count = 32 174, Family ontology triples count = 48692

o-MRR	*DBPedia*15*k*			*Familyontology*		
Epochs n	$n = 16$	$n = 20$	$n = 24$	$n = 16$	$n = 20$	$n = 24$
TransE						
$dim = 64$	0.1696	0.1667	0.1623	0.1450	0.1669	0.1847
$dim = 128$	0.1424	0.1425	0.1395	0.2129	0.2278	0.2398
$dim = 192$	0.1385	0.1399	0.1442	0.2395	0.2503	0.2579
DistMult						
$dim = 64$	0.0399	0.0439	0.0456	0.6227	0.6591	0.6636
$dim = 128$	0.0598	0.0623	0.0646	0.6878	0.6893	0.6884
$dim = 192$	0.0652	0.0670	0.0697	0.6773	0.6762	0.6727
BoxE						
$dim = 64$	0.2688	0.2727	0.2747	0.4143	0.4744	0.5203
$dim = 128$	0.2776	0.2793	0.2837	0.5638	0.6047	0.6208
$dim = 192$	0.2808	0.2848	**0.2884**	0.6235	0.6327	0.6338
RotatE						
$dim = 64$	0.1852	0.2132	0.2370	0.7379	0.7615	0.7752
$dim = 128$	0.2356	0.2573	0.2719	0.7607	0.7758	0.7820
$dim = 192$	0.2587	0.2771	0.2852	0.7805	0.7949	**0.8024**
CompGCN						
$dim = 64$	0.2376	0.2395	0.2390	0.1107	0.1283	0.1430
$dim = 128$	0.2322	0.2349	0.2363	0.1495	0.1708	0.1881
$dim = 192$	0.2264	0.2303	0.2494	0.1564	0.1884	0.2014

Table C.6: o-MRR results

AMRI	*DBPedia*15*k*			*Familyontology*		
Epochs n	$n = 16$	$n = 20$	$n = 24$	$n = 16$	$n = 20$	$n = 24$
TransE						
$dim = 64$	0.8327	0.8490	0.8590	0.9420	0.9463	0.9491
$dim = 128$	0.8474	0.8585	0.8655	0.9436	0.9479	0.9498
$dim = 192$	0.8521	0.8593	0.8644	0.9469	0.9494	0.9532
DistMult						
$dim = 64$	0.1532	0.1713	0.2093	0.9682	0.9732	0.9740
$dim = 128$	0.2473	0.2884	0.3468	0.9742	0.9747	0.9749
$dim = 192$	0.3079	0.3388	0.3528	0.9766	**0.9772**	0.9769
BoxE						
$dim = 64$	0.9212	0.9243	0.9259	0.9230	0.9265	0.9288
$dim = 128$	0.9272	0.9279	0.9293	0.9287	0.9325	0.9338
$dim = 192$	0.9289	0.9303	0.9311	0.9408	0.9421	0.9440
RotatE						
$dim = 64$	0.7210	0.7491	0.7698	0.9548	0.9566	0.9573
$dim = 128$	0.7558	0.7805	0.7983	0.9575	0.9583	0.9591
$dim = 192$	0.7804	0.8009	0.8151	0.9445	0.9435	0.9443
CompGCN						
$dim = 64$	0.8379	0.8491	**0.8551**	0.9415	0.9468	0.9502
$dim = 128$	0.8355	0.8456	0.8531	0.9469	0.9510	0.9540
$dim = 192$	0.8284	0.8408	0.8509	0.9445	0.9489	0.9523

Table C.7: AMRI results

Appendix D

Testcase results

D.1 Family Dataset

Table D.1: Summary of results for different query structures for the dataset family and model **BoxE** (dim: 192, epoch: 24). All query structures have a count of 25 different queries. L/I: Validation using entities from local KG lookup on the initial query, L/R: Validation using entities from local KG lookup on the rewritten queries, O/I: Validation using entities from online KG lookup on the initial query, O/R: Validation using entities from online KG lookup with the rewritten queries.

	Hits@3				o-MRR			
Query Structure	L/I	L/R	O/I	O/R	L/I	L/R	O/I	O/R
1p	0.4400	0.6800	0.4400	0.6800	0.4400	0.6800	0.4400	0.6800
2p	0.6533	0.7733	0.7600	0.8000	0.8632	0.9617	0.9603	1.0000
3p	0.6533	0.8133	0.7733	0.7867	0.7785	0.9333	0.8261	0.8658
2i	0.2667	0.3467	0.3067	0.3467	0.2800	0.3600	0.3200	0.3600
3i	0	0.0400	0	0.0400	0	0.0400	0	0.0400
pi	0.2533	0.3600	0.2933	0.2933	0.3133	0.4229	0.3457	0.3433
ip	0.2266	0.2266	0.2666	0.2933	0.2800	0.2800	0.3200	0.3600
up	0.6933	0.8000	0.5199	0.7466	0.8407	0.9000	0.6207	0.9000
2u	0.5333	0.8000	0.6000	0.8000	0.5600	0.8000	0.6000	0.8000

Table D.2: Summary of results for different query structures for the dataset family and model **CompGCN** (dim: 192, epoch: 24). All query structures have a count of 25 different queries. L/I: Validation using entities from local KG lookup on the initial query, L/R: Validation using entities from local KG lookup on the rewritten queries, O/I: Validation using entities from online KG lookup on the initial query, O/R: Validation using entities from online KG lookup with the rewritten queries.

	Hits@3				o-MRR			
Query Structure	L/I	L/R	O/I	O/R	L/I	L/R	O/I	O/R
1p	0.4400	0.6800	0.4400	0.6800	0.4400	0.6800	0.4400	0.6800
2p	0.2400	0.3067	0.2667	0.2667	0.3288	0.3856	0.3888	0.3919
3p	0	0.1599	0.0800	0.0800	0.0170	0.2168	0.1632	0.1672
2i	0.3333	0.4267	0.3733	0.4400	0.4400	0.5600	0.4800	0.5600
3i	0.0533	0.1067	0.0533	0.1067	0.0800	0.1600	0.0800	0.2000
pi	0.1067	0.2133	0.1733	0.1867	0.1247	0.3119	0.2127	0.2319
ip	0.0400	0.0400	0.0533	0.0667	0.0508	0.0508	0.0995	0.1209
up	0.2133	0.2533	0.1600	0.2133	0.3540	0.3841	0.3080	0.3881
2u	0.4933	0.8000	0.4933	0.7467	0.4874	0.8000	0.6000	0.7400

Table D.3: Summary of results for different query structures for the dataset family and model **DistMult** (dim: 192, epoch: 16). All query structures have a count of 25 different queries. L/I: Validation using entities from local KG lookup on the initial query, L/R: Validation using entities from local KG lookup on the rewritten queries, O/I: Validation using entities from online KG lookup on the initial query, O/R: Validation using entities from online KG lookup with the rewritten queries.

	Hits@3				o-MRR			
Query Structure	L/I	L/R	O/I	O/R	L/I	L/R	O/I	O/R
1p	0.4400	0.6800	0.4400	0.6800	0.4400	0.6800	0.4400	0.6800
2p	0.6400	0.7200	0.6667	0.7067	0.7081	0.8100	0.7629	0.7949
3p	0.3200	0.3867	0.4533	0.4667	0.4395	0.4971	0.5930	0.6056
2i	0.2800	0.3600	0.3200	0.3867	0.2880	0.3680	0.3280	0.4080
3i	0.0067	0.1333	0.0933	0.1333	0.1600	0.2800	0.2400	0.3000
pi	0.1600	0.2667	0.2667	0.3067	0.2407	0.3636	0.3747	0.4080
ip	0.2000	0.2000	0.2133	0.2133	0.3057	0.3057	0.3500	0.3557
up	0.3467	0.3733	0.2667	0.3867	0.4415	0.4902	0.3490	0.4769
2u	0.4800	0.7867	0.5867	0.7867	0.5213	0.7800	0.6200	0.7800

Table D.4: Summary of results for different query structures for the dataset family and model **RotatE** (dim: 192, epoch: 24). All query structures have a count of 25 different queries. L/I: Validation using entities from local KG lookup on the initial query, L/R: Validation using entities from local KG lookup on the rewritten queries, O/I: Validation using entities from online KG lookup on the initial query, O/R: Validation using entities from online KG lookup with the rewritten queries.

	Hits@3				o-MRR			
Query Structure	L/I	L/R	O/I	O/R	L/I	L/R	O/I	O/R
1p	0.4400	0.6800	0.4400	0.6800	0.4400	0.6800	0.4400	0.6800
2p	0.8933	0.9600	0.9067	0.9333	0.9150	0.9700	0.9300	0.9500
3p	0.8533	0.9733	0.8800	0.9200	0.8857	1	0.9380	0.9700
2i	0.2800	0.3200	0.3200	0.3200	0.2800	0.3200	0.3200	0.3200
3i	0	0.0400	0	0.0400	0	0.0400	0	0.0400
pi	0.4133	0.5200	0.4133	0.4267	0.4200	0.5200	0.4200	0.4600
ip	0.2000	0.2133	0.2267	0.2667	0.2200	0.2400	0.2600	0.3000
up	0.7867	0.8267	0.5467	0.8000	0.8531	0.8900	0.6180	0.8500
2u	0.4933	0.8000	0.6133	0.8000	0.4903	0.8000	0.6000	0.8000

Table D.5: Summary of results for different query structures for the dataset family and model **TransE** (dim: 192, epoch: 24). All query structures have a count of 25 different queries. L/I: Validation using entities from local KG lookup on the initial query, L/R: Validation using entities from local KG lookup on the rewritten queries, O/I: Validation using entities from online KG lookup on the initial query, O/R: Validation using entities from online KG lookup with the rewritten queries.

	Hits@3				o-MRR			
Query Structure	L/I	L/R	O/I	O/R	L/I	L/R	O/I	O/R
1p	0.3067	0.4267	0.2933	0.3733	0.3307	0.4757	0.3183	0.4106
2p	0.2267	0.3733	0.3067	0.3467	0.3426	0.5590	0.4917	0.5307
3p	0.1733	0.3467	0.4000	0.4133	0.3175	0.4457	0.5509	0.5623
2i	0.3200	0.4533	0.3867	0.4267	0.4667	0.5924	0.5067	0.5567
3i	0.1600	0.2533	0.1867	0.2400	0.3000	0.4733	0.3600	0.4533
pi	0.3200	0.4533	0.3867	0.3867	0.6113	0.7901	0.6596	0.6778
ip	0.2267	0.2267	0.3600	0.3467	0.3792	0.3792	0.4711	0.4853
up	0.2667	0.3733	0.2800	0.4267	0.5128	0.6181	0.3953	0.5540
2u	0.4533	0.6133	0.5600	0.6000	0.5036	0.6391	0.5800	0.6223

D.2 DBPedia15k

Table D.6: Summary of results for different query structures for the dataset dbpedia15k and model BoxE (dim: 192, epoch: 24). All query structures have a count of 25 different queries. L/I: Validation using entities from local KG lookup on the initial query, L/R: Validation using entities from local KG lookup on the rewritten queries, O/I: Validation using entities from online KG lookup on the initial query, O/R: Validation using entities from online KG lookup with the rewritten queries.

	Hits@3		Hits@3		o-MRR		o-MRR	
Query Structure	L/I	L/R	O/I	O/R	L/I	L/R	O/I	O/R
1p	0.3067	0.4267	0.1333	0.1200	0.3909	0.5342	0.2430	0.2293
2p	0.0133	0.0133	0.0400	0.0400	0.0536	0.0536	0.1049	0.1049
3p	0	0	0	0	0.0040	0.0040	0.0014	0.0014
2i	0.0800	0.0800	0	0	0.1024	0.1120	0.0100	0.0024
3i	0.0267	0.0267	0	0	0.0200	0.0400	0	0.0008
pi	0	0	0	0	0	0	0	0
ip	0.0133	0.0133	0.0133	0.0133	0.0181	0.0181	0.0133	0.0133
up	0.0400	0.0400	0.0133	0.0133	0.1198	0.1198	0.0374	0.0531
2u	0.3200	0.3867	0.0933	0.1067	0.4836	0.5636	0.2451	0.2214

Table D.7: Summary of results for different query structures for the dataset dbpedia15k and model CompGCN (dim: 192, epoch: 24). All query structures have a count of 25 different queries. L/I: Validation using entities from local KG lookup on the initial query, L/R: Validation using entities from local KG lookup on the rewritten queries, O/I: Validation using entities from online KG lookup on the initial query, O/R: Validation using entities from online KG lookup with the rewritten queries.

