**Title**

Neural-Symbolic Methods for Knowledge Graph Reasoning



**Author**

Cheng, Kewei

**Publication Date**

2024

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

# Neural-Symbolic Methods for Knowledge Graph Reasoning

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Kewei Cheng

2024

ABSTRACT OF THE DISSERTATION

Neural-Symbolic Methods for Knowledge Graph Reasoning

by

Kewei Cheng
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2024
Professor Yizhou Sun, Chair

Knowledge graph (KG) reasoning has been studied extensively due to its wide applications, which is addressed by two lines of research, i.e., traditional symbolic reasoning and modern neural networks-based techniques.

Symbolic reasoning has been the most studied approach toward KG reasoning since the earliest days. This approach represents expert knowledge as statements in a logic-based representation language and implements reasoning as logical inference, primarily addressing high-level reasoning and cognitive processes [18]. For instance, consider the logical rule $\textit{SpeakLanguage}(x, y) \leftarrow \textit{LiveIn}(x, z) \wedge \textit{OfficialLanguage}(z, y)$, along with the observed facts *"LiveIn(Mina Miller, USA)"* and *"OfficialLanguage(USA, English)"*, we can infer the new fact *"SpeakLanguage(Mina Miller, English)"*. Symbolic reasoning approaches have demonstrated strong interpretability and generalizability due to the power of logical rules. Nevertheless, they encounter certain limitations when it comes to scaling up to vast datasets, managing ambiguity, and capturing the correlations between entities and relations. To illustrate, consider the case where both *"USA"* and *"United States"* denote the same country. Symbolic reasoning approaches treat them as distinct entities, thereby failing to automatically transfer the knowledge from the existing triple *"LiveIn(Mina Miller, USA)"* to deduce the new fact *"LiveIn(Mina Miller, United States)"*.

Neural networks-based techniques have recently emerged as state-of-the-art methods in KG reasoning, owing to their successful applications [127]. Unlike symbolic reasoning, these techniques

aim to predict unseen triples by exploring the graph structure in KGs to capture the similarity of entities. Neural networks-based techniques have demonstrated good scalability and a strong ability to capture correlations between entities and relations. However, they fall short in modeling higher-order dependencies of KG relations. For instance, while neural networks-based techniques can recognize that *"USA"* and *"United States"* refer to the same country based on their similarity in embeddings, they cannot infer that *"Mary Stilwell"* and *"Mina Miller"* speak *"English"* without leveraging logical rules.

While symbolic reasoning and neural networks-based techniques are typically seen as distinct approaches toward achieving the ultimate goal of KG reasoning, they possess a natural potential to complement and enhance each other. Symbolic approaches offer significant interpretability and generalizability, leveraging logical rules to provide insights into inferred results and enabling easy generalization to unobserved objects. However, they struggle to handle large datasets and fail to capture correlations between entities and relations. Conversely, neural networks-based techniques address the limitations of symbolic approaches by exploring intrinsic similarities among triples, but they lack interpretability and generalizability. By integrating both techniques into a unified framework, neural-symbolic reasoning provides a more efficient, generalizable, and interpretable way to perform KG reasoning. In this thesis, we provide a comprehensive overview of techniques in integrating symbolic logical rules and neural networks for KG reasoning, with a particular focus on two specific tasks, KG completion and logical rule learning and propose our own solutions to address the limitations of existing works.

The dissertation of Kewei Cheng is approved.

Wei Wang

Guy Van den Broeck

Nanyun Peng

Yizhou Sun, Committee Chair

University of California, Los Angeles

2024

*To my mother.*

TABLE OF CONTENTS

**List of Figures** . . . . . **x**

**List of Tables** . . . . . **xv**

**Acknowledgments** . . . . . **xvii**

**Vita** . . . . . **xviii**

**1 Introduction** . . . . . **1**

1.1 What are Knowledge Graphs? . . . . . 1

1.2 Why are Knowledge Graphs Important? . . . . . 2

1.3 Knowledge Graph Reasoning Tasks . . . . . 3

1.4 Two Approaches to KG Reasoning . . . . . 5

1.5 A New Perspective: Integrating Symbolic Reasoning and Representation Learning 6

1.6 Structure of the Dissertation . . . . . 8

**2 Preliminaries on Knowledge Graph and Symbolic Logic** . . . . . **10**

2.1 Knowledge Graph Definition . . . . . 10

2.1.1 KGs in the Language of Graph . . . . . 10

2.1.2 KGs in the Language of Symbolic Logic . . . . . 11

2.2 Symbolic Logic . . . . . 12

2.2.1 Propositional Logic . . . . . 12

2.2.2 First-Order Logic (FOL) . . . . . 13

2.2.3 Literal, Clause, and Other Terminologies . . . . . 15

2.2.4 Fuzzy Logic . . . . . 16

2.3 Logical Rules . . . 17
2.3.1 Knowledge Graph Inference Using Logical Rules . . . 19
2.3.2 Logical Rules in the Context of Knowledge Graph . . . 20
2.4 Notations . . . 21

**3 Knowledge Graph Completion . . . 22**
3.1 Overview . . . 22
3.2 Symbolic Reasoning Based KG Completion . . . 23
3.2.1 Problem Definition . . . 24
3.2.2 Boolean Logic-based Reasoning . . . 25
3.2.3 Probabilistic Logic-based Reasoning . . . 29
3.3 Representation Learning Based KG Completion . . . 34
3.3.1 Overview of Knowledge Graph Embedding Methods . . . 36
3.3.2 Geometric Transformation-based KGEs . . . 38
3.3.3 Bilinear KGE Models . . . 41
3.3.4 Deep Representation Learning-based KGE Models . . . 44
3.3.5 Model Training . . . 48
3.4 Neural-Symbolic Integration for KG Completion . . . 51
3.4.1 Embedding-based Variational Inference for Markov Logic Network (MLN) 52
3.4.2 Fuzzy Logic-based Regularization . . . 55
3.5 UniKER: A Recent Advance of Neural-Symbolic Integration for KG Completion . 61
3.5.1 Framework of UniKER . . . 62
3.5.2 Connection to Existing Approaches . . . 68
3.5.3 Experiments . . . 70
3.6 Summary and Discussions . . . 77

**4 Logical Rule Learning . . . . . 80**

4.1 Overview . . . . . 80

4.2 Search-based Logical Rule Learning . . . . . 82

4.2.1 Inductive Logic Programming (ILP) . . . . . 84

4.2.2 Association Rule Mining-based Approaches . . . . . 89

4.3 Neural Symbolic Integration for Logical Rule Learning . . . . . 93

4.3.1 Differentiable Searching-based Methods . . . . . 94

4.3.2 Learning Based Methods . . . . . 102

4.4 RLogic: A Representation-learning Based Model For Logical Rule Learning . . . . 110

4.4.1 Framework of RLogic . . . . . 110

4.4.2 Connection to Existing Approaches . . . . . 118

4.4.3 Experiments . . . . . 121

4.5 NCRL: An Extension of RLogic . . . . . 128

4.5.1 Framework of NCRL . . . . . 129

4.5.2 Connection to Existing Approaches . . . . . 135

4.5.3 Experiments . . . . . 137

4.6 Summary and Discussions . . . . . 144

**5 Combining Large Language Models and Knowledge Graphs . . . . . 147**

5.1 Overview . . . . . 147

5.2 Multi-step Reasoning with LLMs . . . . . 149

5.2.1 In-context Learning . . . . . 149

5.2.2 Integrate LLMs with Logical Inference . . . . . 150

5.3 Multi-step Reasoning with KGs . . . . . 151

5.4 Structure Guided Prompt: Instructing Large Language Model in Multi-Step Reasoning by Exploring Graph Structure of the Text . . . . . 152
5.4.1 Framework of Structure Guided Prompt . . . . . 153
5.4.2 Exploring Representative KG Reasoning Tasks . . . . . 155
5.4.3 Results . . . . . 160
5.5 Summary and Discussions . . . . . 166

**6 Future Works . . . . . 167**

**Bibliography . . . . . 172**

LIST OF FIGURES

1.1 **(a)** A toy example of KG containing 5 entities, 3 relations, and 4 observed facts. **(b)** A set of new facts denoted as the red dotted lines that can be inferred from existing facts. 3

2.1 An example of a knowledge graph. Each rectangle in the figure represents an entity in the KG and each black line connecting two rectangles in the figure represents a relation in the KG. The dashed lines are the missing links, which can be inferred from the observed facts in KG. . . . . 11

2.2 An example of Kinship knowledge graph. A closed path in the Kinship knowledge graph corresponding to Eq. (2.8) is highlighted by a red dotted triangle. . . . . 20

3.1 Symbolic reasoning over a toy example of KG. **Left**: A toy example of KG. **Right**: A predefined logical rule. The new fact denoted as the red dotted line can be inferred from the logical rule and the KG. . . . . 24

3.2 Illustration of logical reasoning algorithms using the KG in Fig. 3.1. **Left:** using a *MAX-SAT solver* to derive new conclusions from a given set of premises. **Right:** using a *forward chaining algorithm* to iteratively derive new facts from the existing facts in the KG. Different colors are used to represent different types of relations in KG. . . . . 27

3.3 An illustration of Markov Logic Network (MLN). The MLN is constructed based on two logical rules, which are associated with different weights. Two entities $A$ and $B$ are given in this example. Edges are denoted with two different colors, corresponding to the colors used in the rules they are related to. . . . . 30

3.4 Illustration of knowledge graph embedding. **Left**: A toy KG. **Right**: Entities are mapped into an embedding space. Two entities that are close in the embedding space are similar to each other semantically. (Relation embeddings are not shown.) Embeddings are learned from the observed facts in KG (**Left** to **Right**), and the learned embedding can be used to infer new facts, such as the dotted red line on the left panel (**Right** to **Left**). . . . . 36

3.5 Neural-symbolic reasoning over a toy example of KG. **Left**: KG embedding model. **Right**: Traditional logical inference model. Integrating both approaches can lead to enhanced reasoning capabilities. . . . . 51

3.6 Given (a) a KG with observed facts and (b) a set of definite Horn rules, different inference results can be obtained using (c) KGE, (d) logical inference, and (e) the proposed UniKER approach. The synergy of KGE and logical reasoning via UniKER is more powerful than a simple union of (c) and (d). . . . . 62

3.7 Illustration of potential useful hidden triples by taking Eq.(3.34) as an example. . . . . 65

3.8 Proportion to the optimal number of inferred triples w.r.t. #iterations for efficiency analysis of Forward Chaining. . . . . 76

4.1 An instance of a kinship knowledge graph is depicted, where various relationships are denoted by arrows of distinct colors. . . . . 81

4.2 The search space for rules with head relation $r_2$ and body length 2. A path from the root to the leaf corresponds to a possible rule body. Together with the head relation, it forms a candidate rule. The figure highlights an example of a candidate rule that corresponds to the path in red. . . . . 83

4.3 Left: An example KG and a detected rule with length 2. Right: A search tree for rules with length 2 and head relation $r_2$. A path from the root to the leaf corresponds to a rule body. Together with the head relation, it forms a candidate rule. . . . . 89

4.4 Illustration of how the compositionality of logical rules helps improve systematic generalization. (a) logical rule extraction from the observed graph (i.e., training stage) and (b) Inference on an unseen graph (i.e., test stage). The train and the test graphs have disjoint sets of entities. By combining logical rules ① and ② we can successfully learn rule ③ for prediction on unseen graphs. . . . . 107

4.5 Different order to deduct a relation path. (a) left-wise decomposition; (b) irregular decomposition. . . . . 114

4.6 Illustration of the RLogic framework. It contains two main components: (1) a relation path encoder to model $q(r_h|\mathbf{r_b})$; and (2) a close ratio predictor to model $p(r_t|r_h)$. Given a path $\mathbf{r_b}$, the relation path encoder reduces $\mathbf{r_b}$ into a single head $r_h$ by recursively merge relation pairs in path $\mathbf{r_b}$ according to $q(r_h|r_i, r_j)$. Then, the close ratio predictor bridges the gap between "ideal prediction" following logical rules and "real observation" by predicting the ratio that the relation path $\mathbf{r_b}$ will close. . . . . 119

4.7 Deduction from $\delta_1$ and $\delta_3$ to $\delta_2$. . . . . 122

4.8 Equal-length rule detection comparison on Family dataset[1] . . . . . 126

4.9 Longer-length rule detection comparison on Family dataset. Left: overall rule detection performance, where the lengths of rule bodies are varied among $\{2, 3, 4, 5, 6\}$. Right: rule detection performance w.r.t body length. . . . . 126