	Hits@3		Hits@3		o-MRR		o-MRR	
Query Structure	L/I	L/R	O/I	O/R	L/I	L/R	O/I	O/R
1p	0.3600	0.4667	0.2133	0.1200	0.4770	0.6355	0.3460	0.2124
2p	0	0	0	0	0.0153	0.0153	0.0186	0.0186
3p	0	0	0	0	0	0	0.0009	0.0009
2i	0.0800	0.1200	0.0267	0.0400	0.1364	0.2124	0.0294	0.0644
3i	0.0400	0.0933	0	0.0133	0.0400	0.1733	0	0.0200
pi	0	0	0	0	0.0033	0.0033	0.0025	0.0025
ip	0	0	0	0	0.0032	0.0032	0.0013	0.0013
up	0.0133	0.0133	0.0267	0.0133	0.0835	0.0835	0.0344	0.0366
2u	0.3867	0.4533	0.2533	0.1333	0.5645	0.6079	0.4060	0.2591

Table D.8: Summary of results for different query structures for the dataset dbpedia15k and model DistMult (dim: 192, epoch: 24). All query structures have a count of 25 different queries. L/I: Validation using entities from local KG lookup on the initial query, L/R: Validation using entities from local KG lookup on the rewritten queries, O/I: Validation using entities from online KG lookup on the initial query, O/R: Validation using entities from online KG lookup with the rewritten queries.

	Hits@3		Hits@3		o-MRR		o-MRR	
Query Structure	L/I	L/R	O/I	O/R	L/I	L/R	O/I	O/R
1p	0.4533	0.5867	0.2267	0.0533	0.5027	0.6457	0.4027	0.1855
2p	0.0267	0.0267	0.0267	0.0267	0.0555	0.0555	0.0614	0.0614
3p	0	0	0	0	0.0027	0.0027	0.0017	0.0017
2i	0.1200	0.1200	0	0	0.1328	0.1536	0.0070	0.0021
3i	0.0400	0.0400	0	0	0.0400	0.0800	0	0
pi	0	0	0	0	0.0027	0.0027	0.0013	0.0013
ip	0	0	0	0	0.0041	0.0041	0.0010	0.0010
up	0.0800	0.0800	0.0400	0.0400	0.1469	0.1469	0.0670	0.0657
2u	0.4667	0.5333	0.2267	0.1200	0.5722	0.5988	0.3860	0.2462

Table D.9: Summary of results for different query structures for the dataset dbpedia15k and model RotatE (dim: 192, epoch: 24). All query structures have a count of 25 different queries. L/I: Validation using entities from local KG lookup on the initial query, L/R: Validation using entities from local KG lookup on the rewritten queries, O/I: Validation using entities from online KG lookup on the initial query, O/R: Validation using entities from online KG lookup with the rewritten queries.

	Hits@3		Hits@3		o-MRR		o-MRR	
Query Structure	L/I	L/R	O/I	O/R	L/I	L/R	O/I	O/R
1p	0.3200	0.3867	0.1067	0.1200	0.4744	0.5697	0.2739	0.3382
2p	0	0	0	0	0.0066	0.0066	0.0116	0.0116
3p	0	0	0	0	0	0	0	0
2i	0.0933	0.0933	0.0267	0.0133	0.1036	0.1096	0.0378	0.0235
3i	0.0267	0.0267	0	0	0.02	0.0527	0	0
pi	0	0	0	0	0	0	0	0
ip	0	0	0	0	0.0055	0.0055	0.0037	0.0037
up	0.0400	0.0400	0.0267	0.0267	0.1283	0.1283	0.0854	0.0864
2u	0.3600	0.3733	0.1733	0.1067	0.5386	0.5586	0.3099	0.2889

Table D.10: Summary of results for different query structures for the dataset dbpedia15k and model **TransE** (dim: 192, epoch: 24). All query structures have a count of 25 different queries. L/I: Validation using entities from local KG lookup on the initial query, L/R: Validation using entities from local KG lookup on the rewritten queries, O/I: Validation using entities from online KG lookup on the initial query, O/R: Validation using entities from online KG lookup with the rewritten queries.

	Hits@3				o-MRR			
Query Structure	L/I	L/R	O/I	O/R	L/I	L/R	O/I	O/R
1p	0.2533	0.6000	0.3600	0.2000	0.4691	0.6854	0.4667	0.2306
2p	0	0	0.0267	0.0267	0.0295	0.0295	0.0530	0.0530
3p	0	0	0.0133	0.0133	0.0041	0.0041	0.0408	0.0408
2i	0.0800	0.1733	0.0133	0.0267	0.1083	0.2283	0.0436	0.0571
3i	0.0260	0.1600	0	0	0.0200	0.1800	0	0
pi	0	0	0	0	0	0	0	0
ip	0	0	0	0	0.0162	0.0162	0.0057	0.0057
up	0.1067	0.1067	0.0267	0.0267	0.1357	0.1357	0.0252	0.0390
2u	0.5867	0.6933	0.4267	0.3067	0.6697	0.7630	0.5131	0.4075

Appendix E

Experiment queries generation

E.1 DBPedia15k

- **1p**:
  - `q(?w) :- http://dbpedia.org/ontology/birthPlace(?x,?w)`
  - $q(?w) :- http://dbpedia.org/ontology/Settlement(?w)$
  - $q(?w) :- http://dbpedia.org/ontology/Governor(?w)$
  - $q(?w) :- http://dbpedia.org/ontology/starring(?x, ?w)$
  - $q(?w) :- http://dbpedia.org/ontology/recordLabel(?w, ?x)$
  - $q(?w) :- http://dbpedia.org/ontology/occupation(?w, ?x)$
  - $q(?w) :- http://schema.org/Place(?w)$
  - $q(?w) :- http://xmlns.com/foaf/0.1/Person(?w)$
  - $q(?w) :- http://www.ontologydesignpatterns.org/ont/dul/DUL.owl\#Agent(?w)$
  - $q(?w) :- http://dbpedia.org/ontology/writer(?w, ?x)$
  - $q(?w) :- http://dbpedia.org/ontology/Place(?w)$
  - $q(?w) :- http://dbpedia.org/ontology/Place(?w)$
  - $q(?w) :- http://dbpedia.org/ontology/Person(?w)$
  - $q(?w) :- http://dbpedia.org/ontology/Island(?w)$
  - $q(?w) :- http://schema.org/Place(?w)$
  - $q(?w) :- http://dbpedia.org/ontology/country(?w, ?x)$
  - $q(?w) :- http://dbpedia.org/ontology/Species(?w)$
  - $q(?w) :- http://dbpedia.org/ontology/spokenIn(?w, ?x)$
  - $q(?w) :- http://dbpedia.org/ontology/formerTeam(?x, ?w)$

- $q(?w)$:-$http://www.wikidata.org/entity/Q215627(?w)$
- $q(?w)$:-$http://dbpedia.org/ontology/Settlement(?w)$
- $q(?w)$:-$http://dbpedia.org/ontology/instrument(?x,?w)$
- $q(?w)$:-$http://dbpedia.org/ontology/Island(?w)$
- $q(?w)$:-$http://dbpedia.org/ontology/genre(?w,?x)$
- $q(?w)$:-$http://dbpedia.org/ontology/Location(?w)$

- **2p**:
  - $q(?w)$:- <http://dbpedia.org/ontology/instrument>$(?y,?x) \cap$ <http://dbpedia.org/ontology/director>$(?w,?y)$
  - $q(?w)$:- $< http://dbpedia.org/ontology/birthPlace > (?x,?y) \cap < http://dbpedia.org/ontology/team > (?w,?y)$
  - $q(?w)$:- $< http://dbpedia.org/ontology/birthPlace > (?x,?y) \cap < http://dbpedia.org/ontology/starring > (?w,?y)$
  - $q(?w)$:- $< http://dbpedia.org/ontology/populationPlace > (?y,?x) \cap < http://dbpedia.org/ontology/influenced > (?y,?w)$
  - $q(?w)$:- $< http://dbpedia.org/ontology/starring > (?x,?y) \cap < http://dbpedia.org/ontology/deathPlace > (?y,?w)$
  - $q(?w)$:- $< http://dbpedia.org/ontology/formerBandMember > (?y,?x) \cap < http://dbpedia.org/ontology/literaryGenre > (?y,?w)$
  - $q(?w)$:- $< http://dbpedia.org/ontology/stateOfOrigin > (?x,?y) \cap < http://dbpedia.org/ontology/recordLabel > (?y,?w)$
  - $q(?w)$:- $< http://dbpedia.org/ontology/influencedBy > (?y,?x) \cap < http://dbpedia.org/ontology/instrument > (?y,?w)$
  - $q(?w)$:- $< http://dbpedia.org/ontology/associatedBand > (?y,?x) \cap < http://dbpedia.org/ontology/deathCause > (?w,?y)$
  - $q(?w)$:- $< http://dbpedia.org/ontology/formerBandMember > (?x,?y) \cap < http://dbpedia.org/ontology/birthPlace > (?y,?w)$
  - $q(?w)$:- $< http://dbpedia.org/ontology/recordLabel > (?x,?y) \cap < http://dbpedia.org/ontology/occupation > (?w,?y)$
  - $q(?w)$:- $< http://dbpedia.org/ontology/birthPlace > (?y,?x) \cap < http://dbpedia.org/ontology/executiveProducer > (?w,?y)$
  - $q(?w)$:- $< http://dbpedia.org/ontology/stylisticOrigin > (?x,?y) \cap < http://dbpedia.org/ontology/influencedBy > (?w,?y)$
  - $q(?w)$:- $< http://dbpedia.org/ontology/genre > (?x,?y) \cap < http://dbpedia.org/ontology/associatedMusicalArtist > (?y,?w)$

- $q(?w) :- <http://dbpedia.org/ontology/recordLabel>(?y, ?x) \cap <http://dbpedia.org/ontology/birthPlace>(?w, ?y)$
- $q(?w) :- <http://dbpedia.org/ontology/associatedBand>(?y, ?x) \cap <http://dbpedia.org/ontology/starring>(?y, ?w)$
- $q(?w) :- <http://dbpedia.org/ontology/formerBandMember>(?y, ?x) \cap <http://dbpedia.org/ontology/associatedMusicalArtist>(?y, ?w)$
- $q(?w) :- <http://dbpedia.org/ontology/instrument>(?y, ?x) \cap <http://dbpedia.org/ontology/stylisticOrigin>(?w, ?y)$
- $q(?w) :- <http://dbpedia.org/ontology/starring>(?x, ?y) \cap <http://dbpedia.org/ontology/league>(?y, ?w)$
- $q(?w) :- <http://dbpedia.org/ontology/recordLabel>(?x, ?y) \cap <http://dbpedia.org/ontology/associatedBand>(?w, ?y)$
- $q(?w) :- <http://dbpedia.org/ontology/starring>(?y, ?x) \cap <http://dbpedia.org/ontology/genre>(?y, ?w)$
- $q(?w) :- <http://dbpedia.org/ontology/almaMater>(?y, ?x) \cap <http://dbpedia.org/ontology/starring>(?y, ?w)$
- $q(?w) :- <http://dbpedia.org/ontology/genre>(?x, ?y) \cap <http://dbpedia.org/ontology/starring>(?w, ?y)$
- $q(?w) :- <http://dbpedia.org/ontology/largestCity>(?x, ?y) \cap <http://dbpedia.org/ontology/associatedBand>(?w, ?y)$
- $q(?w) :- <http://dbpedia.org/ontology/birthPlace>(?y, ?x) \cap <http://dbpedia.org/ontology/timeZone>(?y, ?w)$

- **3p**:
  - $q(?w) :- <http://dbpedia.org/ontology/ethnicGroup>(?x, ?y) \cap <http://dbpedia.org/ontology/distributor>(?y, ?z) \cap <http://dbpedia.org/ontology/distributor>(?z, ?w)$
  - $q(?w) :- <http://dbpedia.org/ontology/starring>(?x, ?y) \cap <http://dbpedia.org/ontology/recordLabel>(?z, ?y) \cap <http://dbpedia.org/ontology/starring>(?z, ?w)$
  - $q(?w) :- <http://dbpedia.org/ontology/award>(?y, ?x) \cap <http://dbpedia.org/ontology/distributingLabel>(?y, ?z) \cap <http://dbpedia.org/ontology/associatedBand>(?z, ?w)$
  - $q(?w) :- <http://dbpedia.org/ontology/populationPlace>(?y, ?x) \cap <http://dbpedia.org/ontology/genre>(?z, ?y) \cap <http:$