4.10 Case study of RLogic in inferring rules with different lengths. The figure on the top gives rule bodies with different lengths (i.e., 2, 5, 6). The table at the bottom presents predicted rule heads ranked by probabilities. The bold predicates correspond to the ground truth answers. . . . . 127

4.11 Learning an accurate hierarchical structure is significant for rule discovery: (a) a good compositional structure; (b) an improper compositional structure. . . . . 130

4.12 An overview of NCRL. It samples paths from KG (e.g.,$[r_1, r_3, r_4, r_5, r_3]$), and predicts the relations that directly connect the sampled paths (e.g.,$r_6$) based on the learned rules. NCRL takes the embeddings of predicates in the sampled paths as the input and outputs $\boldsymbol{\theta}$ as the probability of each relation to be the rule head. . . . . 131

4.13 Illustration of how the compositionality of logical rules helps improve systematic generalization. (a) logical rule extraction from the observed graph (i.e., training stage) and (b) Inference on an unseen graph (i.e., test stage). The train and the test graphs have disjoint sets of entities. By combining logical rules ① and ② we can successfully learn rule ③ for prediction on unseen graphs. . . . . 136

4.14 Performance of KG completion vs sparsity ratio on Kinship. . . . . 140

4.15 Performance of KG completion vs # logical rules on Kinship. . . . . 140

4.16 Performance of KG completion vs embedding dimension on Kinship. . . . . 140

5.1 An example illustrating how humans manage multi-step questions. Our objective is to deduce the relationship between two individuals, Michelle and Stanley, highlighted in red, from a given story. Given that the story involves various individuals, humans typically first create a graph to clearly visualize the relationships among them. Then, they infer the relationship step by step, based on the graph. . . . . 149

5.2 GPT-4's performance using 0-shot chain-of-thought (0-CoT) (represented as the orange bars) compared to the results of *Structure Guided Prompt* (represented as the blue bars) across a variety of tasks. It is evident that *Structure Guided Prompt* consistently and significantly outperforms the approach with 0-CoT. . . . . 152

5.3 The graph representation of a story from CLUTRR dataset. our objective is to determine the family relationship between two nodes, *Seth* and *Jeremy*, which are highlighted in red. . . . . 154

5.4 The bridging question in HotpotQA. It relies on multi-hop sequential reasoning to answer the question. . . . . 157

5.5 Main results. Different methods are illustrated through color-coded bars: blue bars indicate the results achieved using our *Structure Guided Prompt*, while orange bars show the performance with 0-shot chain-of-thought(0-CoT). Additionally, green bars depict the performance without 0-CoT. These results demonstrate that the *Structure Guided Prompt* consistently and significantly outperforms the other methods, both with and without 0-CoT, across GPT-3.5 and GPT-4 models. . . . . 159

LIST OF TABLES

2.1 Comparison of concepts in the language of symbolic logic v.s. knowledge graph. . . . 21

3.1 Summary of Notations . . . . . 25

3.2 Summary of Notations . . . . . 39

3.3 Comparison of different neural-symbolic methods for KG completion. . . . . 69

3.4 Comparison of space and time complexity for model training. . . . . 71

3.5 Data Statistics. . . . . 71

3.6 Effectiveness on KG completion task . . . . . 73

3.7 Efficiency Analysis on Family Dataset. . . . . 75

3.8 UniKER-TransE vs. Forward Chaining on Family dataset (whose test set only retains triples that can be inferred by logical rules) on triple *True*/*False* classification task. . . . 76

3.9 Results of reasoning of UniKER-TransE v.s. TransE on Family dataset (whose test set eliminates triples that can derive from logical rules). . . . . 77

4.1 Capabilities of neural-symbolic methods: differentiable searching-based methods v.s. learning-based methods. . . . . 104

4.2 Data statistics. . . . . 122

4.3 Transductive link prediction. The bold numbers represent the best performances among all methods while the underlined numbers represent the best performances among all rule learning methods. . . . . 123

4.4 Inductive link prediction. The bold numbers represent the best performances among all methods. . . . . 124

4.5 Combine learned rules with KG embedding for KG completion task. . . . . 125

4.6 Training time (minutes) of logical rule learning methods on WN18-RR, FB15K-237, and YAGO3-10 datasets. . . . . 128

4.7 Top rules learned by RLogic on FB15K-237. We highlight the predicates which convey the same semantic meaning with boldface. . . . . 129

4.8 Data statistics of widely used benchmark knowledge graphs. . . . . 138

4.9 KG completion. The red numbers represent the best performances among all methods, while the blue numbers represent the best performances among all *rule learning* methods. . . . . 139

4.10 Training time (s) of rule learning methods . . . . . 141

4.11 Data statistics of CLUTRR datasets. . . . . 142

4.12 Data statistics of GraphLog datasets. NC: number of classes. ND: number of distinct resolution edge sequences (distinct descriptors). ARL: average resolution length. AN: average number of nodes. AE: average number of edges. . . . . 142

4.13 Results of inductive relational reasoning on CLUTRR dataset. Trained on path samples with hops $\{2, 3, 4\}$ and evaluated on path samples with hops $\{5, \ldots, 10\}$. The red numbers represent the best performances while the brown numbers represent the second best performances. . . . . 143

4.14 Results of inductive relational reasoning on GraphLog datasets for robustness analysis. 143

4.15 Top rules learned on YAGO3-10. We highlight the composition and predicate which share the same semantic meaning with boldface. . . . . 144

# ACKNOWLEDGMENTS

I wish to extend my heartfelt thanks and respect to my committee chair, Prof. Yizhou Sun, for her invaluable guidance and unwavering support during my doctoral journey. Her expertise, meticulousness, commitment, and visionary outlook have greatly enhanced my research capabilities, shaped my worldview, and motivated my professional aspirations. I am also grateful to my committee members, Prof. Wei Wang, Prof. Guy Van den Broeck, and Prof. Nanyun (Violet) Peng, for their precious time and insightful feedback on my work. My gratitude extends to all my peers and friends at UCLA ScAi Lab for their camaraderie and collaboration, including Ting Chen, Yupeng Gu, Junheng Hao, Xuelu Chen, Yunsheng Bai, Ziniu Hu, Patricia Xiao, Song Jiang, Yewen Wang, Roshni Iyer, Zijie Huang, Shichang Zhang, Zongyue Qin, Derek Xu, Arjun Subramonian, Fred Xu, Yanqiao Zhu, Fang Sun, Weikai Li, Zongyu Lin, Changjun Fan, Jiarong Xu, Xiao Luo, Zijun Xue, Jin Wang, Manoj Reddy, Zeyu Li, Xiusi Chen, Derek Ma, Mandy Wang, and Yijia Xiao. I am equally thankful to my industry collaborators, including Xian Li, Yifan Ethan Xu, Xin Luna Dong, Nesreen K. Ahmed, Theodore Willke, Jingfeng Yang, and Haoming Jiang. My parents deserve special acknowledgment for their endless love and support, which have been my pillars of strength. I cannot fully express my gratitude and love for them. Lastly, my heartfelt appreciation goes to my dear friends who have stood by me and cheered me on throughout my Ph.D. journey. Your presence has added vibrant colors to my life, and I am immensely grateful for your friendship.

# VITA

| | |
|---|---|
| 2011–2015 | Bachelor of Engineering in Software Engineering, Sichuan University (SCU). |
| 2015–2017 | Master of Science in Computer Science, Arizona State University (ASU). |
| 2020 | Applied Scientist Intern, Amazon. |
| 2021 | Applied Scientist Intern, Amazon. |
| 2022 | Research Intern, Intel AI Lab. |
| 2023 | Research Intern, Intel AI Lab. |
| 2023 | Applied Scientist Intern, Amazon. |

# PUBLICATIONS

Kewei Cheng, Xian Li, Zhengyang Wang, Chenwei Zhang, Binxuan Huang, Yifan Ethan Xu, Xin Luna Dong and Yizhou Sun. Tab-Cleaner: Weakly Supervised Tabular Data Cleaning via Pre-training for E-commerce Catalog. In Proceedings of 61st Annual Meeting of the Association for Computational Linguistics (ACL) - Industry Track, 2023

Kewei Cheng, Nesreen Ahmed and Yizhou Sun. Neural Compositional Rule Learning for Knowledge Graph Reasoning. In Proceedings of 11th International Conference on Learning Representations (ICLR), 2023.

Kewei Cheng, Jiahao Liu, Wei Wang and Yizhou Sun. RLogic: Recursive Logical Rule Learning from Knowledge Graphs. In Proceedings of 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), 2022.

Kewei Cheng, Xian Li, Yifan Ethan Xu, Xin Luna Dong and Yizhou Sun. PGE: Product Graph

Embedding Learning for Error Detection. In Proceedings of the 48th International Conference on Very Large Databases (VLDB), 2022.

Kewei Cheng, Ziqing Yang, Ming Zhang and Yizhou Sun. UniKER: A Unified Framework for Combining Embedding and DefiniteHorn Rule Reasoning for Knowledge Graph Inference. In Proceedings of the 36th Conference on Empirical Methods in Natural Language Processing (EMNLP), 2021.

# CHAPTER 1

# Introduction

Knowledge graphs have received more and more attention recently, and have been successfully applied to many domains and downstream tasks. In this dissertation, we aim at providing a novel neural-symbolic perspective in summarizing recent advances in a variety of knowledge graph reasoning tasks. By bridging the gap between neural networks and symbolic reasoning, this dissertation offers a unique and comprehensive perspective on knowledge graph reasoning. It can be used to equip researchers, practitioners, and enthusiasts with the knowledge and tools necessary to tackle the complex challenges posed by knowledge graphs and contribute to the continued advancement of this exciting field.

## 1.1 What are Knowledge Graphs?

On one hand, knowledge graphs (KGs) are a collection of triples storing facts about entities via relations between them, in the form of *(head entity, relation, tail entity)*. For instance, *(Mina Miller, LiveIn, USA)* is a triple, where *"Mina Miller"* and *"USA"* are entities in the KG connected by the relation *"LiveIn"*. On the other hand, as the name suggests, knowledge graphs are essentially graphs, where nodes denote entities and links denote relationships between those entities. A toy example of KG can be found in Fig. 1.1(a), which contains 5 entities, 3 relations, and 4 observed facts denoted as triples (e.g., *(Thomas Alva Edison, IsMarriedTo, Mary Stilwell)*).

Various types of KGs have emerged, which include (1) general-purpose KGs such as DBpedia [3] and Freebase [12]; (2) domain-specific KGs such as STRING [120] in the biomedical domain and Amazon Product Graph [38] in E-Commerce; and (3) common sense KGs such as WordNet [75] and ConceptNet [114] that describe common sense knowledge on general concepts instead of

concrete entities.

## 1.2 Why are Knowledge Graphs Important?

Knowledge has played a critical role in *Human Intelligence*. The knowledge that gets accumulated over time enables humans to conduct reasoning and solve problems. A similar goal has been set for *Artificial Intelligence (AI)*, which aims to mimic actions and decisions taken by humans. Knowledge graphs have served as an important way to represent, store, and manage knowledge, which are usually collected from various sources. Bringing knowledge graphs into AI can systematically improve the performance and reasoning capabilities of AI systems, which have been widely applied to a variety of applications. For example, Google has been using KGs to power its search engine since 2012, which turns *webpage retrieval* into *knowledge retrieval* and thus significantly enhances user experiences [37]. Following Google, many companies such as Meta, Amazon, LinkedIn, etc, launched their own knowledge graphs, encompassing similar ideas. To this day, knowledge graphs have been gaining a lot of momentum. KGs have been actively explored and deployed across multiple domains, such as recommender systems in e-commerce, drug re-purposing in bio-medicine, question and answering (Q&A) systems, and dialogue systems in AI assistants. We provide a few examples of KG applications below.

- **Search engines.** KGs have been widely used in search engines, such as Google [37], Microsoft's Bing [93], and Yahoo [10]. For example, the Google search engine returns related info such as education, birthplace, and spouse in a knowledge panel, when a query is about a famous person.

- **E-Commerce.** KGs have been widely adopted in E-Commerce systems, which are built to capture the behavioral relationship between products and shoppers, such as clickthrough and purchase. These KGs facilitate the recommender systems that are designed for personal recommendations for products and ads. Some examples of companies that are heavily relied on KGs include Amazon [38], eBay [90], and Alibaba [143].