$//dbpedia.org/ontology/starring > (?w, ?z)$

- $q(?w) :- < http : //dbpedia.org/ontology/city > (?y, ?x) \cap < http : //dbpedia.org/ontology/instrument > (?y, ?z) \cap < http : //dbpedia.org/ontology/starring > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/hometown > (?x, ?y) \cap < http : //dbpedia.org/ontology/director > (?z, ?y) \cap < http : //dbpedia.org/ontology/stylisticOrigin > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/genre > (?y, ?x) \cap < http : //dbpedia.org/ontology/recordLabel > (?y, ?z) \cap < http : //dbpedia.org/ontology/residence > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/producer > (?y, ?x) \cap < http : //dbpedia.org/ontology/notableWork > (?z, ?y) \cap < http : //dbpedia.org/ontology/countySeat > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/recordLabel > (?x, ?y) \cap < http : //dbpedia.org/ontology/governmentType > (?y, ?z) \cap < http : //dbpedia.org/ontology/recordLabel > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/deathPlace > (?x, ?y) \cap < http : //dbpedia.org/ontology/mainInterest > (?y, ?z) \cap < http : //dbpedia.org/ontology/country > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/affiliation > (?y, ?x) \cap < http : //dbpedia.org/ontology/editing > (?z, ?y) \cap < http : //dbpedia.org/ontology/foundationPlace > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/restingPlace > (?y, ?x) \cap < http : //dbpedia.org/ontology/birthPlace > (?z, ?y) \cap < http : //dbpedia.org/ontology/bandMember > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/director > (?x, ?y) \cap < http : //dbpedia.org/ontology/restingPlace > (?y, ?z) \cap < http : //dbpedia.org/ontology/team > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/instrument > (?y, ?x) \cap < http : //dbpedia.org/ontology/editing > (?y, ?z) \cap < http : //dbpedia.org/ontology/stateOfOrigin > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/distributor > (?y, ?x) \cap < http : //dbpedia.org/ontology/campus > (?y, ?z) \cap < http : //dbpedia.org/ontology/hometown > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/writer > (?y, ?x) \cap < http : //dbpedia.org/ontology/influencedBy > (?z, ?y) \cap < http :$

$//dbpedia.org/ontology/genre > (?z, ?w)$

- $q(?w) :- < http : //dbpedia.org/ontology/birthPlace > (?x, ?y) \cap < http : //dbpedia.org/ontology/director > (?y, ?z) \cap < http : //dbpedia.org/ontology/starring > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/birthPlace > (?y, ?x) \cap < http : //dbpedia.org/ontology/award > (?y, ?z) \cap < http : //dbpedia.org/ontology/birthPlace > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/editing > (?y, ?x) \cap < http : //dbpedia.org/ontology/musicComposer > (?y, ?z) \cap < http : //dbpedia.org/ontology/isPartOf > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/isPartOf > (?x, ?y) \cap < http : //dbpedia.org/ontology/country > (?y, ?z) \cap < http : //dbpedia.org/ontology/type > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/almaMater > (?x, ?y) \cap < http : //dbpedia.org/ontology/instrument > (?z, ?y) \cap < http : //dbpedia.org/ontology/type > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/editing > (?x, ?y) \cap < http : //dbpedia.org/ontology/affiliation > (?z, ?y) \cap < http : //dbpedia.org/ontology/team > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/affiliation > (?y, ?x) \cap < http : //dbpedia.org/ontology/producer > (?z, ?y) \cap < http : //dbpedia.org/ontology/birthPlace > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/associatedBand > (?x, ?y) \cap < http : //dbpedia.org/ontology/influencedBy > (?y, ?z) \cap < http : //dbpedia.org/ontology/locationCountry > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/hometown > (?y, ?x) \cap < http : //dbpedia.org/ontology/type > (?z, ?y) \cap < http : //dbpedia.org/ontology/country > (?z, ?w)$

- **2i**:
  - $q(?w) :- < http : //dbpedia.org/ontology/occupation > (?w, ?x) \cap < http : //dbpedia.org/ontology/deathPlace > (?y, ?w)$
  - $q(?w) :- < http : //dbpedia.org/ontology/campus > (?x, ?w) \cap < http : //dbpedia.org/ontology/starring > (?y, ?w)$
  - $q(?w) :- < http : //dbpedia.org/ontology/genre > (?w, ?x) \cap < http : //dbpedia.org/ontology/ground > (?y, ?w)$

- $q(?w) :- <http://dbpedia.org/ontology/ethnicGroup>(?w, ?x) \cap <http://www.w3.org/2002/07/owl\#differentFrom>(?w, ?y)$
- $q(?w) :- <http://dbpedia.org/ontology/nationality>(?x, ?w) \cap <http://dbpedia.org/ontology/starring>(?w, ?y)$
- $q(?w) :- <http://dbpedia.org/ontology/parentCompany>(?x, ?w) \cap <http://dbpedia.org/ontology/spouse>(?w, ?y)$
- $q(?w) :- <http://dbpedia.org/ontology/writer>(?x, ?w) \cap <http://dbpedia.org/ontology/leaderName>(?y, ?w)$
- $q(?w) :- <http://dbpedia.org/ontology/director>(?w, ?x) \cap <http://dbpedia.org/ontology/isPartOf>(?y, ?w)$
- $q(?w) :- <http://dbpedia.org/ontology/starring>(?w, ?x) \cap <http://dbpedia.org/ontology/musicComposer>(?y, ?w)$
- $q(?w) :- <http://dbpedia.org/ontology/Agent>(?w) \cap <http://dbpedia.org/ontology/type>(?x, ?w)$
- $q(?w) :- <http://dbpedia.org/ontology/location>(?w, ?x) \cap <http://dbpedia.org/ontology/genre>(?y, ?w)$
- $q(?w) :- <http://www.wikidata.org/entity/Q486972>(?w) \cap <http://dbpedia.org/ontology/Place>(?w)$
- $q(?w) :- <http://dbpedia.org/ontology/Island>(?w) \cap <http://dbpedia.org/ontology/birthPlace>(?x, ?w)$
- $q(?w) :- <http://dbpedia.org/ontology/stylisticOrigin>(?w, ?x) \cap <http://dbpedia.org/ontology/country>(?y, ?w)$
- $q(?w) :- <http://dbpedia.org/ontology/language>(?w, ?x) \cap <http://dbpedia.org/ontology/governmentType>(?w, ?y)$
- $q(?w) :- <http://dbpedia.org/ontology/Island>(?w) \cap <http://dbpedia.org/ontology/Settlement>(?w)$
- $q(?w) :- <http://schema.org/Person>(?w) \cap <http://dbpedia.org/ontology/Settlement>(?w)$
- $q(?w) :- <http://www.wikidata.org/entity/Q486972>(?w) \cap <http://dbpedia.org/ontology/musicComposer>(?x, ?w)$
- $q(?w) :- <http://dbpedia.org/ontology/PopulatedPlace>(?w) \cap <http://dbpedia.org/ontology/Island>(?w)$
- $q(?w) :- <http://dbpedia.org/ontology/timeZone>(?w, ?x) \cap <http://dbpedia.org/ontology/almaMater>(?w, ?y)$
- $q(?w) :- <http://dbpedia.org/ontology/PopulatedPlace>(?w) \cap <http:$

$//dbpedia.org/ontology/distributor > (?w, ?x)$

- $q(?w) :- < http : //dbpedia.org/ontology/associatedBand > (?w, ?x) \cap < http : //dbpedia.org/ontology/genre > (?w, ?y)$
- $q(?w) :- < http : //dbpedia.org/ontology/place > (?x, ?w) \cap < http : //dbpedia.org/ontology/sisterCollege > (?y, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/cinematography > (?x, ?w) \cap < http : //dbpedia.org/ontology/musicFusionGenre > (?w, ?y)$
- $q(?w) :- < http : //www.wikidata.org/entity/Q215627 > (?w) \cap < http : //dbpedia.org/ontology/Island > (?w)$

- **3i**:
  - $q(?w) :- < http : //dbpedia.org/ontology/Place > (?w) \cap < http : //dbpedia.org/ontology/birthPlace > (?y, ?w) \cap < http : //dbpedia.org/ontology/starring > (?x, ?w)$
  - $q(?w) :- < http : //dbpedia.org/ontology/anthem > (?z, ?w) \cap < http : //dbpedia.org/ontology/birthPlace > (?y, ?w) \cap < http : //dbpedia.org/ontology/recordLabel > (?x, ?w)$
  - $q(?w) :- < http : //dbpedia.org/ontology/occupation > (?w, ?z) \cap < http : //dbpedia.org/ontology/largestCity > (?w, ?y) \cap < http : //dbpedia.org/ontology/occupation > (?x, ?w)$
  - $q(?w) :- < http : //dbpedia.org/ontology/Place > (?w) \cap < http : //dbpedia.org/ontology/executiveProducer > (?w, ?y) \cap < http : //dbpedia.org/ontology/musicComposer > (?x, ?w)$
  - $q(?w) :- < http : //dbpedia.org/ontology/instrument > (?w, ?z) \cap < http : //dbpedia.org/ontology/country > (?w, ?y) \cap < http : //dbpedia.org/ontology/city > (?w, ?x)$
  - $q(?w) :- < http : //dbpedia.org/ontology/Island > (?w) \cap < http : //dbpedia.org/ontology/Location > (?w) \cap < http : //dbpedia.org/ontology/Place > (?w)$
  - $q(?w) :- < http : //dbpedia.org/ontology/Island > (?w) \cap < http : //schema.org/Place > (?w) \cap < http : //dbpedia.org/ontology/Person > (?w)$
  - $q(?w) :- < http : //dbpedia.org/ontology/Person > (?w) \cap < http : //www.wikidata.org/entity/Q486972 > (?w) \cap < http : //dbpedia.org/ontology/associatedBand > (?w, ?x)$
  - $q(?w) :- < http : //xmlns.com/foaf/0.1/Person > (?w) \cap < http :$

$//dbpedia.org/ontology/Location > (?w) \cap < http://schema.org/Place > (?w)$

- $q(?w) \text{:-} < http://xmlns.com/foaf/0.1/Person > (?w) \cap < http://dbpedia.org/ontology/twinCountry > (?y, ?w) \cap < http://dbpedia.org/ontology/hometown > (?x, ?w)$
- $q(?w) \text{:-} < http://dbpedia.org/ontology/award > (?z, ?w) \cap < http://dbpedia.org/ontology/parentCompany > (?w, ?y) \cap < http://dbpedia.org/ontology/affiliation > (?x, ?w)$
- $q(?w) \text{:-} < http://www.wikidata.org/entity/Q5 > (?w) \cap < http://dbpedia.org/ontology/CelestialBody > (?w) \cap < http://dbpedia.org/ontology/Island > (?w)$
- $q(?w) \text{:-} < http://dbpedia.org/ontology/Island > (?w) \cap < http://dbpedia.org/ontology/SoccerClub > (?w) \cap < http://dbpedia.org/ontology/Island > (?w)$
- $q(?w) \text{:-} < http://dbpedia.org/ontology/locationCity > (?w, ?z) \cap < http://dbpedia.org/ontology/almaMater > (?w, ?y) \cap < http://dbpedia.org/ontology/populationPlace > (?x, ?w)$
- $q(?w) \text{:-} < http://dbpedia.org/ontology/birthPlace > (?w, ?z) \cap < http://dbpedia.org/ontology/hometown > (?w, ?y) \cap < http://dbpedia.org/ontology/birthPlace > (?x, ?w)$
- $q(?w) \text{:-} < http://dbpedia.org/ontology/distributingLabel > (?z, ?w) \cap < http://dbpedia.org/ontology/parentCompany > (?y, ?w) \cap < http://dbpedia.org/ontology/distributor > (?x, ?w)$
- $q(?w) \text{:-} < http://dbpedia.org/ontology/PopulatedPlace > (?w) \cap < http://dbpedia.org/ontology/Place > (?w) \cap < http://dbpedia.org/ontology/Governor > (?w)$
- $q(?w) \text{:-} < http://dbpedia.org/ontology/PopulatedPlace > (?w) \cap < http://dbpedia.org/ontology/populationPlace > (?y, ?w) \cap < http://dbpedia.org/ontology/languageFamily > (?w, ?x)$
- $q(?w) \text{:-} < http://www.wikidata.org/entity/Q486972 > (?w) \cap < http://dbpedia.org/ontology/genre > (?w, ?y) \cap < http://dbpedia.org/ontology/restingPlace > (?w, ?x)$
- $q(?w) \text{:-} < http://dbpedia.org/ontology/hometown > (?w, ?z) \cap < http://dbpedia.org/ontology/birthPlace > (?w, ?y) \cap < http://dbpedia.org/ontology/associatedMusicalArtist > (?w, ?x)$
- $q(?w) \text{:-} < http://dbpedia.org/ontology/owner > (?z, ?w) \cap <$

$http : //dbpedia.org/ontology/league > (?y, ?w) \cap < http : //dbpedia.org/ontology/birthPlace > (?w, ?x)$

- $q(?w) :- < http : //dbpedia.org/ontology/PopulatedPlace > (?w) \cap < http : //dbpedia.org/ontology/Place > (?w) \cap < http : //dbpedia.org/ontology/starring > (?w, ?x)$
- $q(?w) :- < http : //dbpedia.org/ontology/PopulatedPlace > (?w) \cap < http : //dbpedia.org/ontology/almaMater > (?w, ?y) \cap < http : //dbpedia.org/ontology/occupation > (?x, ?w)$
- $q(?w) :- < http : //schema.org/Person > (?w) \cap < http : //dbpedia.org/ontology/Settlement > (?w) \cap < http : //schema.org/Place > (?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/position > (?z, ?w) \cap < http : //dbpedia.org/ontology/associatedBand > (?y, ?w) \cap < http : //dbpedia.org/ontology/instrument > (?w, ?x)$