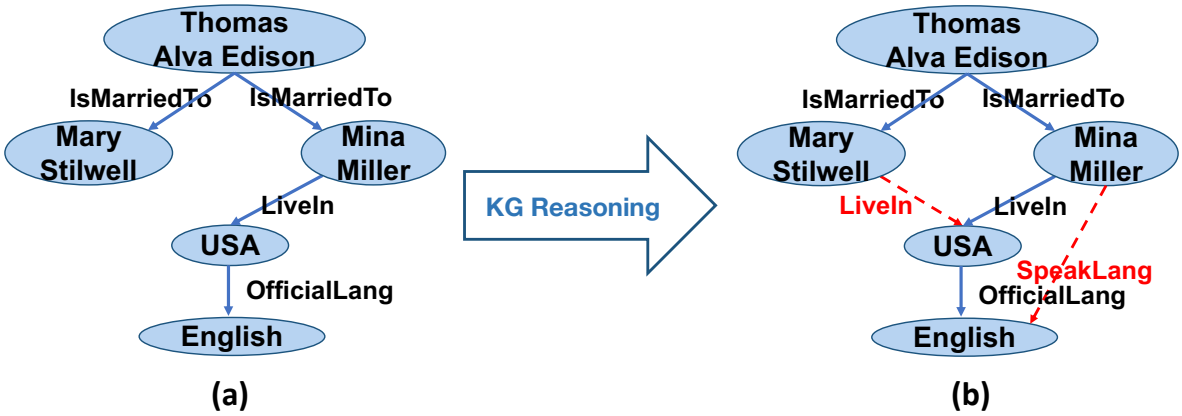



Figure 1.1: **(a)** A toy example of KG containing 5 entities, 3 relations, and 4 observed facts. **(b)** A set of new facts denoted as the red dotted lines that can be inferred from existing facts.

- **Finance.** KGs have been evidenced in financial systems. One application is to construct cooperate KGs to capture business relations between relevant stocks, which is used for stock price movement prediction. Another application is in finance security, where KGs can assist in financial crime prevention and investigation, allowing banking institutions to understand the flow of money across their clientele and identify non-compliant customers. Some representative financial companies using knowledge graphs include Bloomberg [73], Capital One [14], and Wells Fargo [80].

- **Biomedical science and healthcare.** KGs such as Bio2RDF [16] are also benefiting the healthcare industry by organizing and categorizing relationships within medical research, which assist medical providers by validating diagnoses and identifying treatment plans based on individual needs. In addition, biomedical KGs have significantly boosted the new drug design and drug re-purposing [43].

## 1.3 Knowledge Graph Reasoning Tasks

KG reasoning aims at inferring new knowledge from a knowledge graph, by using the relationships and facts encoded in the graph. For instance, we are interested in inferring missing triples or

summarizing the underlying pattern of the KGs in the form of logical rules. In this dissertation, we consider two types of reasoning tasks.

**Knowledge Graph Completion.** Knowledge graph completion aims at predicting one-hop queries by reasoning with existing facts. For example, given the head entity *"Mary Stilwell"* and the relation *"LiveIn"* in Fig. 1.1, the knowledge graph completion task aims to predict the missing tail entity. As the query *(Mary Stilwell, LiveIn, ?)* can be considered as a one-hop link prediction, we call it a one-hop query, in contrast to complex queries such as multi-hop queries and first-order logic queries.

**Logical Rule Induction from KGs.** Logical rules in the form of $head \leftarrow body$ are frequently used by people to conduct reasoning in daily life or specific domains. Most of these logical rules are provided by human experts summarized from their experiences, which are usually labor-intensive to obtain. Logical rule induction from KGs aims at *automatically* learning logical rules by summarizing instance-level facts in KGs to template-level rules. For example, as shown in Fig. 1.1, we may learn the following logical rule:

$$\text{SpeakLanguage}(x, y) \leftarrow \text{LiveIn}(x, z) \wedge \text{OfficialLanguage}(z, y) \tag{1.1}$$

Logical rules serve as another important form of knowledge, which enable reasoning and make the reasoning process transparent, accountable, and trustworthy.

Overall, KG reasoning aims at expanding the knowledge represented in the graph, which is a crucial step in the process of knowledge representation and management and is essential for supporting a wide range of applications, including question-answering, recommendation systems, and decision support systems. By leveraging the relationships and facts encoded in a KG, KG reasoning can provide a powerful and flexible way of inferring new knowledge and making predictions.

## 1.4 Two Approaches to KG Reasoning

The most studied approach to KG reasoning since the earliest days of AI is symbolic reasoning. Symbolic reasoning encodes expert knowledge as statements in a logic-based representation language and implements reasoning in the form of logical inference, which can address high-level reasoning and model thought processes. In this way, symbolic reasoning is good at reasoning abstract concepts and can be used to derive logical conclusions from premises. For example, given a logical rule $\textit{SpeakLanguage}(x, y) \leftarrow \textit{LiveIn}(x, z) \wedge \textit{OfficialLanguage}(z, y)$, and the observed facts *LiveIn(Mina Miller, USA)* and *OfficialLanguage(USA, English)*, we can infer the new fact *SpeakLanguage(Mina Miller, English)* as shown in Fig. 1.1.

Because symbolic reasoning is based on logic and inference rules that are human-understandable, the inference process can be easily interpreted by human beings. Additionally, symbolic reasoning can generalize to new scenarios and unseen entities because logical rules are not limited to specific entities or domains. Despite its strong interpretability and generalizability, symbolic reasoning has two main limitations. First, it is very fragile and heavily relies on the quality of the KGs. In reality, unfortunately, KGs are far from complete and are full of noises and ambiguities. For example, *"USA"* and *"United States"* may refer to the same country, but symbolic reasoning approaches treat them as separate entities. Second, symbolic reasoning fails to scale to large datasets, due to the high complexity nature of the algorithms.

More recently, representation learning-based techniques have been proposed for KG reasoning. Unlike symbolic reasoning, representation learning-based techniques map entities into continuous vectors and relations into operations that are defined over entities. The reasoning tasks are then turned into learning tasks, which are usually less sensitive to noisy data and more scalable. For instance, these techniques can identify that both *"USA"* and *"United States"* refer to the same country, showcasing their capacity for flexible and robust modeling. However, there are several disadvantages of representation learning-based approaches. First, unlike symbolic reasoning, representation learning-based techniques lack the ability to leverage logical rules, limiting their reasoning capacity. For instance, without the capability of leveraging logical rules, representation learning-based techniques cannot infer *"Mary Stilwell"* and *"Mina Miller"* speak *"English"* from

Fig. 1.1. Second, these approaches are data hungry, which cannot generalize to low-resource entities/relations nor handle cold start cases. Third, similar to most neural network approaches, these approaches lack of interpretability, and the reasoning process is hard to understand by human.

In summary, symbolic logical reasoning techniques and representation-learning-based techniques are the two main directions for KG reasoning, which correspond to the first wave of AI and the second wave of AI, respectively. Both lines of research have their pros and cons. It is natural to combine these two lines, i.e., the neural symbolic integration, which is the focus of this dissertation.

## 1.5 A New Perspective: Integrating Symbolic Reasoning and Representation Learning

When considering knowledge graph reasoning, symbolic reasoning and representation learning are often seen as distinct approaches. However, these approaches possess the potential to mutually enhance each other, leading to more comprehensive and effective KG reasoning models.

On one hand, symbolic approaches are particularly well suited for tasks that require precise, logically consistent reasoning. By leveraging logical rules, symbolic approaches offer interpretability and generalizability. They enable insightful interpretations of inferred results and facilitate straightforward generalization to unobserved objects. However, symbolic approaches encounter challenges in scaling to large datasets and navigating complex or ambiguous situations. In such cases, where clear sets of rules or logical relationships are unavailable, symbolic reasoning struggles to provide accurate predictions. On the other hand, representation learning-based techniques address the limitations of symbolic approaches by exploring complex patterns among triples. They excel at capturing intricate relationships and dependencies within knowledge graphs, enabling effective reasoning in the absence of explicit rules. By mapping entities to continuous vectors and leveraging operations to represent relations, representation learning-based techniques offer scalability in handling large datasets. Nevertheless, they lack interpretability and generalizability compared to symbolic approaches. Understanding the reasoning processes and generalizing to new, unseen instances can be challenging with representation learning alone. Furthermore, representation

learning-based techniques often require substantial amounts of training data to achieve optimal performance.

By combining the strengths of symbolic reasoning and representation learning, researchers can achieve a more comprehensive approach to KG reasoning. Leveraging the interpretability and logical consistency of symbolic reasoning, while harnessing the pattern recognition and scalability of representation learning, can lead to neural-symbolic models that deliver a more **scalable**, **generalizable**, and **interpretable** solution to perform KG reasoning.

- **Scalable**: Symbolic reasoning can be computationally expensive for large-scale problems. For example, in a large-scale knowledge graph reasoning task, traditional symbolic methods would not be suitable, as their computational complexity increases exponentially with the size of the knowledge graph. In contrast, representation learning exhibits a formidable computational advantage. This reduction in complexity allows for more efficient and scalable computations, enabling improved performance and broader applicability of neural-symbolic models in handling complex knowledge graphs.

- **Generalizable**: The strong generalizability of symbolic logic can be leveraged to compensate for the lack of data availability for neural methods. For example, although representation learning highly rely on large amounts of high-quality training data, neural-symbolic methods are not significantly impacted by limitations on the amount of available training data. In a few-shot learning task, a neural-symbolic system can use symbolic knowledge as extra data to enrich the limited training samples.

- **Interpretable**: Neural-symbolic systems can provide explicit computation processes, such as a traced reasoning process or a chain of evidence of results. For example, in medical diagnosis, neural-symbolic systems are expected not only to make a decision but also to show the reason for this decision in order to aid the doctor in the diagnosis.

Due to the advantages brought by the neural-symbolic systems, in this dissertation, we will focus on discussing how to seamlessly integrate the discrete symbol of symbolic systems with

the continuous vector of neural systems to improve the reasoning performance of the neural-symbolic systems, which may have the potential to revolutionize the way we interact with and utilize knowledge graphs, by making it easier and more effective to access and use the information stored in these systems.

## 1.6 Structure of the Dissertation

The first chapter provides a general introduction to the problem of KG reasoning and the unique neural-symbolic view of this dissertation. In Chapter 2, we cover the basic concepts and definitions that will be used in later chapters. We discuss each of the KG reasoning tasks in Chapters 3 and 4.

- **Chapter 3: Knowledge Graph Completion.** Knowledge graph completion aims to predict missing facts by reasoning with existing facts. Specifically, this chapter only focuses on *one-hop queries* in contrast to the complex queries mentioned in the next chapter. We give a comprehensive introduction to various methods for KG completion, including (1) traditional symbolic reasoning methods (2) recent representation learning-based methods, and (3) neural-symbolic integration-based methods. In the end, we introduce UniKER [24], our recent advance in the line of neural-symbolic integration, which combines logical reasoning and representation learning to enhance the KG completion task.

- **Chapter 4: Logical Rule Learning.** Logical rules are widely used to represent domain knowledge and hypothesis, which are fundamental to symbolic reasoning-based methods discussed in Chapters 3. Despite the potential benefits brought by logical rules, they are usually *labor-intensive* to obtain in the early days. In this chapter, we provide a comprehensive survey on the problem of *automatically* learning logic rules from knowledge graphs, which include two lines: (1) the traditional searching-based methods; and (2) and recent neural-symbolic integration approaches. In the end, we introduce our RLogic [23] and NCRL [21] algorithm, which is the state-of-the-art neural-symbolic rule learning algorithm.

KGs serve as a powerful method for representing data with complex relations, effectively capturing extended interrelations between entities. This capability makes them instrumental in augmenting

large language models (LLMs) by offering a structured approach to knowledge representation. Following this, we also delve into analyzing the text's graph structure to guide the LLMs in multi-step reasoning.

- **Chapter 5: Combining Large Language Models and Knowledge Graphs.** While KGs are invaluable for their ability to depict complex relational data and trace extended entity relationships, their construction and evolution present significant challenges. Concurrently, large language models (LLMs) like ChatGPT and GPT-4 are advancing the domains of natural language processing and artificial intelligence with their remarkable emergent capabilities and adaptability. Thus, integrating LLMs with KGs can be synergistic, harnessing the strengths of both to enhance data representation and cognitive processing. Given that graphs excel in encoding data with complex relations and in capturing extended interrelations between entities, this chapter concentrates on strategies that combines LLMs and KGs. Specifically, we introduce our *Structure Guided Prompt* framework [22], an innovative three-stage task-agnostic prompting framework to explicitly convert unstructured text into a graph via LLMs and instruct them to navigate this graph using task-specific strategies to formulate responses.

Finally, Chapter 6 concludes this dissertation and lists several future research directions.

# CHAPTER 2

# Preliminaries on Knowledge Graph and Symbolic Logic

## 2.1 Knowledge Graph Definition

In this section, we introduce the definition of knowledge graphs in the languages of both *graph* and *symbolic logic*.

### 2.1.1 KGs in the Language of Graph

A knowledge graph (KG) is a heterogeneous graph, where nodes denote entities and links denote the relations of different types between entities. Formally, a knowledge graph can be defined as follows.

**Definition 1. (Knowledge Graph)** A knowledge graph is a directed graph $\mathcal{G} = \{\mathcal{E}, \mathcal{R}, \mathcal{O}\}$, which consists of a set of *entities* $\mathcal{E}$, a set of *relations* $\mathcal{R}$, and a set of *facts* $\mathcal{O}$. A *fact* is represented by a triple $(e_i, r_k, e_j) \in \mathcal{O}$, where $e_i \in \mathcal{E}$ denotes the *head*, $e_j \in \mathcal{E}$ denotes the *tail*, and $r_k \in \mathcal{R}$ denotes the *relation* between the head and the tail.

**Example 2.1.1.** An example of a knowledge graph is given in Fig. 2.1. This KG contains 9 entities such as *"Bill Gates"* and *"Microsoft"*, 8 relations such as *"Founded"* and *"BornIn"*, and 7 facts such as (*Bill Gates, Founded, Microsoft*).

An interesting reasoning task for the above example might be "*which language does Satya Nadella speak*", denoted as a query *(Satya Nadella, SpeakLanguage, ?)*. This reasoning task is called *KG completion*, which corresponds to the *link prediction* task in graph learning.

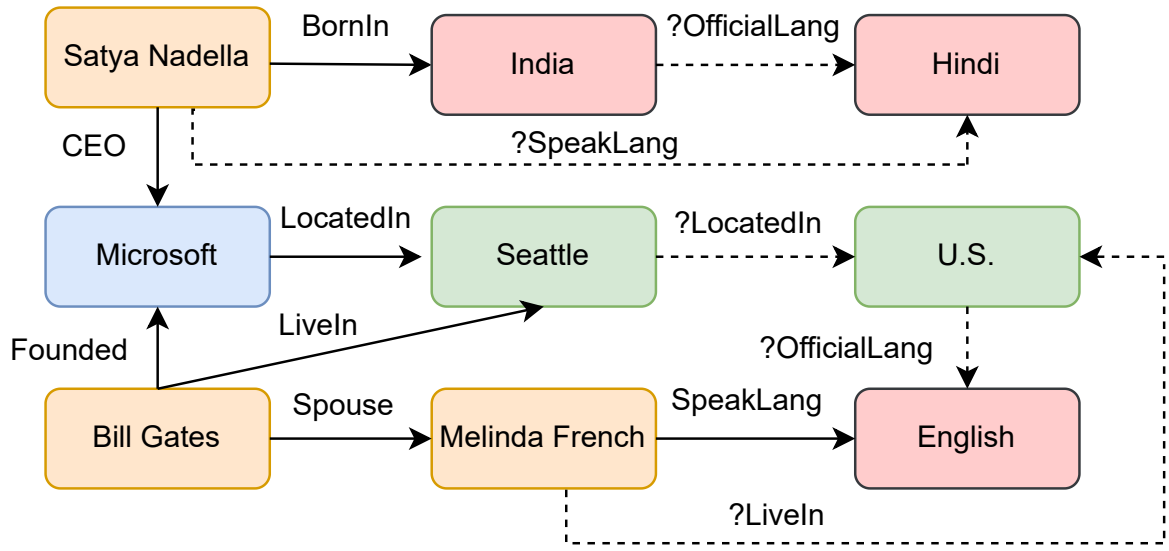



Figure 2.1: An example of a knowledge graph. Each rectangle in the figure represents an entity in the KG and each black line connecting two rectangles in the figure represents a relation in the KG. The dashed lines are the missing links, which can be inferred from the observed facts in KG.

### 2.1.2 KGs in the Language of Symbolic Logic

In addition to the intuitive graph perspective of knowledge graphs, they can also be interpreted and reasoned in the form of symbolic logic. In the world of symbolic logic, entities can be considered as **constants** and relations are binary **predicates**. Each predicate is a function that takes a set of arguments and outputs either *True* or *False* [67]. In the KG setting, each predicate takes two arguments, which can be denoted as $r(\cdot,\cdot)$. **A ground predicate** (a.k.a. **atom** or **atomic formula**) is a predicate whose arguments are all instantiated by particular constants. For example, we may have a predicate *Founded*$(\cdot,\cdot)$. By assigning constants *"Bill Gates"* and *"Microsoft"* to it, we get a ground predicate *Founded(Bill Gates, Microsoft)*. A triple $(e_i, r_k, e_j)$ is essentially a ground predicate, denoted as $r_k(e_i, e_j)$ in the language of logic. In the reasoning task, a ground predicate can be regarded as a binary random variable: $r_k(e_i, e_j) = 1$ when the triple $(e_i, r_k, e_j)$ holds true, and $r_k(e_i, e_j) = 0$ otherwise. **A possible world** assigns a truth value 0 or 1 to each possible ground predicate. Given the observed facts $O$ and their corresponding ground predicates

$\mathbf{v}_O = \{r_k(e_i, e_j)|(e_i, r_k, e_j) \in O\}$, the task of **knowledge graph completion** is to predict the truth value for ground predicates corresponding to all **hidden triples** (i.e., unobserved triples) $H = \Omega \setminus O$, where $\Omega = \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, i.e., the set of all possible triples.

## 2.2 Symbolic Logic

The dissertation has a heavy component of symbolic reasoning. Considering that readers likely lack the background about symbolic logic, we provide a brief but more general introduction to it in this section. A reader who is familiar with the topic can skip this part.

Symbolic logic aims at providing a *formal* language for reasoning, in contrast to *natural* language. *Propositional logic* and *first-order logic* are the two main formal systems for symbolic logic. In the following section, we will define and compare some basic concepts in propositional logic and first-order logic which will be used later in Chapters 3 and 4. A more thorough introduction to symbolic logic can be found in [131, 84, 9].

### 2.2.1 Propositional Logic

In our daily life, a statement is widely used to make a declaration, which is either *True* or *False*. These statements are called propositions and are denoted with symbols such as $P$ and $Q$. A famous example of a proposition is *"Socrates is a man"*, which is *True*. Propositional logic deals with propositions and relations between them, which is also called *zeroth-order logic*.

If a proposition cannot be broken down into smaller propositions, it is called **atomic propositions**. Examples of the atomic propositions include *"All men are mortal"*, *"Socrates is a man"*, and *"Socrates is mortal"*, which can be denoted as $P$, $Q$, and $R$, respectively. Each atomic proposition is assigned a binary truth value, which is called an *interpretation*. For example, $P$ is *True*, denoted as $I(P) = 1$.

Atomic propositions can be used to define more complex propositions via logical connectives (or logical operators), including *negation* ($\neg$), *disjunction* ($\vee$), *conjuction* ($\wedge$), and *implication* ($\rightarrow$). **Logical connectives** can be considered as functions that take propositions as arguments, and

output more complex ones. The truth value of the composite proposition is determined by the truth table of each logical connective. For example, $I(\neg P) = 0$, if $I(P) = 1$. These operators are not independent, and any two of them can define the other two.

With a set of atomic propositions and logical connectives, we can recursively define all valid logical expressions, which are called **well-formed formulas**. *Atomic formulas* or *atoms* are well-formed formulas with no logical connectives. In propositional logic, *atomic formulas* are essentially atomic propositions. Formulas are usually denoted as $\phi$, $\psi$, and $\chi$. Given two well-formed formulas $\phi$ and $\psi$, (1) $\phi \wedge \psi$, (2) $\phi \vee \psi$, (3) $\phi \rightarrow \psi$, and (4) $\neg\phi$ are all well-formed formulas.

Based on the logical language defined above, we can leverage **a rule of inference** in propositional logic such as *Modus Ponens* to deduce new propositions from existing ones. Modus Ponens says if the premises $\phi$ and $\phi \rightarrow \psi$ are both *True*, then the conclusion $\psi$ is also *True*, which can be verified by the truth table. There are other inference rules, and Modus Ponens is the simplest and most popular one.

### 2.2.2 First-Order Logic (FOL)

Propositional logic is limited in its expressive power and can only deal with simple propositional statements that are either *True* or *False*. It cannot represent more complex structures, such as relationships between objects or properties of objects, and does not allow for quantification. For example, *"All men are mortal"* can only be represented as an atomic proposition $P$ in propositional logic without further exploring its inner structure. To address the limitation, first-order logic (FOL) (also known as predicate logic) extends propositional logic by (1) enriching atom representation with objects, functions, and predicates, and (2) introducing variables and quantifiers to formulas.

**Enriching Atom Representation with Objects, Functions, and Predicates** Different from propositional logic, where atomic formulas (atoms) are atomic propositions, FOL atoms have a much richer internal structure. The atoms in FOL contain three types of symbols related to *objects*, *functions*, and *predicates*, which are introduced below.

- **Objects**: Objects refer to things that are of interest, which could be persons, numbers, locations, etc. If an object is known, we use an *constant* to denote it; otherwise, we use a *variable* to denote it, which will be explained later. For example, *Socrates* is an object constant. An object, either a constant or a variable, is a *term*. Objects correspond to entities in KGs.

- **Functions**: Functions take terms as arguments and output objects. An $n$-ary function can be written as $f(t_1, \ldots, t_n)$, where $t_1, \ldots, t_n$ are terms. For example, the function *FatherOf(Socrates)* can be interpreted as *"the father of Socrates"*. Functional expressions are also *terms*. Functions do not typically exist in KGs.

- **Predicates**: Predicates takes terms as arguments and output *True* or *False*. An $n$-ary predicate is written as $P(t_1, \ldots, t_n)$, where $t_1, \ldots, t_n$ are terms. A predicate is an atom. For example, the predicate *Men(Socrates)* is an atom. It means *"Socrates is a man"*, which can only be represented as an atomic proposition $Q$ in propositional logic. In most existing KGs, only binary predicates are considered. For example, *Founded(Bill Gates, Microsoft)* is a binary predicate. If a predicate takes object constants as the arguments, it is called a *ground predicate*. Both *Men(Socrates)* and *Founded(Bill Gates, Microsoft)* are ground predicates.

**Introducing Variables and Quantifiers to Formulas.** In order to accommodate the situations that *all* or *some* of the objects have some property, FOL introduces *object variables* and *quantifiers*, which are discussed below.

- **Variables.** In contrast to the object constants, variables denote any objects, which can be considered as a placeholder. A variable is also a term, which can be the arguments of functions and predicates. For example, $Men(x)$ contains a variable $x$, which can be interpreted as $x$ is a man. Atoms and well-defined formulas may include variables, such as $Men(x)$ and *Founded(x, Microsoft)*.

- **Quantifiers.** *Universal quantifier* (denoted as $\forall$) and *existential quantifier* (denoted as $\exists$) are used to describe whether all objects or some objects make a formula *True*. Assume $\phi$ is

a well-defined formula with a variable $x$, both $(\forall x)\phi$ and $(\exists x)\phi$ are well-defined formulas. For example, we can use $(\forall x)(Men(x) \rightarrow Mortal(x))$ to represent "All men are mortal."

**Well-formed formulas** in first-order logic can be recursively defined over atoms via logical connectives and quantifiers. An example of a well-formed formula is $(\exists x)\big(Founded(x, Microsoft) \wedge Born(x, Seattle)\big)$, representing "there exists a person who founded Microsoft and was born in Seattle."

All the **inference rules** in propositional logic work in first-order logic. There are two additional inference rules, which are called *Universal Instantiation* (UI) and *Existential Generalization* (EG). UI allows us to infer a specific instance of a universally quantified statement. For example, from the statement $(\forall x)(Men(x) \rightarrow Mortal(x))$, we can infer $Men(Socrates) \rightarrow Mortal(Socrates)$. EG allows us to generalize a statement about a specific instance to a statement about a whole class of instances. For example, from the statement $Mortal(Socrates)$, we can infer $(\exists x)Mortal(x)$, which means "there exists at least one person that is mortal."

In summary, propositional logic is a *simple* system for reasoning about truth values of propositions, while first-order logic is a more *expressive* system for reasoning about objects, relations between them, and properties of those objects. First-order logic is a better choice for KG reasoning due to its stronger expressivity.

### 2.2.3 Literal, Clause, and Other Terminologies

Some other terminologies are frequently used in symbolic logic, which are introduced below.