- **pi**:
  - $q(?w) :- < http : //dbpedia.org/ontology/genre > (?x, ?y) \cap < http : //dbpedia.org/ontology/state > (?w, ?y) \cap < http : //dbpedia.org/ontology/instrument > (?z, ?w)$
  - $q(?w) :- < http : //dbpedia.org/ontology/company > (?x, ?y) \cap < http : //dbpedia.org/ontology/sisterStation > (?w, ?y) \cap < http : //dbpedia.org/ontology/occupation > (?w, ?z)$
  - $q(?w) :- < http : //dbpedia.org/ontology/associatedMusicalArtist > (?x, ?y) \cap < http : //dbpedia.org/ontology/populationPlace > (?w, ?y) \cap < http : //dbpedia.org/ontology/populationPlace > (?w, ?z)$
  - $q(?w) :- < http : //dbpedia.org/ontology/location > (?x, ?y) \cap < http : //dbpedia.org/ontology/writer > (?w, ?y) \cap < http : //dbpedia.org/ontology/associatedMusicalArtist > (?w, ?z)$
  - $q(?w) :- < http : //dbpedia.org/ontology/director > (?x, ?y) \cap < http : //dbpedia.org/ontology/capital > (?w, ?y) \cap < http : //dbpedia.org/ontology/starring > (?w, ?z)$
  - $q(?w) :- < http : //dbpedia.org/ontology/cinematography > (?x, ?y) \cap < http : //dbpedia.org/ontology/company > (?w, ?y) \cap < http : //dbpedia.org/ontology/knownFor > (?w, ?z)$
  - $q(?w) :- < http : //dbpedia.org/ontology/headquarter > (?x, ?y) \cap < http : //dbpedia.org/ontology/sourceConfluencePlace > (?w, ?y) \cap < http :$

$//dbpedia.org/ontology/director > (?w, ?z)$

- $q(?w) :- < http : //dbpedia.org/ontology/genre > (?x, ?y) \cap < http : //dbpedia.org/ontology/recordLabel > (?w, ?y) \cap < http : //dbpedia.org/ontology/starring > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/director > (?x, ?y) \cap < http : //dbpedia.org/ontology/instrument > (?w, ?y) \cap < http : //dbpedia.org/ontology/stylisticOrigin > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/language > (?x, ?y) \cap < http : //dbpedia.org/ontology/sisterStation > (?y, ?w) \cap < http : //dbpedia.org/ontology/recordLabel > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/successor > (?x, ?y) \cap < http : //dbpedia.org/ontology/occupation > (?y, ?w) \cap < http : //dbpedia.org/ontology/birthPlace > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/timeZone > (?x, ?y) \cap < http : //dbpedia.org/ontology/hometown > (?y, ?w) \cap < http : //dbpedia.org/ontology/locationCity > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/genre > (?x, ?y) \cap < http : //dbpedia.org/ontology/nationality > (?y, ?w) \cap < http : //dbpedia.org/ontology/parentCompany > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/associatedBand > (?x, ?y) \cap < http : //dbpedia.org/ontology/developer > (?y, ?w) \cap < http : //dbpedia.org/ontology/type > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/knownFor > (?x, ?y) \cap < http : //dbpedia.org/ontology/ideology > (?y, ?w) \cap < http : //dbpedia.org/ontology/starring > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/type > (?x, ?y) \cap < http : //dbpedia.org/ontology/hometown > (?w, ?y) \cap < http : //dbpedia.org/ontology/type > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/capital > (?x, ?y) \cap < http : //dbpedia.org/ontology/cinematography > (?w, ?y) \cap < http : //dbpedia.org/ontology/producer > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/associatedBand > (?x, ?y) \cap < http : //dbpedia.org/ontology/ground > (?y, ?w) \cap < http : //dbpedia.org/ontology/influencedBy > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/sisterCollege > (?x, ?y) \cap < http : //dbpedia.org/ontology/formerBandMember > (?w, ?y) \cap < http :$

$//dbpedia.org/ontology/associatedBand > (?z, ?w)$

- $q(?w) :- < http : //dbpedia.org/ontology/occupation > (?x, ?y) \cap < http : //dbpedia.org/ontology/birthPlace > (?w, ?y) \cap < http : //dbpedia.org/ontology/birthPlace > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/recordLabel > (?x, ?y) \cap < http : //dbpedia.org/ontology/stylisticOrigin > (?w, ?y) \cap < http : //dbpedia.org/ontology/genre > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/associatedBand > (?x, ?y) \cap < http : //dbpedia.org/ontology/associatedMusicalArtist > (?y, ?w) \cap < http : //dbpedia.org/ontology/deathPlace > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/leaderName > (?x, ?y) \cap < http : //dbpedia.org/ontology/writer > (?y, ?w) \cap < http : //dbpedia.org/ontology/starring > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/cinematography > (?x, ?y) \cap < http : //dbpedia.org/ontology/occupation > (?w, ?y) \cap < http : //dbpedia.org/ontology/battle > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/populationPlace > (?x, ?y) \cap < http : //dbpedia.org/ontology/subsequentWork > (?w, ?y) \cap < http : //dbpedia.org/ontology/leaderName > (?z, ?w)$

- **ip**:
  - $q(?w) :- < http : //schema.org/Person > (?x) \cap < http : //dbpedia.org/ontology/Agent > (?x) \cap < http : //dbpedia.org/ontology/location > (?w, ?x)$
  - $q(?w) :- < http : //dbpedia.org/ontology/commandStructure > (?z, ?x) \cap < http : //dbpedia.org/ontology/locationCountry > (?z, ?y) \cap < http : //dbpedia.org/ontology/cinematography > (?z, ?w)$
  - $q(?w) :- < http : //dbpedia.org/ontology/birthPlace > (?z, ?x) \cap < http : //dbpedia.org/ontology/associatedMusicalArtist > (?y, ?z) \cap < http : //dbpedia.org/ontology/country > (?z, ?w)$
  - $q(?w) :- < http : //dbpedia.org/ontology/Island > (?x) \cap < http : //schema.org/Person > (?x) \cap < http : //dbpedia.org/ontology/associatedMusicalArtist > (?x, ?w)$
  - $q(?w) :- < http : //dbpedia.org/ontology/creator > (?z, ?x) \cap < http : //dbpedia.org/ontology/country > (?z, ?y) \cap < http : //dbpedia.org/ontology/location > (?z, ?w)$

- $q(?w) :- <http://xmlns.com/foaf/0.1/Person>(?z) \cap <http://dbpedia.org/ontology/associatedBand>(?y, ?z) \cap <http://dbpedia.org/ontology/musicComposer>(?z, ?w)$
- $q(?w) :- <http://www.wikidata.org/entity/Q47521>(?z) \cap <http://dbpedia.org/ontology/associatedMusicalArtist>(?z, ?y) \cap <http://dbpedia.org/ontology/starring>(?z, ?w)$
- $q(?w) :- <http://dbpedia.org/ontology/instrument>(?z, ?x) \cap <http://dbpedia.org/ontology/associatedMusicalArtist>(?z, ?y) \cap <http://dbpedia.org/ontology/director>(?z, ?w)$
- $q(?w) :- <http://schema.org/Person>(?z) \cap <http://dbpedia.org/ontology/starring>(?z, ?y) \cap <http://dbpedia.org/ontology/genre>(?w, ?z)$
- $q(?w) :- <http://www.wikidata.org/entity/Q5>(?z) \cap <http://dbpedia.org/ontology/genre>(?y, ?z) \cap <http://dbpedia.org/ontology/associatedBand>(?w, ?z)$
- $q(?w) :- <http://dbpedia.org/ontology/Architect>(?z) \cap <http://dbpedia.org/ontology/influencedBy>(?y, ?z) \cap <http://dbpedia.org/ontology/education>(?z, ?w)$
- $q(?w) :- <http://www.wikidata.org/entity/Q486972>(?z) \cap <http://dbpedia.org/ontology/producer>(?z, ?y) \cap <http://dbpedia.org/ontology/musicComposer>(?z, ?w)$
- $q(?w) :- <http://dbpedia.org/ontology/timeZone>(?z, ?x) \cap <http://dbpedia.org/ontology/state>(?z, ?y) \cap <http://dbpedia.org/ontology/commander>(?w, ?z)$
- $q(?w) :- <http://dbpedia.org/ontology/starring>(?z, ?x) \cap <http://dbpedia.org/ontology/recordLabel>(?z, ?y) \cap <http://dbpedia.org/ontology/locationCountry>(?w, ?z)$
- $q(?w) :- <http://dbpedia.org/ontology/country>(?x, ?z) \cap <http://dbpedia.org/ontology/writer>(?y, ?z) \cap <http://dbpedia.org/ontology/residence>(?z, ?w)$
- $q(?w) :- <http://dbpedia.org/ontology/Person>(?z) \cap <http://dbpedia.org/ontology/distributor>(?z, ?y) \cap <http://dbpedia.org/ontology/distributor>(?w, ?z)$
- $q(?w) :- <http://dbpedia.org/ontology/location>(?x, ?z) \cap <http://dbpedia.org/ontology/starring>(?z, ?y) \cap <http://dbpedia.org/ontology/mainInterest>(?z, ?w)$

- $q(?w)$:- $< http : //dbpedia.org/ontology/recordLabel > (?x, ?z) \cap < http : //dbpedia.org/ontology/country > (?y, ?z) \cap < http : //dbpedia.org/ontology/birthPlace > (?z, ?w)$
- $q(?w)$:- $< http : //dbpedia.org/ontology/campus > (?z, ?x) \cap < http : //dbpedia.org/ontology/formerTeam > (?z, ?y) \cap < http : //dbpedia.org/ontology/governmentType > (?z, ?w)$
- $q(?w)$:- $< http : //dbpedia.org/ontology/starring > (?x, ?z) \cap < http : //dbpedia.org/ontology/associatedMusicalArtist > (?z, ?y) \cap < http : //dbpedia.org/ontology/associatedMusicalArtist > (?z, ?w)$
- $q(?w)$:- $< http : //dbpedia.org/ontology/writer > (?x, ?z) \cap < http : //dbpedia.org/ontology/region > (?z, ?y) \cap < http : //dbpedia.org/ontology/influencedBy > (?z, ?w)$
- $q(?w)$:- $< http : //dbpedia.org/ontology/industry > (?z, ?x) \cap < http : //dbpedia.org/ontology/musicComposer > (?z, ?y) \cap < http : //dbpedia.org/ontology/starring > (?z, ?w)$
- $q(?w)$:- $< http : //dbpedia.org/ontology/region > (?x, ?z) \cap < http : //dbpedia.org/ontology/distributor > (?y, ?z) \cap < http : //dbpedia.org/ontology/foundationPlace > (?z, ?w)$
- $q(?w)$:- $< http : //schema.org/Place > (?x) \cap < http : //dbpedia.org/ontology/Person > (?x) \cap < http : //dbpedia.org/ontology/affiliation > (?x, ?w)$
- $q(?w)$:- $< http : //www.wikidata.org/entity/Q486972 > (?z) \cap < http : //dbpedia.org/ontology/musicComposer > (?z, ?y) \cap < http : //dbpedia.org/ontology/daylightSavingTimeZone > (?w, ?z)$

- **up**:
  - $q(?w)$:- $< http : //dbpedia.org/ontology/leaderParty > (?z, ?x) \cup < http : //dbpedia.org/ontology/influencedBy > (?z, ?y) \cap < http : //dbpedia.org/ontology/governmentType > (?w, ?z)$
  - $q(?w)$:- $< http : //dbpedia.org/ontology/Person > (?x) \cup < http : //dbpedia.org/ontology/Place > (?x) \cap < http : //dbpedia.org/ontology/successor > (?w, ?x)$
  - $q(?w)$:- $< http : //www.ontologydesignpatterns.org/ont/dul/DUL.owl\#Agent > (?x) \cup < http : //www.ontologydesignpatterns.org/ont/dul/DUL.owl\#NaturalPerson > (?x) \cap < http : //dbpedia.org/ontology/state > (?w, ?x)$
  - $q(?w)$:- $< http : //dbpedia.org/ontology/affiliation > (?x, ?z) \cup <$

$http : //dbpedia.org/ontology/instrument > (?z, ?y) \cap < http : //dbpedia.org/ontology/producer > (?w, ?z)$