- **Literal**: Literals are either atomic formulas (e.g., $l_1 := P$), called positive literals, or the negation of atomic formulas (e.g., $l_2 := \neg P$), called negative literals.
- **Clause**: A Clause $c$ is a formula defined over a set of literals with logical connectives. A *conjuctive clause* is a conjunction of literals, e.g., $c := l_1 \wedge l_2 \wedge l_3$. A *disjunctive clause* is a disjunction of literals, e.g., $c := l_1 \vee l_2 \vee l_3$.
- **Clausal Form or Conjuctive Normal Form**: Every formula can be represented in a clausal

form, also called conjunctive normal form (CNF), which is a conjunction of disjunctive clauses, e.g., $c_1 \wedge c_2 \wedge c_3$.

- **Disjunctive Normal Form**: Every formula can be represented in a disjunctive normal form (DNF), which is a disjunction of conjunctive clauses, e.g., $c_1 \vee c_2 \vee c_3$.

- **Satisfiable**: A formula is satisfiable if there exists some truth value assignment to each ground atom that makes the formula *True*.

- **De Morgan's Law**: De Morgan's law is a fundamental concept in Boolean logic that enables the expression of conjunctions ($\wedge$) and disjunctions ($\vee$) purely in terms of each other through the use of negation ($\neg$). Symbolically, De Morgan's law can be expressed as:

$$\begin{aligned} \neg(\phi \wedge \psi) &\equiv (\neg\phi \vee \neg\psi) \\ \neg(\phi \vee \psi) &\equiv (\neg\phi \wedge \neg\psi) \end{aligned} \tag{2.1}$$

  These rules state that the negation of a conjunction is equivalent to the disjunction of the negations of the individual propositions, while the negation of a disjunction is equivalent to the conjunction of the negations of the individual propositions.

### 2.2.4 Fuzzy Logic

Classical FOL is Boolean logic, meaning each well-formed formula can only take *True* (denoted as 1) or *False* (denoted as 0). Fuzzy logic extends FOL by allowing a soft truth value in the range $[0, 1]$. We call the mapping from atoms to soft truth values an *interpretation*. The interpretation of atoms $x$ is denoted as $I(x)$. Although fuzzy logic has been widely studied in the literature, there is still no common standard for the computation of fuzzy logic. Here we present some broadly used functions to relax the logical AND, OR, and NOT in fuzzy logic.

- **Gödel Logic.** The Gödel logic is defined by [40]. It simply uses min and max to relax the

logical AND and OR, respectively.

$$x \wedge y = \min\{I(x), I(y)\}$$
$$x \vee y = \max\{I(x), I(y)\}$$
$$\neg x = 1 - I(x).$$

- **Lukasiewicz Logic.** Lukasiewicz logic uses the Lukasiewicz t-norm [61] as the relaxation of the basic logical operations.

$$x \wedge y = \max\{0, I(x) + I(y) - 1\}$$
$$x \vee y = \min\{1, I(x) + I(y)\}$$
$$\neg x = 1 - I(x).$$

- **Product Logic.** The third interpretation follows the paper [50], which uses the following function to approximate the basic logical operations:

$$x \wedge y = I(x)I(y)$$
$$x \vee y = I(x) + I(y) - I(x)I(y)$$
$$\neg x = 1 - I(x).$$

## 2.3 Logical Rules

Logical rules are frequently used to represent symbolic knowledge or hypothesis, which we use heavily for reasoning. How to leverage these logical rules in a differentiable way is key to the success of neural-symbolic integration. We thus formally introduce logical rules and connect them to knowledge graphs.

**Definition 2. (Logical rules)** A logical rule is given in the following standard form:

$$\psi \leftarrow \phi \tag{2.2}$$

where $\psi$ and $\phi$ are well-formed formulas, $\psi$ is called *rule head* (or *conclusion*), and $\phi$ is called *rule body* (or *premise*). It states that whenever the premises (or rule body) is *True*, the conclusion (or rule head) can be derived as *True*.

According to whether the rule is with variables (general knowledge) or just constants (specific knowledge), we have (1) *template rule* and (2) *ground rule*. We will illustrate the difference between them with examples.

- **Template Rule**: a template rule or a first-order logic rule, is a general statement that involves variables and quantifiers. For example, consider the template rule: $(\forall x)(Mortal(x) \leftarrow Men(x))$. This is a general statement that involves only one variable $x$, which can take on any value. In practice, the universal quantifier can be dropped when there is not ambiguity.

- **Ground Rule**: a ground rule is a specific instance of a template rule in which all variables are replaced with concrete objects. For instance, we could make the ground rule *Mortal(Socrates)* $\leftarrow$ *Men(Socrates)* by replacing the variable $x$ in the template rule $(\forall x)(Mortal(x) \leftarrow Men(x))$ with a concrete object *"Socrates"* and applying the inference rule of Universal Instantiation (UI). This ground rule is a specific instance of the more general template rule.

**Definition 3. (Horn rules)** The most widely-used logical rules belong to the Horn rules, which are named after the logician Alfred Horn who first studied them. A Horn rule has the following form: if $P_1 \wedge P_2 \wedge \ldots \wedge P_l$, then $Q$. Formally, a Horn rule written in a (reverse) implication form is given below:

$$Q \leftarrow P_1 \wedge P_2 \wedge \ldots \wedge P_l \tag{2.3}$$

where $P_1, \ldots, P_l$ are positive literals. A Horn rule can also be written as a clause (a disjunction ($\vee$) of literals) with at most one positive literal, thus is also called *Horn clause*:

$$\neg P_1 \vee \neg P_2 \vee \ldots \vee \neg P_l \vee Q \tag{2.4}$$

A Horn rule is said to be *definite* or *strict* if it has exactly one positive literal. For example, the rule in Eq.(2.4) is definite if $Q$ is a positive literal. Horn rules are widely used in the context of KGs to derive new facts from existing ones. As shown in Fig. 2.1, we have the following definite Horn rule:

$$\text{SpeakLanguage}(x, y) \leftarrow \text{BornIn}(x, z) \wedge \text{OfficialLanguage}(z, y) \tag{2.5}$$

where $BornIn(x, z)$, $OfficialLanguage(z, y)$ and $SpeakLanguage(x, y)$ corresponds to $P_1$, $P_2$ and $Q$ in Eq.(2.4), respectively. This rule states that if there is a $z$ that satisfies two conditions: (1) "$x$ was born in $z$" and (2) "$z$ has $y$ as an official language", then we can conclude that "$x$ can speak language $y$". Note variables in a Horn rule are implicitly universally quantified with the scope being the entire Horn rule. The Horn rule in Eq.(2.5) is a simplification of the rule below:

$$(\forall x, y, z)(\text{SpeakLanguage}(x, y) \leftarrow \text{BornIn}(x, z) \wedge \text{OfficialLanguage}(z, y))) \tag{2.6}$$

**Definition 4. (Chain-like Horn rules)** A chain-like Horn rule is a special form of Horn rule. It requires every atom in the rule shares one variable with another atom, where the variables form a chain [136]. A chain-like horn rule in general form is given below:

$$r_h(x, y) \leftarrow r_{b_1}(x, z_1) \wedge r_{b_1}(z_1, z_2) \wedge \cdots \wedge r_{b_n}(z_{n-1}, y) \tag{2.7}$$

where $x, z_1, z_2, \ldots, z_{n-1}, y$ are variables that range over the objects in the KG and $r_{b_1}, \ldots, r_{b_n}, r_h$ are binary predicates in the KG. We can see that Horn rule shown in Eq.(2.5) is chain-like. The body of a chain-like Horn rule corresponds to a relation path starting from $x$ and ending with $y$ in KG.

### 2.3.1 Knowledge Graph Inference Using Logical Rules

Intuitively, logical rules provide high-level knowledge regarding relationships between facts. With the help of such rules, humans can make reasoning based on facts they have observed. As discussed in Section 2.2.1, the basic inference rule is Modus Ponens. The general form of Modus Ponens can be expressed as: if the premises $\phi$ and $\phi \rightarrow \psi$ are both *True*, then we can conclude that $\psi$ is also *True*. For example, we have the following logical rule:

$$\text{hasBrother(Eva, Gino)} \leftarrow \text{hasMother(Eva, Faye)} \wedge \text{hasSon(Faye, Gino)} \tag{2.8}$$

According to Fig. 2.2, we know that its premises *hasMother(Eva, Faye)* and *hasSon(Faye, Gino)* are both *True*. Thus, we can make the conclusion that *Eva* has a brother *Gino*.

The inference process is slightly more complicated when the rules contain variables and quantifiers. Universal instantiation (UI), which involves making a specific instance of a universal

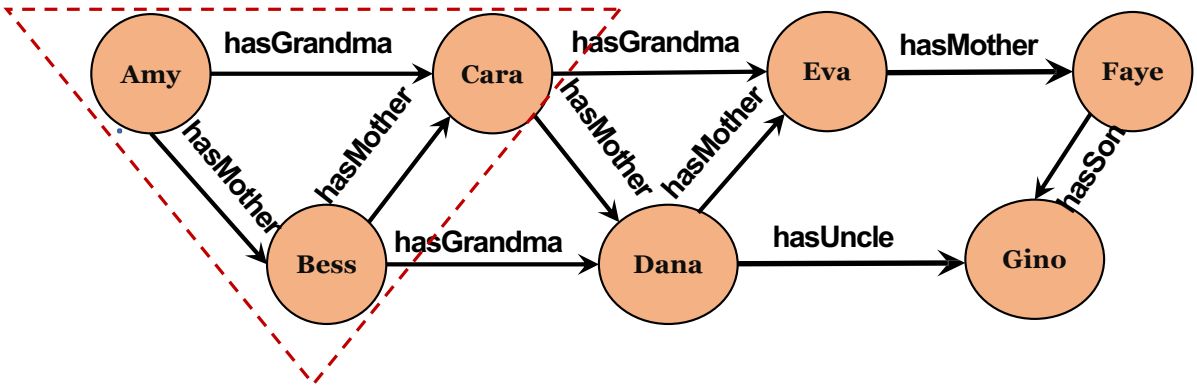



Figure 2.2: An example of Kinship knowledge graph. A closed path in the Kinship knowledge graph corresponding to Eq. (2.8) is highlighted by a red dotted triangle.

quantification by replacing the variables with concrete objects, can be applied. Let us take the KG in Fig. 2.2 as an example to illustrate how to use UI with the following formula:

$$(\forall x, y, z)(\text{hasBrother}(x, y) \leftarrow \text{hasMother}(x, z) \wedge \text{hasSon}(z, y))) \tag{2.9}$$

By replacing the variables $x$, $y$, $z$ with concrete entities *"Eva," "Gino,"* and *"Faye,"* respectively, this general rule leads to a ground rule Eq.(2.8), which is true according to UI. Then we can follow the same reasoning via modus ponens to derive that *hasBrother(Eva, Gino)* is *True*.

In this example, UI allows us to move from a general statement to a specific statement about *"Eva," "Gino,"* and *"Faye."* Following the same way, we can use the same FOL formula with different instantiations of $x$, $y$, and $z$ to infer other individuals' relations.

### 2.3.2 Logical Rules in the Context of Knowledge Graph

The concepts in symbolic logic, including logical rules, are closely connected to the concepts in KGs. In the previous sections, we have mentioned some of the connections. We now summarize them in this section using the example KG in Fig. 2.2. Table 2.1 provides a comparison of how concepts are expressed in the languages of symbolic logic and knowledge graph.

Constants in FOL correspond to entities in KGs, e.g., *"Amy"*. Predicates correspond to relations in KGs, which are binary, e.g., *hasGrandma(x,y)*. Ground predicates correspond to triples in KGs,

Table 2.1: Comparison of concepts in the language of symbolic logic v.s. knowledge graph.

| Symbolic Logic | Knowledge Graph |
|---|---|
| Constants | Entity |
| Predicate | Relation |
| Ground predicate | Triple |
| Rule Instance | Closed Path |

e.g., *hasGrandma(Amy, Cara)*, or *(Amy, hasGrandma, Cara)* in the triple form. A chain-like Horn rule instance can be easily identified as a *"closed path"* in a graph, where the rule body corresponds to a path in the KG, and the rule head corresponds to the edge connecting the starting entity and ending entity in the rule body. For example, given the KG in Fig. 2.2, we can observe closed paths (rule instances) such as:

$$\begin{aligned} \mathcal{CP}_1 &\coloneqq \text{Amy} \xrightarrow{\text{hasMother}} \text{Bess} \xrightarrow{\text{hasMother}} \text{Cara} \xleftarrow{\text{hasGrandma}} \text{Amy} \\ \mathcal{CP}_2 &\coloneqq \text{Dana} \xrightarrow{\text{hasMother}} \text{Eva} \xrightarrow{\text{hasMother}} \text{Faye} \xrightarrow{\text{hasSon}} \text{Gino} \xleftarrow{\text{hasUncle}} \text{Dana} \end{aligned} \tag{2.10}$$

## 2.4 Notations

Throughout this dissertation, we use a boldface lower-case letter $\mathbf{x}$ to represent a vector. Its $i$-th entry is denoted as $\mathbf{x}_i$. The $\ell_p$ norm of a vector for $p \geq 1$ is denoted as $\|\mathbf{x}\|_p$, and $\|\mathbf{x}\|_{1/2}$ means either the $\ell_1$ norm or the $\ell_2$ norm. Let $\text{diag}(\mathbf{x})$ be a diagonal matrix, the $i$-th diagonal entry of which is $\mathbf{x}_i$. $|\mathbf{x}|$ denotes the element-wise absolute value function, $\tanh(\mathbf{x})$ denotes the element-wise hyperbolic tangent function, and $\text{ReLU}(\mathbf{x})$ denotes the element-wise rectified linear unit.

A matrix is represented by a boldface upper-case letter $\mathbf{X}$ with its $ij$-th entry denoted as $\mathbf{X}_{ij}$. $\|\mathbf{X}\|_F$ is the Frobenius norm of a matrix, $\text{tr}(\mathbf{X})$ and $\det(\mathbf{X})$ are the trace and determinant of a square matrix.

# CHAPTER 3

# Knowledge Graph Completion

## 3.1 Overview

A KG is a collection of facts that are believed to be true, and it provides a valuable resource for a wide range of real-world applications, such as semantic parsing, entity disambiguation, information extraction, and question answering [45, 72, 133, 141]. Despite significant efforts to create and maintain KGs, it is impossible to capture all the knowledge in the real world, resulting in significant incompleteness in KGs. To address this issue, the KG completion task aims to predict missing information in a KG. It can be formally formulated as inferring the tail entity given the head entity and the relation (i.e., $(e_i, r_k, ?)$ ) or inferring the head entity given the tail entity and the relation (i.e., $(?, r_k, e_j)$ ). For example, in Fig. 3.1, a query in the KG completion task can be represented as *(Mary Stilwell, LiveIn, ?)*, which asks "where does Mary Stilwell live?"

KG completion can be considered as a *one-hop* query, in contrast to the *complex* queries. We divide existing methods for KG completion into (1) *traditional symbolic reasoning* methods, (2) *recent representation learning-based* methods, and (3) *neural-symbolic integration* methods.

- **Traditional Symbolic Reasoning Methods**: These methods rely on a set of predefined rules to infer new knowledge.
- **Recent Representation Learning-based Methods**: These methods learn high-dimensional embeddings to represent entities and relations in the KG, which can then be used to predict missing facts.
- **Neural-symbolic Integration Methods**: These methods combine symbolic reasoning and

representation learning to improve the accuracy of KG completion. They can leverage the strengths of both symbolic reasoning and representation learning.

In the following sections, we will introduce the most representative methods in each category and discuss their advantages and disadvantages.

## 3.2 Symbolic Reasoning Based KG Completion

In symbolic reasoning-based KG completion, predefined logical rules are used to infer new facts based on the existing facts. These rules are typically expressed in formal logic, such as first-order logic. For instance, in Fig. 3.1, given a logical rule

$$\text{SpeakLanguage}(x, y) \leftarrow \text{LiveIn}(x, z) \wedge \text{OfficialLanguage}(z, y), \tag{3.1}$$

we can answer the query *SpeakLanguage(Mina Miller, ?)* via inference. Specifically, we replace the variables $x, y, z$ in rule (3.1) with concrete entities *"Mina Miller"*, *"English"* and *"USA"* to get the ground rule

$$\begin{aligned}\text{SpeakLanguage(Mina Miller, English)} \leftarrow &\text{LiveIn(Mina Miller, USA)} \\ &\wedge \text{OfficialLanguage(USA, English)}\end{aligned} \tag{3.2}$$

Since both *LivIn(Mina Miller, USA)* and *Official Language(USA, English)* are true due to the two corresponding triples observed in the KG, the rule body is true. By applying Modus Ponens, the conclusion *SpeakLanguage(Mina Miller, English)* is also *True*. Therefore *"English"* should be the answer to our query.

Symbolic reasoning-based KG completion can naturally incorporate domain knowledge in the form of logical rules into the reasoning process. The whole process is transparent and interpretable, which are particularly useful in domains where both effectiveness and interpretability are critical, such as healthcare, finance, and scientific research.

In this section, we will first formally introduce the KG completion task in the language of symbolic reasoning, and then introduce two types of symbolic reasoning approaches for knowledge completion.

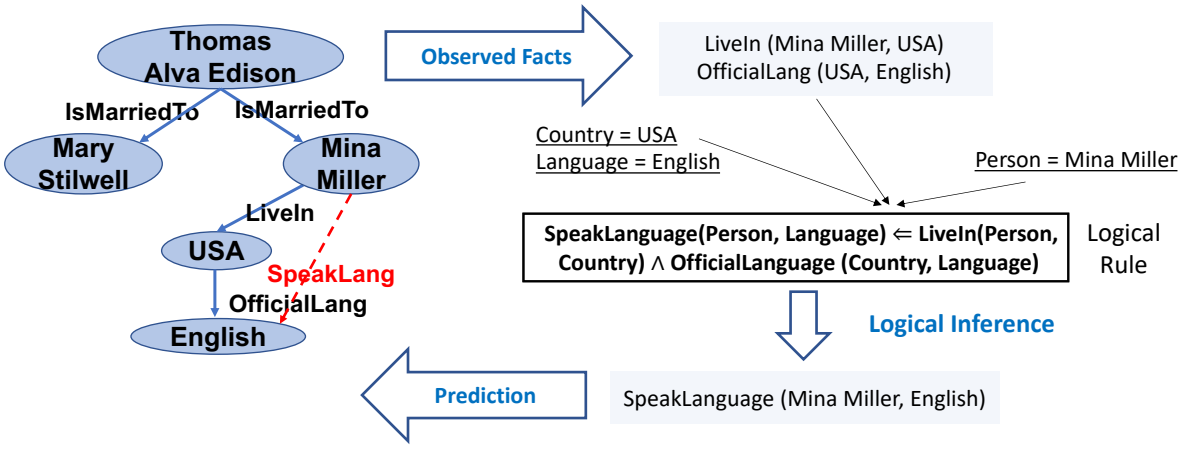



Figure 3.1: Symbolic reasoning over a toy example of KG. **Left**: A toy example of KG. **Right**: A predefined logical rule. The new fact denoted as the red dotted line can be inferred from the logical rule and the KG.

### 3.2.1 Problem Definition

Let $\mathcal{G} = \{\mathcal{E}, \mathcal{R}, \mathcal{O}\}$ be a knowledge graph, where $\mathcal{E}$ denotes the entity set, $\mathcal{R}$ denotes the relation set, and $\mathcal{O}$ denotes the fact set. Each triple $(e_i, r_k, e_j) \in \mathcal{O}$ in KG can be regarded as a ground predicate in the context of symbolic reasoning. We denote the observed ground predicates as $\mathbf{v}_O = \{r_k(e_i, e_j) | (e_i, r_k, e_j) \in \mathcal{O}\}$. For all $r_k(e_i, e_j) \in \mathbf{v}_O$, their truth value is 1. Given a set of predefined logical formulas $\mathcal{F} = \{F_i\}$, where $F_i$ denotes a logical rule, the task of KG completion in the symbolic reasoning view is to assign the truth value for all hidden ground predicates $\mathbf{v}_H = \{r_k(e_i, e_j) | (e_i, r_k, e_j) \notin \mathcal{O}\}$ such that the total number of satisfied logical formulas can be maximized. This is essentially the *maximum satisfiablity problem (MAX-SAT)*. More generally, we can introduce weight $w_i$ to the logical formula $F_i$ to indicate importance or confidence, then it is turned into weighted MAX-SAT problem, i.e., to find the truth value assignment that can maximize the weighted number of satisfied rules.

According to the truth value assigned to each ground predicate, existing symbolic methods can be roughly divided into two categories: (1) *Boolean Logic-based approaches*, where the truth value assignment to predicate $r_k(e_i, e_j)$ is Boolean, i.e., $I(r_k(e_i, e_j)) \in \{0, 1\}$ and (2) *probabilistic*

Table 3.1: Summary of Notations

| | |
|---|---|
| $r_k(e_i, e_j)$ | A ground predicate |
| $I(r_k(e_i, e_j))$ | truth value assigned to the ground predicate $r_k(e_i, e_j)$ |
| $\mathbf{v}_O$ | The set of the observed ground predicates |
| $\mathbf{v}_H$ | The set of the unobserved ground predicates |
| $\mathcal{F}$ | The set of predefined logical formula |
| $(F_i, w_i)$ | logical rule $F_i$ associated with its weight $w_i$ |

*Logic-based approaches*, where the truth value assignment to predicate $r_k(e_i, e_j)$ is in the range of $[0, 1]$, i.e., $I(r_k(e_i, e_j)) \in [0, 1]$. A summary of notations is given in Table 3.1.

### 3.2.2 Boolean Logic-based Reasoning

When each predicate is assigned with either *True* (1) or *False* (0), the knowledge graph completion task is essentially a MAX-SAT problem. The standard form of MAX-SAT problem is a discrete optimization problem, which is to determine the maximum number of satisfied clauses for a given Boolean formula in *conjunctive normal form* (CNF). As introduced in Section 2.2.3, a CNF is a conjunction ($\wedge$) of clauses, where each clause is a disjunction ($\vee$) of literals. A literal can be either an atomic formula or its negation, such as $P$ or $\neg P$. In the context of KG, each atom is a ground predicate of the form $r_k(e_i, e_j) \in \mathbf{v}_O \cup \mathbf{v}_H$; and each logical rule is corresponding to a clause. For example, a rule $Q \leftarrow P_1 \wedge P_2$ can be represented in the form of clause $\neg P_1 \vee \neg P_2 \vee Q$, where $P_1, P_2, Q$ are predicates in KG.

The objective is to identify a truth value assignment (*True* or *False*) for the hidden variables $\mathbf{v}_H$ that maximizes the number of satisfied clauses. When each logical formula $F_i$ is associated with a weight $w_i$, the MAX-SAT problem becomes the *weighted MAX-SAT* problem, where each clause in the given CNF formula is assigned with the corresponding positive weight. The objective is to identify a truth value assignment for the variables that maximizes the total weight of satisfied clauses. If all the weights are equal (i.e., each clause has the same importance), the weighted

MAX-SAT problem reduces to the standard MAX-SAT problem.

The complexity of solving the MAX-SAT problem depends on the size of the given Boolean formula and the number of variables and clauses involved. The MAX-SAT problem is *NP-hard* [88], which means that finding an optimal solution requires exponential time in the worst case. Therefore, *exact solutions* are often infeasible for large and complex instances of the problem. Many *approximation algorithms* have been developed to solve the MAX-SAT problem efficiently in practice. These algorithms aim to find a good approximate solution that is close to the optimal solution, but with lower computational complexity.

*Local search* is one of the most representative approximation algorithms for solving MAX-SAT problem. The main idea behind local search is to explore the solution space by making small changes (flips) to the current assignment of variables, with the goal of finding an assignment that maximizes the number of satisfied clauses (or the total weight of satisfied clauses in the case of weighted MAX-SAT). Let us use a toy example to illustrate how a local search algorithm solves a MAX-SAT problem.

**Example 3.2.1.** Consider the following Boolean formula in CNF:

$$(a \vee b) \wedge (\neg a \vee \neg b) \wedge (b \vee c) \tag{3.3}$$

The goal is to find an assignment of truth values to the variables $a$, $b$, and $c$ that maximizes the number of satisfied clauses.

We start with a random assignment of truth values $a = T, b = T, c = T$. Then we calculate the number of satisfied clauses. We can see that 2 out of 3 clauses are satisfied. After that, we create neighboring assignments by flipping the truth value of a single variable: (1) $a = F, b = T, c = T$, (2) $a = T, b = F, c = T$ and (3) $a = T, b = T, c = F$. Finally, we evaluate the neighboring assignments and choose the one with the maximum number of satisfied clauses. We can see that the best neighboring assignment is (2), where all clauses are satisfied. Since we have found an assignment that satisfies all clauses, the algorithm terminates. The solution is (2) $a = T, b = F, c = T$.