- $q(?w) :- < http : //dbpedia.org/ontology/stylisticOrigin > (?z, ?x) \cup < http : //dbpedia.org/ontology/almaMater > (?y, ?z) \cap < http : //dbpedia.org/ontology/stylisticOrigin > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/Settlement > (?z) \cup < http : //dbpedia.org/ontology/stylisticOrigin > (?z, ?y) \cap < http : //dbpedia.org/ontology/capital > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/keyPerson > (?x, ?z) \cup < http : //dbpedia.org/ontology/occupation > (?y, ?z) \cap < http : //dbpedia.org/ontology/affiliation > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/Island > (?z) \cup < http : //dbpedia.org/ontology/deathPlace > (?z, ?y) \cap < http : //dbpedia.org/ontology/type > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/party > (?z, ?x) \cup < http : //dbpedia.org/ontology/genre > (?y, ?z) \cap < http : //www.w3.org/2002/07/owl\#differentFrom > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/Species > (?z) \cup < http : //dbpedia.org/ontology/producer > (?z, ?y) \cap < http : //dbpedia.org/ontology/narrator > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/Island > (?x) \cup < http : //dbpedia.org/ontology/Island > (?x) \cap < http : //dbpedia.org/ontology/deathPlace > (?w, ?x)$
- $q(?w) :- < http : //dbpedia.org/ontology/starring > (?x, ?z) \cup < http : //dbpedia.org/ontology/owner > (?y, ?z) \cap < http : //dbpedia.org/ontology/country > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/Island > (?z) \cup < http : //dbpedia.org/ontology/country > (?z, ?y) \cap < http : //dbpedia.org/ontology/related > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/starring > (?z, ?x) \cup < http : //dbpedia.org/ontology/musicFusionGenre > (?y, ?z) \cap < http : //dbpedia.org/ontology/almaMater > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/countySeat > (?x, ?z) \cup < http : //dbpedia.org/ontology/formerBandMember > (?y, ?z) \cap < http : //dbpedia.org/ontology/athletics > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/Island > (?x) \cup <$

$http : //dbpedia.org/ontology/Island > (?x) \cap < http : //dbpedia.org/ontology/genre > (?x, ?w)$

- $q(?w) :- < http : //dbpedia.org/ontology/Place > (?z) \cup < http : //dbpedia.org/ontology/locationCountry > (?y, ?z) \cap < http : //dbpedia.org/ontology/instrument > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/genre > (?x, ?z) \cup < http : //dbpedia.org/ontology/type > (?z, ?y) \cap < http : //dbpedia.org/ontology/cinematography > (?w, ?z)$
- $q(?w) :- < http : //www.ontologydesignpatterns.org/ont/dul/DUL.owl\#Agent > (?z) \cup < http : //dbpedia.org/ontology/spouse > (?z, ?y) \cap < http : //dbpedia.org/ontology/genre > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/Settlement > (?z) \cup < http : //dbpedia.org/ontology/starring > (?z, ?y) \cap < http : //dbpedia.org/ontology/birthPlace > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/Island > (?x) \cup < http : //www.wikidata.org/entity/Q486972 > (?x) \cap < http : //dbpedia.org/ontology/starring > (?w, ?x)$
- $q(?w) :- < http : //dbpedia.org/ontology/starring > (?x, ?z) \cup < http : //dbpedia.org/ontology/genre > (?y, ?z) \cap < http : //dbpedia.org/ontology/countySeat > (?w, ?z)$
- $q(?w) :- < http : //dbpedia.org/ontology/associatedMusicalArtist > (?x, ?z) \cup < http : //dbpedia.org/ontology/associatedBand > (?z, ?y) \cap < http : //dbpedia.org/ontology/country > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/PopulatedPlace > (?z) \cup < http : //dbpedia.org/ontology/hometown > (?y, ?z) \cap < http : //dbpedia.org/ontology/league > (?z, ?w)$
- $q(?w) :- < http : //dbpedia.org/ontology/countySeat > (?z, ?x) \cup < http : //dbpedia.org/ontology/populationPlace > (?z, ?y) \cap < http : //dbpedia.org/ontology/country > (?w, ?z)$

- **2u**:
  - $q(?w) :- < http : //dbpedia.org/ontology/distributor > (?x, ?w) \cup < http : //dbpedia.org/ontology/cinematography > (?w, ?y)$
  - $q(?w) :- < http : //dbpedia.org/ontology/birthPlace > (?x, ?w) \cup < http : //dbpedia.org/ontology/birthPlace > (?w, ?y)$
  - $q(?w) :- < http : //dbpedia.org/ontology/timeZone > (?w, ?x) \cup < http :$

$//dbpedia.org/ontology/writer > (?w, ?y)$

- $q(?w) :- < http://www.ontologydesignpatterns.org/ont/dul/DUL.owl\#Agent > (?w) \cup < http://dbpedia.org/ontology/starring > (?w, ?x)$
- $q(?w) :- < http://dbpedia.org/ontology/Island > (?w) \cup < http://dbpedia.org/ontology/Planet > (?w)$
- $q(?w) :- < http://dbpedia.org/ontology/Economist > (?w) \cup < http://dbpedia.org/ontology/birthPlace > (?x, ?w)$
- $q(?w) :- < http://dbpedia.org/ontology/director > (?w, ?x) \cup < http://dbpedia.org/ontology/language > (?w, ?y)$
- $q(?w) :- < http://dbpedia.org/ontology/country > (?x, ?w) \cup < http://dbpedia.org/ontology/birthPlace > (?w, ?y)$
- $q(?w) :- < http://dbpedia.org/ontology/Location > (?w) \cup < http://dbpedia.org/ontology/birthPlace > (?x, ?w)$
- $q(?w) :- < http://dbpedia.org/ontology/Place > (?w) \cup < http://dbpedia.org/ontology/campus > (?w, ?x)$
- $q(?w) :- < http://www.ontologydesignpatterns.org/ont/dul/DUL.owl\#Agent > (?w) \cup < http://dbpedia.org/ontology/Place > (?w)$
- $q(?w) :- < http://dbpedia.org/ontology/Location > (?w) \cup < http://dbpedia.org/ontology/Place > (?w)$
- $q(?w) :- < http://dbpedia.org/ontology/timeZone > (?w, ?x) \cup < http://dbpedia.org/ontology/distributor > (?y, ?w)$
- $q(?w) :- < http://dbpedia.org/ontology/Location > (?w) \cup < http://dbpedia.org/ontology/Settlement > (?w)$
- $q(?w) :- < http://dbpedia.org/ontology/associatedMusicalArtist > (?w, ?x) \cup < http://dbpedia.org/ontology/genre > (?y, ?w)$
- $q(?w) :- < http://dbpedia.org/ontology/timeZone > (?w, ?x) \cup < http://dbpedia.org/ontology/almaMater > (?w, ?y)$
- $q(?w) :- < http://dbpedia.org/ontology/territory > (?w, ?x) \cup < http://dbpedia.org/ontology/occupation > (?w, ?y)$
- $q(?w) :- < http://dbpedia.org/ontology/genre > (?x, ?w) \cup < http://dbpedia.org/ontology/influencedBy > (?y, ?w)$
- $q(?w) :- < http://dbpedia.org/ontology/starring > (?w, ?x) \cup < http://dbpedia.org/ontology/instrument > (?y, ?w)$
- $q(?w) :- < http://dbpedia.org/ontology/isPartOf > (?w, ?x) \cup < http://dbpedia.org/ontology/recordLabel > (?y, ?w)$

- $q(?w) :- <http://dbpedia.org/ontology/basedOn>(?w, ?x) \cup <http://dbpedia.org/ontology/cinematography>(?y, ?w)$
- $q(?w) :- <http://dbpedia.org/ontology/Island>(?w) \cup <http://dbpedia.org/ontology/associatedMusicalArtist>(?w, ?x)$
- $q(?w) :- <http://dbpedia.org/ontology/campus>(?w, ?x) \cup <http://dbpedia.org/ontology/musicComposer>(?w, ?y)$
- $q(?w) :- <http://dbpedia.org/ontology/company>(?x, ?w) \cup <http://dbpedia.org/ontology/associatedBand>(?w, ?y)$
- $q(?w) :- <http://www.wikidata.org/entity/Q215627>(?w) \cup <http://www.wikidata.org/entity/Q486972>(?w)$

E.2 Family Dataset

- **1p**:
  - $q(?w) :- <https://www.wikidata.org/prop/P40>(?w, ?x)$
  - $q(?w) :- <https://www.wikidata.org/wiki/Q10861465>(?w)$
  - $q(?w) :- <https://www.wikidata.org/wiki/Q7569>(?w)$
  - $q(?w) :- <https://www.wikidata.org/prop/P26>(?x, ?w)$
  - $q(?w) :- <https://www.wikidata.org/prop/P3373>(?x, ?w)$
  - $q(?w) :- <https://www.wikidata.org/wiki/Q177232>(?w)$
  - $q(?w) :- <https://www.wikidata.org/prop/P25>(?x, ?w)$
  - $q(?w) :- <https://www.wikidata.org/wiki/Q7566>(?w)$
  - $q(?w) :- <https://www.wikidata.org/wiki/Q308194>(?w)$
  - $q(?w) :- <https://www.wikidata.org/prop/P26>(?x, ?w)$
  - $q(?w) :- <https://www.wikidata.org/wiki/Q177232>(?w)$
  - $q(?w) :- <https://www.wikidata.org/wiki/Q10861465>(?w)$
  - $q(?w) :- <https://www.wikidata.org/wiki/Q1196129>(?w)$
  - $q(?w) :- <https://www.wikidata.org/prop/P26>(?w, ?x)$
  - $q(?w) :- <https://www.wikidata.org/wiki/Q7566>(?w)$
  - $q(?w) :- <https://www.wikidata.org/prop/P26>(?x, ?w)$
  - $q(?w) :- <https://www.wikidata.org/wiki/Q31184>(?w)$
  - $q(?w) :- <https://www.wikidata.org/wiki/Q7560>(?w)$
  - $q(?w) :- <https://www.wikidata.org/wiki/Q595094>(?w)$

- $q(?w) :- <https://www.wikidata.org/prop/P22>(?w, ?x)$
- $q(?w) :- <https://www.wikidata.org/wiki/Q308194>(?w)$
- $q(?w) :- <https://www.wikidata.org/wiki/Q595094>(?w)$
- $q(?w) :- <https://www.wikidata.org/prop/P3373>(?w, ?x)$
- $q(?w) :- <https://www.wikidata.org/prop/P40>(?w, ?x)$
- $q(?w) :- <https://www.wikidata.org/prop/P3373>(?x, ?w)$

- **2p**:
  - $q(?w) :- <https://www.wikidata.org/prop/P22>(?x, ?y) \cap <https://www.wikidata.org/prop/P3373>(?w, ?y)$
  - $q(?w) :- <https://www.wikidata.org/prop/P26>(?x, ?y) \cap <https://www.wikidata.org/prop/P26>(?y, ?w)$
  - $q(?w) :- <https://www.wikidata.org/prop/P40>(?x, ?y) \cap <https://www.wikidata.org/prop/P3373>(?y, ?w)$
  - $q(?w) :- <https://www.wikidata.org/prop/P26>(?y, ?x) \cap <https://www.wikidata.org/prop/P3373>(?y, ?w)$
  - $q(?w) :- <https://www.wikidata.org/prop/P40>(?y, ?x) \cap <https://www.wikidata.org/prop/P26>(?y, ?w)$
  - $q(?w) :- <https://www.wikidata.org/prop/P25>(?x, ?y) \cap <https://www.wikidata.org/prop/P3373>(?w, ?y)$
  - $q(?w) :- <https://www.wikidata.org/prop/P3373>(?x, ?y) \cap <https://www.wikidata.org/prop/P26>(?w, ?y)$
  - $q(?w) :- <https://www.wikidata.org/prop/P22>(?y, ?x) \cap <https://www.wikidata.org/prop/P3373>(?y, ?w)$
  - $q(?w) :- <https://www.wikidata.org/prop/P3373>(?x, ?y) \cap <https://www.wikidata.org/prop/P3373>(?w, ?y)$
  - $q(?w) :- <https://www.wikidata.org/prop/P40>(?y, ?x) \cap <https://www.wikidata.org/prop/P40>(?y, ?w)$
  - $q(?w) :- <https://www.wikidata.org/prop/P3373>(?x, ?y) \cap <https://www.wikidata.org/prop/P40>(?w, ?y)$
  - $q(?w) :- <https://www.wikidata.org/prop/P3373>(?y, ?x) \cap <https://www.wikidata.org/prop/P40>(?y, ?w)$
  - $q(?w) :- <https://www.wikidata.org/prop/P22>(?y, ?x) \cap <https://www.wikidata.org/prop/P3373>(?y, ?w)$
  - $q(?w) :- <https://www.wikidata.org/prop/P26>(?x, ?y) \cap <https://www.wikidata.org/prop/P26>(?y, ?w)$

- $q(?w) :- <https://www.wikidata.org/prop/P3373>(?y, ?x) \cap <https://www.wikidata.org/prop/P1038>(?w, ?y)$
- $q(?w) :- <https://www.wikidata.org/prop/P25>(?x, ?y) \cap <https://www.wikidata.org/prop/P26>(?w, ?y)$
- $q(?w) :- <https://www.wikidata.org/prop/P3373>(?y, ?x) \cap <https://www.wikidata.org/prop/P22>(?y, ?w)$
- $q(?w) :- <https://www.wikidata.org/prop/P3373>(?y, ?x) \cap <https://www.wikidata.org/prop/P22>(?y, ?w)$
- $q(?w) :- <https://www.wikidata.org/prop/P3373>(?y, ?x) \cap <https://www.wikidata.org/prop/P40>(?y, ?w)$
- $q(?w) :- <https://www.wikidata.org/prop/P40>(?x, ?y) \cap <https://www.wikidata.org/prop/P3373>(?y, ?w)$
- $q(?w) :- <https://www.wikidata.org/prop/P26>(?x, ?y) \cap <https://www.wikidata.org/prop/P40>(?w, ?y)$
- $q(?w) :- <https://www.wikidata.org/prop/P3373>(?y, ?x) \cap <https://www.wikidata.org/prop/P3373>(?w, ?y)$
- $q(?w) :- <https://www.wikidata.org/prop/P22>(?x, ?y) \cap <https://www.wikidata.org/prop/P40>(?w, ?y)$
- $q(?w) :- <https://www.wikidata.org/prop/P3373>(?x, ?y) \cap <https://www.wikidata.org/prop/P22>(?y, ?w)$
- $q(?w) :- <https://www.wikidata.org/prop/P3373>(?y, ?x) \cap <https://www.wikidata.org/prop/P22>(?w, ?y)$

- **3p**:
  - $q(?w) :- <https://www.wikidata.org/prop/P3373>(?x, ?y) \cap <https://www.wikidata.org/prop/P22>(?y, ?z) \cap <https://www.wikidata.org/prop/P40>(?w, ?z)$
  - $q(?w) :- <https://www.wikidata.org/prop/P25>(?x, ?y) \cap <https://www.wikidata.org/prop/P26>(?y, ?z) \cap <https://www.wikidata.org/prop/P40>(?w, ?z)$
  - $q(?w) :- <https://www.wikidata.org/prop/P22>(?x, ?y) \cap <https://www.wikidata.org/prop/P40>(?y, ?z) \cap <https://www.wikidata.org/prop/P40>(?z, ?w)$
  - $q(?w) :- <https://www.wikidata.org/prop/P3373>(?x, ?y) \cap <https://www.wikidata.org/prop/P40>(?y, ?z) \cap <https://www.wikidata.org/prop/P22>(?z, ?w)$

- $q(?w) :- <https://www.wikidata.org/prop/P3373>(?x, ?y) \cap <https://www.wikidata.org/prop/P22>(?z, ?y) \cap <https://www.wikidata.org/prop/P3373>(?z, ?w)$
- $q(?w) :- <https://www.wikidata.org/prop/P40>(?y, ?x) \cap <https://www.wikidata.org/prop/P3373>(?z, ?y) \cap <https://www.wikidata.org/prop/P26>(?z, ?w)$
- $q(?w) :- <https://www.wikidata.org/prop/P22>(?x, ?y) \cap <https://www.wikidata.org/prop/P40>(?z, ?y) \cap <https://www.wikidata.org/prop/P40>(?w, ?z)$
- $q(?w) :- <https://www.wikidata.org/prop/P3373>(?x, ?y) \cap <https://www.wikidata.org/prop/P22>(?y, ?z) \cap <https://www.wikidata.org/prop/P40>(?z, ?w)$
- $q(?w) :- <https://www.wikidata.org/prop/P3373>(?x, ?y) \cap <https://www.wikidata.org/prop/P22>(?y, ?z) \cap <https://www.wikidata.org/prop/P3373>(?w, ?z)$
- $q(?w) :- <https://www.wikidata.org/prop/P25>(?x, ?y) \cap <https://www.wikidata.org/prop/P22>(?y, ?z) \cap <https://www.wikidata.org/prop/P40>(?z, ?w)$
- $q(?w) :- <https://www.wikidata.org/prop/P40>(?x, ?y) \cap <https://www.wikidata.org/prop/P3373>(?z, ?y) \cap <https://www.wikidata.org/prop/P26>(?z, ?w)$
- $q(?w) :- <https://www.wikidata.org/prop/P40>(?x, ?y) \cap <https://www.wikidata.org/prop/P3373>(?y, ?z) \cap <https://www.wikidata.org/prop/P3373>(?w, ?z)$
- $q(?w) :- <https://www.wikidata.org/prop/P26>(?x, ?y) \cap <https://www.wikidata.org/prop/P40>(?y, ?z) \cap <https://www.wikidata.org/prop/P40>(?z, ?w)$
- $q(?w) :- <https://www.wikidata.org/prop/P22>(?y, ?x) \cap <https://www.wikidata.org/prop/P3373>(?z, ?y) \cap <https://www.wikidata.org/prop/P3373>(?z, ?w)$
- $q(?w) :- <https://www.wikidata.org/prop/P40>(?y, ?x) \cap <https://www.wikidata.org/prop/P40>(?y, ?z) \cap <https://www.wikidata.org/prop/P3373>(?w, ?z)$
- $q(?w) :- <https://www.wikidata.org/prop/P1038>(?y, ?x) \cap <https://www.wikidata.org/prop/P22>(?y, ?z) \cap <https://www.wikidata.org/prop/P40>(?w, ?z)$

- $q(?w) :- < https://www.wikidata.org/prop/P22 > (?y, ?x) \cap < https://www.wikidata.org/prop/P3373 > (?y, ?z) \cap < https://www.wikidata.org/prop/P22 > (?w, ?z)$
- $q(?w) :- < https://www.wikidata.org/prop/P3373 > (?x, ?y) \cap < https://www.wikidata.org/prop/P40 > (?y, ?z) \cap < https://www.wikidata.org/prop/P3373 > (?z, ?w)$
- $q(?w) :- < https://www.wikidata.org/prop/P26 > (?x, ?y) \cap < https://www.wikidata.org/prop/P3373 > (?z, ?y) \cap < https://www.wikidata.org/prop/P40 > (?w, ?z)$
- $q(?w) :- < https://www.wikidata.org/prop/P40 > (?y, ?x) \cap < https://www.wikidata.org/prop/P40 > (?z, ?y) \cap < https://www.wikidata.org/prop/P3373 > (?z, ?w)$
- $q(?w) :- < https://www.wikidata.org/prop/P40 > (?x, ?y) \cap < https://www.wikidata.org/prop/P3373 > (?y, ?z) \cap < https://www.wikidata.org/prop/P3373 > (?w, ?z)$
- $q(?w) :- < https://www.wikidata.org/prop/P25 > (?y, ?x) \cap < https://www.wikidata.org/prop/P40 > (?y, ?z) \cap < https://www.wikidata.org/prop/P26 > (?w, ?z)$
- $q(?w) :- < https://www.wikidata.org/prop/P1038 > (?x, ?y) \cap < https://www.wikidata.org/prop/P40 > (?z, ?y) \cap < https://www.wikidata.org/prop/P3373 > (?w, ?z)$
- $q(?w) :- < https://www.wikidata.org/prop/P1038 > (?x, ?y) \cap < https://www.wikidata.org/prop/P3373 > (?z, ?y) \cap < https://www.wikidata.org/prop/P3373 > (?z, ?w)$
- $q(?w) :- < https://www.wikidata.org/prop/P22 > (?x, ?y) \cap < https://www.wikidata.org/prop/P3373 > (?z, ?y) \cap < https://www.wikidata.org/prop/P3373 > (?z, ?w)$

- **2i**:
  - $q(?w) :- < https://www.wikidata.org/wiki/Q7560 > (?w) \cap < https://www.wikidata.org/prop/P40 > (?w, ?x)$
  - $q(?w) :- < https://www.wikidata.org/wiki/Q10861465 > (?w) \cap < https://www.wikidata.org/wiki/Q7566 > (?w)$
  - $q(?w) :- < https://www.wikidata.org/wiki/Q31184 > (?w) \cap < https://www.wikidata.org/wiki/Q1196129 > (?w)$
  - $q(?w) :- < https://www.wikidata.org/prop/P26 > (?x, ?w) \cap < https:$

$//www.wikidata.org/prop/P3373 > (?w, ?y)$

- $q(?w) :- < https://www.wikidata.org/prop/P3373 > (?x, ?w) \cap < https://www.wikidata.org/prop/P26 > (?y, ?w)$
- $q(?w) :- < https://www.wikidata.org/wiki/Q5 > (?w) \cap < https://www.wikidata.org/prop/P22 > (?w, ?x)$
- $q(?w) :- < https://www.wikidata.org/wiki/Q1196129 > (?w) \cap < https://www.wikidata.org/wiki/Q595094 > (?w)$
- $q(?w) :- < https://www.wikidata.org/wiki/Q308194 > (?w) \cap < https://www.wikidata.org/prop/P26 > (?w, ?x)$
- $q(?w) :- < https://www.wikidata.org/wiki/Q7569 > (?w) \cap < https://www.wikidata.org/wiki/Q308194 > (?w)$
- $q(?w) :- < https://www.wikidata.org/prop/P22 > (?x, ?w) \cap < https://www.wikidata.org/prop/P26 > (?y, ?w)$
- $q(?w) :- < https://www.wikidata.org/prop/P3373 > (?w, ?x) \cap < https://www.wikidata.org/prop/P3373 > (?y, ?w)$
- $q(?w) :- < https://www.wikidata.org/prop/P3373 > (?w, ?x) \cap < https://www.wikidata.org/prop/P22 > (?y, ?w)$
- $q(?w) :- < https://www.wikidata.org/prop/P3373 > (?x, ?w) \cap < https://www.wikidata.org/prop/P3373 > (?w, ?y)$
- $q(?w) :- < https://www.wikidata.org/prop/P3373 > (?w, ?x) \cap < https://www.wikidata.org/prop/P3373 > (?w, ?y)$
- $q(?w) :- < https://www.wikidata.org/prop/P26 > (?w, ?x) \cap < https://www.wikidata.org/prop/P3373 > (?y, ?w)$
- $q(?w) :- < https://www.wikidata.org/prop/P40 > (?w, ?x) \cap < https://www.wikidata.org/prop/P22 > (?y, ?w)$
- $q(?w) :- < https://www.wikidata.org/prop/P40 > (?x, ?w) \cap < https://www.wikidata.org/prop/P40 > (?y, ?w)$
- $q(?w) :- < https://www.wikidata.org/wiki/Q595094 > (?w) \cap < https://www.wikidata.org/wiki/Q1196129 > (?w)$
- $q(?w) :- < https://www.wikidata.org/prop/P22 > (?w, ?x) \cap < https://www.wikidata.org/prop/P3373 > (?y, ?w)$
- $q(?w) :- < https://www.wikidata.org/wiki/Q595094 > (?w) \cap < https://www.wikidata.org/prop/P26 > (?x, ?w)$
- $q(?w) :- < https://www.wikidata.org/prop/P3373 > (?x, ?w) \cap < https://www.wikidata.org/prop/P3373 > (?w, ?y)$

- $q(?w) :- <https://www.wikidata.org/wiki/Q308194>(?w) \cap <https://www.wikidata.org/wiki/Q5>(?w)$
- $q(?w) :- <https://www.wikidata.org/wiki/Q177232>(?w) \cap <https://www.wikidata.org/prop/P26>(?x, ?w)$
- $q(?w) :- <https://www.wikidata.org/prop/P3373>(?w, ?x) \cap <https://www.wikidata.org/prop/P3373>(?y, ?w)$
- $q(?w) :- <https://www.wikidata.org/wiki/Q7566>(?w) \cap <https://www.wikidata.org/prop/P40>(?x, ?w)$