Although logical inference is difficult in general due to the NP-hard nature of MAX-SAT, more efficient algorithms (in polynomial time) can be used when restricting logical rules to definite Horn

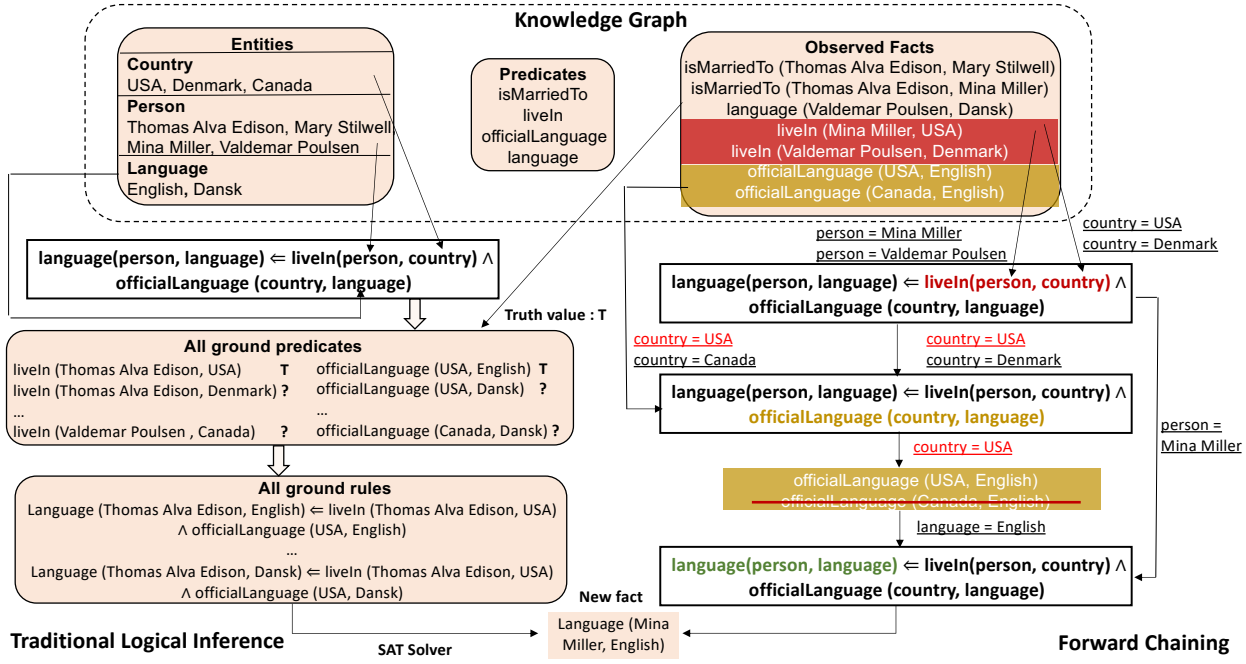



Figure 3.2: Illustration of logical reasoning algorithms using the KG in Fig. 3.1. **Left:** using a *MAX-SAT solver* to derive new conclusions from a given set of premises. **Right:** using a *forward chaining algorithm* to iteratively derive new facts from the existing facts in the KG. Different colors are used to represent different types of relations in KG.

rules (see definition of definite Horn rules in Sec. 2.3). Two commonly used inference algorithms, namely *forward-chaining* and *backward-chaining*, are introduced next.

#### 3.2.2.1 Forward-chaining Algorithm

Forward-chaining algorithm is a *bottom-up* approach that starts with the available evidences and derives new conclusions based on those evidences. More specifically, it starts with a set of initial facts, triggers all rules whose premises are satisfied, adds their conclusion to the known facts, and repeats this process until the problem is solved or no new facts can be added. Let us take the definite Horn rule in Fig. 3.2 as an example to illustrate how the forward-chaining algorithm works.

**Example 3.2.2.** Given the definite Horn rule Eq. (3.1), our goal is to answer the query *Speak-Language(Mina Miller, ?)* using the forward chaining algorithm. We first focus on all observed

triples related to the first predicate in the body, $LiveIn(x, z)$. The variable $x$ is fixed as *"Mina Miller"* in this case. Assuming we know $z$ is a country, it is then limited to a small set of concrete entities $\{$*"USA"*,*"Denmark"*$\}$. Only *liveIn(Mina Miller, USA)* is observed, thus *"Denmark"* can be excluded. After that, we ground the next predicate in the body $OfficialLanguage(z, y)$ with observed triples. By restricting $z$ as *"USA"*, the only applicable triple to ground *OfficialLanguage (z, y)* is *OfficialLanguage (USA, English)*. Finally, we get the ground rule body $LiveIn(MinaMiller, USA) \wedge OfficialLanguage(USA, English)$, and infer the conclusion *SpeakLanguage (Mina Miller, English)*.

#### 3.2.2.2 Backward-chaining Algorithm

As the opposite of the forward-chaining algorithm, a backward-chaining algorithm is a *top-down* approach that starts with the conclusion and works backward to find the supporting evidences and determine whether the conclusion can be satisfied. More specifically, it uses a set of rules and facts to attempt to prove the goal, recursively working backward through the rules to find a set of conditions that can satisfy the goal. Let us use the same example in Fig. 3.2 to rewrite the inference process of the backward chaining algorithm for illustration.

**Example 3.2.3.** Different from the forward chaining algorithm, the first step of the backward chaining algorithm is to start with the goal we want to achieve. In this case, the goal is *speakLanguage (Mina Miller, English)*. The next step is to look at the rules we have and identify which rules can achieve the goal. Here, we only have the rule (3.1). Then, with the rule (3.1), we can work backward to see if any necessary conditions are satisfied. In this example, the goal *speakLanguage (Mina Miller, English)* is achieved by substituting $x$ and $y$ in rule head (i.e., *speakLanguage (x, y)*) with *"Mina Miller"* and *"English"*, respectively. We then add the conditions *LiveIn (Mina Miller, z)* and *OfficialLanguage (z, English)* by iteratively replacing $x$ with *"Mina Miller"* and $y$ with *"English"* in the rule body. Since $z$ is a country, it is limited to a small set of concrete entities $\{$*"USA"*,*"Denmark"*$\}$. By replacing $z$ with *"USA"*, the conditions become *LiveIn (Mina Miller, USA)* and *OfficialLanguage (USA, English)*, which are *True* according to our observation. And hence the goal *speakLanguage (Mina Miller, English)* is proved *True* using backward chaining.

#### 3.2.2.3 Comparing Forward-Chaining and Backward-Chaining

The main difference between forward-chaining and backward-chaining is the direction of reasoning. Forward-chaining works forward from the available evidences, while backward-chaining works backward from the goal (or the conclusion). Forward chaining is data-driven, as it starts from evidences. During the reasoning process, it will infer many conclusions, some of which might not be relevant to our question. Backward chaining is goal-driven, as it starts from the goal. The reasoning process only involves the evidences and rules that are in the reasoning chain, which is usually much faster than forward chaining.

### 3.2.3 Probabilistic Logic-based Reasoning

Although Boolean logic shows effectiveness in capturing complex relationships and knowledge about the world, they are not able to handle the uncertainty present in many real-world applications. In other words, there might be different possible worlds in terms of the truth values taking by each unseen predicate.

To address this issue, probabilistic logic provides a powerful solution by integrating both logic and probabilities. This approach allows for the simultaneous use of the expressiveness of FOL and the robustness of probabilistic approaches to better model and reason KGs with uncertainty. *Markov Logic Network (MLN)* and *Probabilistic Soft Logic (PSL)* are the two most representative probabilistic logic approaches.

#### 3.2.3.1 Markov Logic Network (MLN)

Graphical models are a family of powerful probabilistic models to handle complex dependency among variables. Markov Logic Network (MLN) [100] is a special case of Markov Networks, which is in the family of (undirected) graphical models [125]. Markov Networks represent variables as nodes and their dependency as edges. A joint probability of all the variables can be defined according to the graph structure. In the KG completion tasks, the variables are the ground predicates ($\mathbf{v}_O \cup \mathbf{v}_H$), which are binary variables taking either 1 or 0 (*True* or *False*).

$$(0.9)\ \forall x\ \mathrm{Sport}(x) \Rightarrow \mathrm{Healthy}(x)$$
$$(0.6)\ \forall x, y\ \mathrm{Friends}(x, y) \Rightarrow (\mathrm{Sport}(x) \Rightarrow \mathrm{Sport}(y))$$

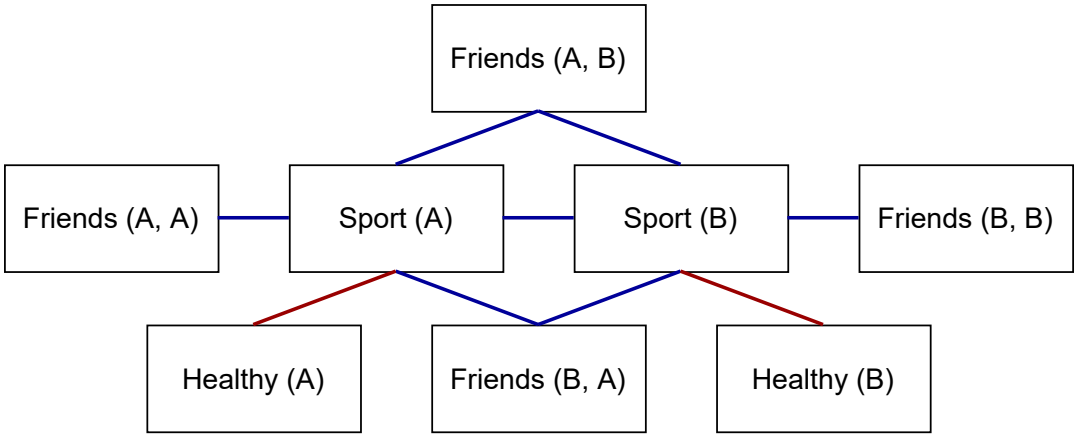



Figure 3.3: An illustration of Markov Logic Network (MLN). The MLN is constructed based on two logical rules, which are associated with different weights. Two entities $A$ and $B$ are given in this example. Edges are denoted with two different colors, corresponding to the colors used in the rules they are related to.

Given a set of FOL formulas $\mathcal{F}$ and their weight vector $\boldsymbol{w}$, MLN defines the graph structure of a Markov network in the following way:

- A node is built for each ground predicate in the MLN, including both observed and unobserved ground predicates in KG. We use $\mathbf{v}_O$ to denote all the variables that are observed, which correspond to the observed triples in KG; and use $\mathbf{v}_H$ to denote all the other ground predicates that are not observed.
- An edge is built to connect two nodes if and only if the corresponding two ground predicates participate in the same ground rule.

**Example 3.2.4.** An example of the graph structure of a Markov Logic Network (MLN) is given in Fig. 3.3. The MLN is composed of two formulas associated with different weights. The red formula indicates that a person who plays sports is usually healthy. The blue formula says that if two people are friends and one of them plays sports, the other person will also play sports. Two entities $A$ and $B$ are given in this example. We can observe that in the ground Markov network, a node for each ground predicate is a Boolean random variable, such as $Sport(A)$ and $Healthy(B)$. These variables are assigned to 1 if their corresponding ground predicate is observed ($\mathbf{v}_O$); and they are unknown otherwise ($\mathbf{v}_H$). An edge is built to connect two nodes if and only if the corresponding two ground predicates participate in the same ground rule. For example, an edge connecting $Sport(A)$ and $Healthy(A)$ is built due to their participation in the rule $Sport(x) \implies Healthy(x)$.

A general Markov network decomposes the joint probability of all the variables according to the cliques (a complete subgraph). In the case of MLN, the cliques are corresponding to the logical rules, as the edges are defined based on whether two variables are in the same logical rules. For example, *Sport(A)* and *Healthy(A)* form a clique with size 2, which correspond to the ground rule *Sport(A)* $\Rightarrow$ *Healthy(A)*. The joint probability of $\mathbf{v}_O$ and $\mathbf{v}_H$ under MLN is then defined as:

$$p_{\boldsymbol{w}}(\mathbf{v}_O, \mathbf{v}_H) = \frac{1}{Z(\boldsymbol{w})} \exp\Big( \sum_{i:F_i \in \mathcal{F}} w_i n_i(\mathbf{v}_O, \mathbf{v}_H) \Big) \tag{3.4}$$

where $w_i$ is the learnable weight for the FOL formula $F_i$, $n_i(\mathbf{v}_O, \mathbf{v}_H)$ is the number of ground formulas of $F_i$ that are *True* given the values of $\mathbf{v}_O$ and $\mathbf{v}_H$ (e.g., the number of times that *Sport(x)* $\Rightarrow$

*Healthy(x)* are true given the assignments of *Sport(A)*, *Sport(B)*, *Healthy(A)*, *Healthy(B)*), and $Z(\boldsymbol{w})$ is a normalization constant to make the probabilities of all worlds sum to one.