- **3i**:
  - $q(?w) :- <https://www.wikidata.org/wiki/Q7566>(?w) \cap <https://www.wikidata.org/wiki/Q7566>(?w) \cap <https://www.wikidata.org/wiki/Q7560>(?w)$
  - $q(?w) :- <https://www.wikidata.org/prop/P3373>(?w, ?z) \cap <https://www.wikidata.org/prop/P40>(?y, ?w) \cap <https://www.wikidata.org/prop/P1038>(?x, ?w)$
  - $q(?w) :- <https://www.wikidata.org/wiki/Q5>(?w) \cap <https://www.wikidata.org/prop/P40>(?y, ?w) \cap <https://www.wikidata.org/prop/P3373>(?x, ?w)$
  - $q(?w) :- <https://www.wikidata.org/prop/P40>(?z, ?w) \cap <https://www.wikidata.org/prop/P3373>(?w, ?y) \cap <https://www.wikidata.org/prop/P40>(?x, ?w)$
  - $q(?w) :- <https://www.wikidata.org/wiki/Q308194>(?w) \cap <https://www.wikidata.org/prop/P40>(?w, ?y) \cap <https://www.wikidata.org/prop/P3373>(?x, ?w)$
  - $q(?w) :- <https://www.wikidata.org/prop/P26>(?z, ?w) \cap <https://www.wikidata.org/prop/P1038>(?w, ?y) \cap <https://www.wikidata.org/prop/P3373>(?x, ?w)$
  - $q(?w) :- <https://www.wikidata.org/wiki/Q7565>(?w) \cap <https://www.wikidata.org/prop/P22>(?y, ?w) \cap <https://www.wikidata.org/prop/P40>(?x, ?w)$
  - $q(?w) :- <https://www.wikidata.org/wiki/Q31184>(?w) \cap <https://www.wikidata.org/wiki/Q7566>(?w) \cap <https://www.wikidata.org/prop/P22>(?w, ?x)$
  - $q(?w) :- <https://www.wikidata.org/prop/P26>(?w, ?z) \cap <https://www.wikidata.org/prop/P22>(?y, ?w) \cap <https://$

$//www.wikidata.org/prop/P3373 > (?x, ?w)$

- $q(?w) :- < https : //www.wikidata.org/prop/P3373 > (?w, ?z) \cap < https : //www.wikidata.org/prop/P26 > (?y, ?w) \cap < https : //www.wikidata.org/prop/P3373 > (?x, ?w)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q308194 > (?w) \cap < https : //www.wikidata.org/prop/P3373 > (?y, ?w) \cap < https : //www.wikidata.org/prop/P22 > (?x, ?w)$
- $q(?w) :- < https : //www.wikidata.org/prop/P40 > (?z, ?w) \cap < https : //www.wikidata.org/prop/P3373 > (?y, ?w) \cap < https : //www.wikidata.org/prop/P3373 > (?x, ?w)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q10861465 > (?w) \cap < https : //www.wikidata.org/prop/P3373 > (?w, ?y) \cap < https : //www.wikidata.org/prop/P40 > (?x, ?w)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q595094 > (?w) \cap < https : //www.wikidata.org/wiki/Q7560 > (?w) \cap < https : //www.wikidata.org/wiki/Q5 > (?w)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q7560 > (?w) \cap < https : //www.wikidata.org/wiki/Q177232 > (?w) \cap < https : //www.wikidata.org/wiki/Q7560 > (?w)$
- $q(?w) :- < https : //www.wikidata.org/prop/P3373 > (?z, ?w) \cap < https : //www.wikidata.org/prop/P40 > (?y, ?w) \cap < https : //www.wikidata.org/prop/P40 > (?x, ?w)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q308194 > (?w) \cap < https : //www.wikidata.org/wiki/Q31184 > (?w) \cap < https : //www.wikidata.org/wiki/Q10861465 > (?w)$
- $q(?w) :- < https : //www.wikidata.org/prop/P3373 > (?w, ?z) \cap < https : //www.wikidata.org/prop/P3373 > (?y, ?w) \cap < https : //www.wikidata.org/prop/P26 > (?x, ?w)$
- $q(?w) :- < https : //www.wikidata.org/prop/P3373 > (?w, ?z) \cap < https : //www.wikidata.org/prop/P40 > (?w, ?y) \cap < https : //www.wikidata.org/prop/P26 > (?w, ?x)$
- $q(?w) :- < https : //www.wikidata.org/prop/P3373 > (?w, ?z) \cap < https : //www.wikidata.org/prop/P3373 > (?w, ?y) \cap < https : //www.wikidata.org/prop/P25 > (?w, ?x)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q31184 > (?w) \cap < https : //www.wikidata.org/wiki/Q595094 > (?w) \cap < https :$

$//www.wikidata.org/prop/P40 > (?w, ?x)$

- $q(?w) :- < https : //www.wikidata.org/wiki/Q5 > (?w) \cap < https : //www.wikidata.org/prop/P22 > (?w, ?y) \cap < https : //www.wikidata.org/prop/P25 > (?w, ?x)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q10861465 > (?w) \cap < https : //www.wikidata.org/wiki/Q308194 > (?w) \cap < https : //www.wikidata.org/wiki/Q10861465 > (?w)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q7560 > (?w) \cap < https : //www.wikidata.org/prop/P3373 > (?w, ?y) \cap < https : //www.wikidata.org/prop/P22 > (?w, ?x)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q1196129 > (?w) \cap < https : //www.wikidata.org/wiki/Q177232 > (?w) \cap < https : //www.wikidata.org/wiki/Q7560 > (?w)$

- **pi**:
  - $q(?w) :- < https : //www.wikidata.org/prop/P3373 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P3373 > (?w, ?y) \cap < https : //www.wikidata.org/prop/P3373 > (?w, ?z)$
  - $q(?w) :- < https : //www.wikidata.org/prop/P3373 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P40 > (?w, ?y) \cap < https : //www.wikidata.org/prop/P26 > (?z, ?w)$
  - $q(?w) :- < https : //www.wikidata.org/prop/P40 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P3373 > (?y, ?w) \cap < https : //www.wikidata.org/prop/P22 > (?z, ?w)$
  - $q(?w) :- < https : //www.wikidata.org/prop/P3373 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P26 > (?y, ?w) \cap < https : //www.wikidata.org/prop/P3373 > (?z, ?w)$
  - $q(?w) :- < https : //www.wikidata.org/prop/P3373 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P22 > (?y, ?w) \cap < https : //www.wikidata.org/prop/P26 > (?z, ?w)$
  - $q(?w) :- < https : //www.wikidata.org/prop/P26 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P3373 > (?w, ?y) \cap < https : //www.wikidata.org/prop/P3373 > (?w, ?z)$
  - $q(?w) :- < https : //www.wikidata.org/prop/P25 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P3373 > (?w, ?y) \cap < https : //www.wikidata.org/prop/P40 > (?w, ?z)$

- $q(?w) \text{:-} < https : //www.wikidata.org/prop/P40 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P40 > (?y, ?w) \cap < https : //www.wikidata.org/prop/P3373 > (?z, ?w)$
- $q(?w) \text{:-} < https : //www.wikidata.org/prop/P1038 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P1038 > (?w, ?y) \cap < https : //www.wikidata.org/prop/P40 > (?z, ?w)$
- $q(?w) \text{:-} < https : //www.wikidata.org/prop/P22 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P40 > (?y, ?w) \cap < https : //www.wikidata.org/prop/P40 > (?w, ?z)$
- $q(?w) \text{:-} < https : //www.wikidata.org/prop/P3373 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P26 > (?y, ?w) \cap < https : //www.wikidata.org/prop/P3373 > (?w, ?z)$
- $q(?w) \text{:-} < https : //www.wikidata.org/prop/P22 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P26 > (?w, ?y) \cap < https : //www.wikidata.org/prop/P3373 > (?w, ?z)$
- $q(?w) \text{:-} < https : //www.wikidata.org/prop/P40 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P40 > (?y, ?w) \cap < https : //www.wikidata.org/prop/P40 > (?z, ?w)$
- $q(?w) \text{:-} < https : //www.wikidata.org/prop/P3373 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P26 > (?y, ?w) \cap < https : //www.wikidata.org/prop/P26 > (?w, ?z)$
- $q(?w) \text{:-} < https : //www.wikidata.org/prop/P22 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P40 > (?w, ?y) \cap < https : //www.wikidata.org/prop/P40 > (?z, ?w)$
- $q(?w) \text{:-} < https : //www.wikidata.org/prop/P26 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P40 > (?y, ?w) \cap < https : //www.wikidata.org/prop/P3373 > (?w, ?z)$
- $q(?w) \text{:-} < https : //www.wikidata.org/prop/P26 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P3373 > (?w, ?y) \cap < https : //www.wikidata.org/prop/P22 > (?z, ?w)$
- $q(?w) \text{:-} < https : //www.wikidata.org/prop/P3373 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P40 > (?y, ?w) \cap < https : //www.wikidata.org/prop/P26 > (?w, ?z)$
- $q(?w) \text{:-} < https : //www.wikidata.org/prop/P3373 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P40 > (?w, ?y) \cap < https : //www.wikidata.org/prop/P40 > (?z, ?w)$

- $q(?w) :- < https : //www.wikidata.org/prop/P40 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P22 > (?w, ?y) \cap < https : //www.wikidata.org/prop/P3373 > (?z, ?w)$
- $q(?w) :- < https : //www.wikidata.org/prop/P22 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P26 > (?w, ?y) \cap < https : //www.wikidata.org/prop/P22 > (?z, ?w)$
- $q(?w) :- < https : //www.wikidata.org/prop/P40 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P40 > (?w, ?y) \cap < https : //www.wikidata.org/prop/P40 > (?w, ?z)$
- $q(?w) :- < https : //www.wikidata.org/prop/P3373 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P3373 > (?w, ?y) \cap < https : //www.wikidata.org/prop/P25 > (?w, ?z)$
- $q(?w) :- < https : //www.wikidata.org/prop/P40 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P40 > (?w, ?y) \cap < https : //www.wikidata.org/prop/P40 > (?z, ?w)$
- $q(?w) :- < https : //www.wikidata.org/prop/P40 > (?x, ?y) \cap < https : //www.wikidata.org/prop/P22 > (?y, ?w) \cap < https : //www.wikidata.org/prop/P40 > (?z, ?w)$

- **ip**:
  - $q(?w) :- < https : //www.wikidata.org/wiki/Q308194 > (?z) \cap < https : //www.wikidata.org/prop/P26 > (?y, ?z) \cap < https : //www.wikidata.org/prop/P40 > (?z, ?w)$
  - $q(?w) :- < https : //www.wikidata.org/prop/P3373 > (?z, ?x) \cap < https : //www.wikidata.org/prop/P26 > (?y, ?z) \cap < https : //www.wikidata.org/prop/P40 > (?w, ?z)$
  - $q(?w) :- < https : //www.wikidata.org/wiki/Q7560 > (?x) \cap < https : //www.wikidata.org/wiki/Q5 > (?x) \cap < https : //www.wikidata.org/prop/P22 > (?w, ?x)$
  - $q(?w) :- < https : //www.wikidata.org/prop/P40 > (?z, ?x) \cap < https : //www.wikidata.org/prop/P3373 > (?y, ?z) \cap < https : //www.wikidata.org/prop/P3373 > (?w, ?z)$
  - $q(?w) :- < https : //www.wikidata.org/prop/P3373 > (?z, ?x) \cap < https : //www.wikidata.org/prop/P22 > (?z, ?y) \cap < https : //www.wikidata.org/prop/P22 > (?z, ?w)$
  - $q(?w) :- < https : //www.wikidata.org/prop/P40 > (?z, ?x) \cap <$

$https : //www.wikidata.org/prop/P40 > (?z, ?y) \cap < https : //www.wikidata.org/prop/P3373 > (?w, ?z)$

- $q(?w) :- < https : //www.wikidata.org/prop/P26 > (?x, ?z) \cap < https : //www.wikidata.org/prop/P25 > (?z, ?y) \cap < https : //www.wikidata.org/prop/P22 > (?w, ?z)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q5 > (?x) \cap < https : //www.wikidata.org/wiki/Q7566 > (?x) \cap < https : //www.wikidata.org/prop/P40 > (?x, ?w)$
- $q(?w) :- < https : //www.wikidata.org/prop/P3373 > (?z, ?x) \cap < https : //www.wikidata.org/prop/P40 > (?y, ?z) \cap < https : //www.wikidata.org/prop/P22 > (?z, ?w)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q10861465 > (?x) \cap < https : //www.wikidata.org/wiki/Q7560 > (?x) \cap < https : //www.wikidata.org/prop/P22 > (?x, ?w)$
- $q(?w) :- < https : //www.wikidata.org/prop/P26 > (?x, ?z) \cap < https : //www.wikidata.org/prop/P40 > (?y, ?z) \cap < https : //www.wikidata.org/prop/P22 > (?z, ?w)$
- $q(?w) :- < https : //www.wikidata.org/prop/P1038 > (?x, ?z) \cap < https : //www.wikidata.org/prop/P40 > (?z, ?y) \cap < https : //www.wikidata.org/prop/P3373 > (?z, ?w)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q1196129 > (?x) \cap < https : //www.wikidata.org/wiki/Q1196129 > (?x) \cap < https : //www.wikidata.org/prop/P40 > (?w, ?x)$
- $q(?w) :- < https : //www.wikidata.org/prop/P40 > (?z, ?x) \cap < https : //www.wikidata.org/prop/P25 > (?z, ?y) \cap < https : //www.wikidata.org/prop/P26 > (?z, ?w)$
- $q(?w) :- < https : //www.wikidata.org/prop/P26 > (?z, ?x) \cap < https : //www.wikidata.org/prop/P22 > (?y, ?z) \cap < https : //www.wikidata.org/prop/P22 > (?w, ?z)$
- $q(?w) :- < https : //www.wikidata.org/prop/P3373 > (?x, ?z) \cap < https : //www.wikidata.org/prop/P3373 > (?z, ?y) \cap < https : //www.wikidata.org/prop/P3373 > (?w, ?z)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q595094 > (?x) \cap < https : //www.wikidata.org/wiki/Q7560 > (?x) \cap < https : //www.wikidata.org/prop/P25 > (?w, ?x)$
- $q(?w) :- < https : //www.wikidata.org/prop/P3373 > (?x, ?z) \cap <$

$https$: $//www.wikidata.org/prop/P3373$ > $(?y, ?z)$ ∩ < $https$: $//www.wikidata.org/prop/P40 > (?w, ?z)$

- $q(?w)$:- < $https$: $//www.wikidata.org/prop/P40$ > $(?z, ?x)$ ∩ < $https$: $//www.wikidata.org/prop/P25$ > $(?y, ?z)$ ∩ < $https$: $//www.wikidata.org/prop/P3373 > (?z, ?w)$
- $q(?w)$:- < $https$: $//www.wikidata.org/prop/P40$ > $(?x, ?z)$ ∩ < $https$: $//www.wikidata.org/prop/P22$ > $(?z, ?y)$ ∩ < $https$: $//www.wikidata.org/prop/P3373 > (?w, ?z)$
- $q(?w)$:- < $https$: $//www.wikidata.org/prop/P40$ > $(?x, ?z)$ ∩ < $https$: $//www.wikidata.org/prop/P3373$ > $(?z, ?y)$ ∩ < $https$: $//www.wikidata.org/prop/P22 > (?z, ?w)$
- $q(?w)$:- < $https$: $//www.wikidata.org/wiki/Q7560$ > $(?z)$ ∩ < $https$: $//www.wikidata.org/prop/P22$ > $(?y, ?z)$ ∩ < $https$: $//www.wikidata.org/prop/P3373 > (?w, ?z)$
- $q(?w)$:- < $https$: $//www.wikidata.org/wiki/Q7565$ > $(?x)$ ∩ < $https$: $//www.wikidata.org/wiki/Q7560$ > $(?x)$ ∩ < $https$: $//www.wikidata.org/prop/P40 > (?x, ?w)$
- $q(?w)$:- < $https$: $//www.wikidata.org/wiki/Q177232$ > $(?z)$ ∩ < $https$: $//www.wikidata.org/prop/P40$ > $(?y, ?z)$ ∩ < $https$: $//www.wikidata.org/prop/P22 > (?z, ?w)$
- $q(?w)$:- < $https$: $//www.wikidata.org/prop/P26$ > $(?z, ?x)$ ∩ < $https$: $//www.wikidata.org/prop/P40$ > $(?y, ?z)$ ∩ < $https$: $//www.wikidata.org/prop/P3373 > (?w, ?z)$

- **up**:
  - $q(?w)$:- < $https$: $//www.wikidata.org/wiki/Q308194$ > $(?x)$∪ < $https$: $//www.wikidata.org/wiki/Q595094$ > $(?x)$ ∩ < $https$: $//www.wikidata.org/prop/P22 > (?x, ?w)$
  - $q(?w)$:- < $https$: $//www.wikidata.org/wiki/Q5$ > $(?x)$∪ < $https$: $//www.wikidata.org/wiki/Q31184$ > $(?x)$ ∩ < $https$: $//www.wikidata.org/prop/P1038 > (?w, ?x)$
  - $q(?w)$:- < $https$: $//www.wikidata.org/prop/P3373$ > $(?x, ?z)$∪ < $https$: $//www.wikidata.org/prop/P40$ > $(?y, ?z)$ ∩ < $https$: $//www.wikidata.org/prop/P40 > (?z, ?w)$
  - $q(?w)$:- < $https$: $//www.wikidata.org/prop/P40$ > $(?z, ?x)$∪ < $https$: $//www.wikidata.org/prop/P22$ > $(?z, ?y)$ ∩ < $https$:

$//www.wikidata.org/prop/P3373 > (?z, ?w)$

- $q(?w) :- < https : //www.wikidata.org/wiki/Q595094 > (?z) \cup < https : //www.wikidata.org/prop/P3373 > (?y, ?z) \cap < https : //www.wikidata.org/prop/P22 > (?w, ?z)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q595094 > (?z) \cup < https : //www.wikidata.org/prop/P40 > (?y, ?z) \cap < https : //www.wikidata.org/prop/P40 > (?z, ?w)$
- $q(?w) :- < https : //www.wikidata.org/prop/P40 > (?x, ?z) \cup < https : //www.wikidata.org/prop/P40 > (?y, ?z) \cap < https : //www.wikidata.org/prop/P40 > (?w, ?z)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q177232 > (?x) \cup < https : //www.wikidata.org/wiki/Q1196129 > (?x) \cap < https : //www.wikidata.org/prop/P26 > (?w, ?x)$
- $q(?w) :- < https : //www.wikidata.org/prop/P3373 > (?z, ?x) \cup < https : //www.wikidata.org/prop/P3373 > (?z, ?y) \cap < https : //www.wikidata.org/prop/P25 > (?w, ?z)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q31184 > (?x) \cup < https : //www.wikidata.org/wiki/Q7565 > (?x) \cap < https : //www.wikidata.org/prop/P3373 > (?w, ?x)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q308194 > (?x) \cup < https : //www.wikidata.org/wiki/Q7566 > (?x) \cap < https : //www.wikidata.org/prop/P3373 > (?x, ?w)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q308194 > (?z) \cup < https : //www.wikidata.org/prop/P22 > (?z, ?y) \cap < https : //www.wikidata.org/prop/P22 > (?w, ?z)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q7566 > (?x) \cup < https : //www.wikidata.org/wiki/Q10861465 > (?x) \cap < https : //www.wikidata.org/prop/P3373 > (?x, ?w)$
- $q(?w) :- < https : //www.wikidata.org/prop/P40 > (?x, ?z) \cup < https : //www.wikidata.org/prop/P40 > (?y, ?z) \cap < https : //www.wikidata.org/prop/P25 > (?z, ?w)$
- $q(?w) :- < https : //www.wikidata.org/prop/P25 > (?z, ?x) \cup < https : //www.wikidata.org/prop/P3373 > (?y, ?z) \cap < https : //www.wikidata.org/prop/P22 > (?w, ?z)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q31184 > (?x) \cup < https : //www.wikidata.org/wiki/Q7560 > (?x) \cap < https :$

$//www.wikidata.org/prop/P40 > (?w, ?x)$

- $q(?w) :- < https : //www.wikidata.org/wiki/Q595094 > (?x) \cup < https : //www.wikidata.org/wiki/Q10861465 > (?x) \cap < https : //www.wikidata.org/prop/P25 > (?w, ?x)$
- $q(?w) :- < https : //www.wikidata.org/prop/P40 > (?z, ?x) \cup < https : //www.wikidata.org/prop/P40 > (?z, ?y) \cap < https : //www.wikidata.org/prop/P3373 > (?z, ?w)$
- $q(?w) :- < https : //www.wikidata.org/prop/P22 > (?x, ?z) \cup < https : //www.wikidata.org/prop/P26 > (?y, ?z) \cap < https : //www.wikidata.org/prop/P3373 > (?w, ?z)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q1196129 > (?z) \cup < https : //www.wikidata.org/prop/P3373 > (?z, ?y) \cap < https : //www.wikidata.org/prop/P25 > (?z, ?w)$
- $q(?w) :- < https : //www.wikidata.org/prop/P26 > (?z, ?x) \cup < https : //www.wikidata.org/prop/P40 > (?y, ?z) \cap < https : //www.wikidata.org/prop/P22 > (?w, ?z)$
- $q(?w) :- < https : //www.wikidata.org/prop/P3373 > (?z, ?x) \cup < https : //www.wikidata.org/prop/P3373 > (?y, ?z) \cap < https : //www.wikidata.org/prop/P22 > (?w, ?z)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q7565 > (?z) \cup < https : //www.wikidata.org/prop/P22 > (?y, ?z) \cap < https : //www.wikidata.org/prop/P3373 > (?z, ?w)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q10861465 > (?z) \cup < https : //www.wikidata.org/prop/P26 > (?y, ?z) \cap < https : //www.wikidata.org/prop/P40 > (?z, ?w)$
- $q(?w) :- < https : //www.wikidata.org/wiki/Q595094 > (?x) \cup < https : //www.wikidata.org/wiki/Q7569 > (?x) \cap < https : //www.wikidata.org/prop/P3373 > (?w, ?x)$

- **2u**:
  - $q(?w) :- < https : //www.wikidata.org/prop/P40 > (?x, ?w) \cup < https : //www.wikidata.org/prop/P3373 > (?w, ?y)$
  - $q(?w) :- < https : //www.wikidata.org/prop/P3373 > (?w, ?x) \cup < https : //www.wikidata.org/prop/P26 > (?y, ?w)$
  - $q(?w) :- < https : //www.wikidata.org/prop/P40 > (?w, ?x) \cup < https : //www.wikidata.org/prop/P40 > (?w, ?y)$

- $q(?w) :- <https://www.wikidata.org/wiki/Q177232>(?w) \cup <https://www.wikidata.org/wiki/Q308194>(?w)$
- $q(?w) :- <https://www.wikidata.org/wiki/Q5>(?w) \cup <https://www.wikidata.org/prop/P26>(?x, ?w)$
- $q(?w) :- <https://www.wikidata.org/wiki/Q308194>(?w) \cup <https://www.wikidata.org/wiki/Q177232>(?w)$
- $q(?w) :- <https://www.wikidata.org/wiki/Q10861465>(?w) \cup <https://www.wikidata.org/wiki/Q308194>(?w)$
- $q(?w) :- <https://www.wikidata.org/prop/P3373>(?x, ?w) \cup <https://www.wikidata.org/prop/P40>(?w, ?y)$
- $q(?w) :- <https://www.wikidata.org/prop/P22>(?x, ?w) \cup <https://www.wikidata.org/prop/P26>(?y, ?w)$
- $q(?w) :- <https://www.wikidata.org/prop/P22>(?w, ?x) \cup <https://www.wikidata.org/prop/P40>(?y, ?w)$
- $q(?w) :- <https://www.wikidata.org/wiki/Q31184>(?w) \cup <https://www.wikidata.org/wiki/Q177232>(?w)$
- $q(?w) :- <https://www.wikidata.org/wiki/Q7569>(?w) \cup <https://www.wikidata.org/prop/P40>(?w, ?x)$
- $q(?w) :- <https://www.wikidata.org/prop/P26>(?x, ?w) \cup <https://www.wikidata.org/prop/P3373>(?y, ?w)$
- $q(?w) :- <https://www.wikidata.org/wiki/Q7569>(?w) \cup <https://www.wikidata.org/wiki/Q7560>(?w)$
- $q(?w) :- <https://www.wikidata.org/wiki/Q7565>(?w) \cup <https://www.wikidata.org/wiki/Q10861465>(?w)$
- $q(?w) :- <https://www.wikidata.org/prop/P3373>(?x, ?w) \cup <https://www.wikidata.org/prop/P3373>(?w, ?y)$
- $q(?w) :- <https://www.wikidata.org/wiki/Q177232>(?w) \cup <https://www.wikidata.org/prop/P26>(?x, ?w)$
- $q(?w) :- <https://www.wikidata.org/wiki/Q10861465>(?w) \cup <https://www.wikidata.org/wiki/Q595094>(?w)$
- $q(?w) :- <https://www.wikidata.org/wiki/Q5>(?w) \cup <https://www.wikidata.org/prop/P3373>(?x, ?w)$
- $q(?w) :- <https://www.wikidata.org/wiki/Q5>(?w) \cup <https://www.wikidata.org/wiki/Q177232>(?w)$
- $q(?w) :- <https://www.wikidata.org/wiki/Q1196129>(?w) \cup <https://$

$//www.wikidata.org/prop/P40 > (?x, ?w)$

- $q(?w) \text{:-} < https : //www.wikidata.org/prop/P3373 > (?w, ?x) \cup < https : //www.wikidata.org/prop/P26 > (?w, ?y)$
- $q(?w) \text{:-} < https : //www.wikidata.org/wiki/Q595094 > (?w) \cup < https : //www.wikidata.org/wiki/Q7560 > (?w)$
- $q(?w) \text{:-} < https : //www.wikidata.org/wiki/Q10861465 > (?w) \cup < https : //www.wikidata.org/wiki/Q10861465 > (?w)$
- $q(?w) \text{:-} < https : //www.wikidata.org/prop/P40 > (?w, ?x) \cup < https : //www.wikidata.org/prop/P3373 > (?y, ?w)$

1 2 3

[1] https://github.com/AImenes/query-answering-and-embeddings
[2] https://github.com/AImenes/PerfectRef
[3] https://github.com/AImenes/