The KG completion task then becomes the *inference problem* for $\mathbf{v}_H$ in MLNs, which involves computing the most probable assignment of values to the unseen triples in the KG, given the observed triples and the model: $\max_{\mathbf{v}_H} p_{\mathbf{w}}(\mathbf{v}_H|\mathbf{v}_O) \equiv \max_{\mathbf{v}_H} p_{\mathbf{w}}(\mathbf{v}_O, \mathbf{v}_H)$. MLN inference is computationally expensive, which is #P-complete [100]. To alleviate the burden of computing, MLN inference usually uses Markov chain Monte Carlo (MCMC), and in particular, Gibbs sampling [17, 92] to get an approximate solution. The basic idea is to sample each ground predicate in turn given its Markov blanket. A Markov blanket of a ground predicate is the set of ground predicates that appear together with it in any ground rule. Another popular method for MLN inference is belief propagation [109, 140]. It accelerates inference in MLNs by exploiting dependencies in the network structure. The key idea behind belief propagation is to propagate beliefs, or probability distributions, through the network in a message-passing algorithm. The algorithm iteratively updates the beliefs of each node in the network based on messages received from its neighbors.

In the inference problem, we usually assume the weights for each formula $\boldsymbol{w}$ are given. The weights can actually be learned by maximizing the likelihood: $\max_{\boldsymbol{w}} p_{\boldsymbol{w}}(\mathbf{v}_O, \mathbf{v}_H)$, which corresponds to the *learning problem*. The weights are usually learned approximately by simplyfing the original objective via dropping certain dependencies, such as pseudo-likelihood [7]. The learning problem is usually not discussed in the KG completion setting.

Although MLN provides a good theoretical model in combining logic and probabilistic model, it is hard to scale to large KGs for inference, due to (1) the expensive process of grounding all the predicates and logical rules; and (2) even the approximate inference algorithm takes a long time to converge.

#### 3.2.3.2 Probabilistic Soft Logic (PSL)

Probabilistic Soft Logic (PSL) [4] provides another way to incorporate uncertainty into the reasoning process. It relaxes the Boolean assignment to each variable to a soft probability, which allows for the underlying inference to be solved quickly as a *convex optimization problem*. We denote

$I(r_k(e_i, e_j))$ the **soft truth values** in an interval between $[0, 1]$ to a ground predicate $r_k(e_i, e_j)$, indicating how likely the ground predicate holds true. We call the mapping $I : \mathbf{v}_O \cup \mathbf{v}_H \rightarrow [0, 1]$ as an **interpretation**.

To determine the degree to which a ground rule is satisfied, PSL uses the Lukasiewicz t-norm [49] and its corresponding t-conorm as the relaxation of the logical AND and OR to non-binary variables, which has been introduced in Section 2.2.4:

$$
\begin{aligned}
I(x \wedge y) &= \max\{0, I(x) + I(y) - 1\} \\
I(x \vee y) &= \min\{1, I(x) + I(y)\} \\
I(\neg x) &= 1 - I(x).
\end{aligned}
$$

where $x$ and $y$ can be any well-defined formulas including ground rules (i.e., $f$) and ground predicates (i.e., $r_k(e_i, e_j)$). We can use the above functions to calculate the soft truth value for any ground rule $f$, denoted as $I(f)$, as a ground logical rule can be transformed from implication form to disjunction form as shown below:

$$
r_{\text{head}} \leftarrow r_{\text{body}} \equiv r_{\text{head}} \vee \neg r_{\text{body}} \tag{3.5}
$$

We define $d(f) = 1 - I(f)$ as **distance to satisfaction**. It softly measures the degree to which a ground rule is violated.

By applying the above definitions of the logical operators to a ground logical rule in disjunction form, the distance to satisfaction of a ground logical rule can be computed as:

$$
\begin{aligned}
d(f) &= 1 - I(f) \\
&= 1 - \min(1, I(r_{\text{head}}) + I(\neg r_{\text{body}})) \\
&= \max\{0, I(r_{\text{head}}) - I(r_{\text{body}})\}
\end{aligned}
$$

We can observe that a ground rule $f$ is satisfied (when $d(f) = 0$) if and only if the truth value of its head $I(r_{\text{head}})$ is the same or higher than its body $I(r_{\text{body}})$.

Given a set of ground atomic formulas (predicates), a PSL program defines a distribution over possible interpretations $I : \mathbf{v}_O \cup \mathbf{v}_H \rightarrow [0, 1]$. Let $\mathcal{F}$ be the set of all ground rules that are instances

of a rule in the program, the probability density function over $I$ is:

$$\frac{1}{Z(\boldsymbol{w})} \exp\Big( \sum_{i:f_i \in \mathcal{F}} w_i (d(f_i))^p \Big) \tag{3.6}$$

where $w_i$ is the weight of the ground rule $f_i$, $Z(\boldsymbol{w})$ is the continuous version of the normalization constant for $\boldsymbol{w}$ to make the probabilities of all worlds sum up to one, and $p \in \{1, 2\}$ provides a choice of two different loss functions. Informally, the linear loss function ($p = 1$) favors interpretations that completely satisfy one rule at the expense of higher distance from satisfaction for conflicting rules, whereas the quadratic loss function ($p = 2$) favors interpretations that satisfy all rules to some degree, which typically have truth values farther away from the extreme values.

Unlike MLN, the underlying inference problem of PSL can be mathematically modeled as a convex optimization problem, making it easier to solve and more efficient than MLN. This also enables PSL to be more scalable, making it a better choice for large-scale applications. In addition, PSL's templates are designed to be flexible, allowing users to easily customize them to suit their needs, thus making them generally easier to use than MLN. For instance, PSL can be utilized to tackle the entity resolution task effectively [4]. An entity resolution task involves determining whether two entities in a dataset refer to the same real-world object. The degree to which two entities are believed to be the same depends on various factors, such as how similar their names are. In PSL, we can define a rule that expresses this dependency as follows:

$$1.0 : \text{Same}(P_1, P_2) \leftarrow \text{Name}(P_1, N_1) \wedge \text{Name}(P_2, N_2) \wedge \text{Similar}(N_1, N_2) \tag{3.7}$$

where $P_1, P_2$ denote two persons and $N_1, N_2$ denote their names.

Similar to MLN, PSL-based approaches still require the grounding of the logical rules, which is time-consuming for large-scale KGs.

## 3.3 Representation Learning Based KG Completion

Despite the strong interpretability and generalizability of traditional symbolic methods in performing reasoning over a KG, they are not applicable to real-world KGs due to real-world KGs are usually (1) large-scale, (2) complex, and (3) noisy.

- **Large-scale** Symbolic reasoning approaches usually have high computational complexity. In the inference process of symbolic reasoning-based KG completion, logical rules need to be grounded as all ground rules are necessary for inferring missing knowledge. Considering all possible combinations of entities for each rule, the complexity of grounding a logical rule is $O(|\mathcal{E}|^n)$, where $|\mathcal{E}|$ represents the number of entities in the KG and $n$ is the number of variables in the logical rules. For instance, in Eq.(3.1), we have three variables $x$, $y$, and $z$, thus the complexity of grounding Eq.(3.1) is $O(|\mathcal{E}|^3)$.

- **Complex** Symbolic reasoning approaches rely on the explicit representation of logical rules, which require a large amount of human experts' effort to create. Since logical rules may not be able to capture all the nuances of the data, symbolic reasoning-based KG completion methods also struggle to capture complex patterns in the data. For instance, symbolic reasoning approaches may struggle with handling ambiguity and capturing the correlation between entities and relations when multiple entities represent the same entity, such as "Los Angeles" and "LA".

- **Noisy** Symbolic reasoning approaches are sensitive to the noisy triples in KG. Since KG construction usually involves automatic information extraction, some of the triples might inevitably be wrong.

To address the above issues, knowledge graph embedding (KGE) methods have been widely used recently for KG completion. The general idea of KGE methods is to represent *discrete symbols* such as entities and relations with low-dimensional *continuous vectors* [127], which are called *embeddings*, such that the KG structure can be preserved using these vectors. The embeddings then can be used to predict new facts. The idea is illustrated in Fig. 3.4.

KGE methods can be trained efficiently using stochastic gradient descent algorithms, which can converge quickly and lead to good results with relatively few iterations. They are highly scalable and can handle real-world large KGs with millions of entities and relations. Additionally, KGE methods can capture complex patterns in the data, such as addressing entity disambiguation. For instance, KGE methods can learn distinct sub-representations for the entity "Apple" that allows

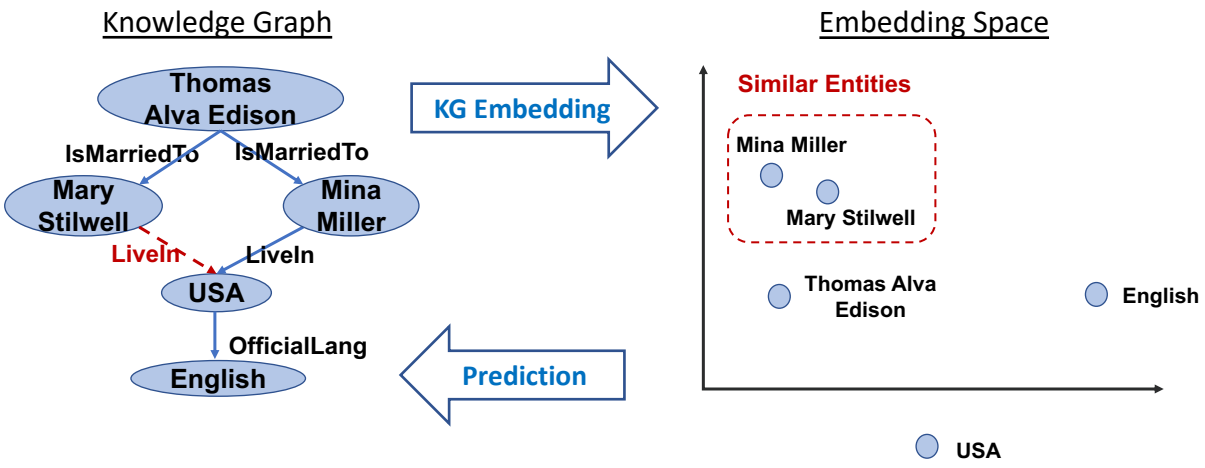



Figure 3.4: Illustration of knowledge graph embedding. **Left**: A toy KG. **Right**: Entities are mapped into an embedding space. Two entities that are close in the embedding space are similar to each other semantically. (Relation embeddings are not shown.) Embeddings are learned from the observed facts in KG (**Left** to **Right**), and the learned embedding can be used to infer new facts, such as the dotted red line on the left panel (**Right** to **Left**).

them to differentiate between the two meanings of "Apple" in the two triples: *(Steve Jobs, eat, apple)* and *(Steve Jobs, co-found, Apple)*. KGE methods can also naturally handle the noises in KG and preserve the similarity between similar entities and similar relations in their embedding vectors.

### 3.3.1 Overview of Knowledge Graph Embedding Methods

Unlike traditional symbolic methods, KGE only relies on observed facts in KG without leveraging logical rules. Score functions, which measure the plausibility of triples in KGs, are the crux of KGE models. According to different score functions, existing KGE methods can be roughly divided into three categories: (1) *geometric operation-based models*, (2) *bilinear models*, and (3) *deep learning-based models*. We will introduce the most representative methods in each category in Sec. 3.3.2, 3.3.3, and 3.3.4, respectively.

A typical KG embedding method consists of three components: (1) an encoder that maps

entities and relations into embeddings; (2) an embedding-based score function that measures the plausibility of every triple; and (3) a loss function that measures how good the model parameters are. We now give a brief introduction to the three components.

#### 3.3.1.1 The Embedding Encoder

Given a knowledge graph $\mathcal{G} = \{\mathcal{E}, \mathcal{R}, \mathcal{O}\}$, the encoder maps entities $e \in \mathcal{E}$ into an low dimensional vectors and relations $r \in \mathcal{R}$ into operations defined over those vectors. Earlier embedding models use *shallow* encoders, which associate every entity and relation with a learnable embedding vector. Later embedding models use *deeper* encoders, and the embedding vectors for each entity and relation can be computed via a deep neural network, whose parameters can be learned.

#### 3.3.1.2 The Score Function and the Properties of Relations

The next question is to decide the plausibility of a triple given the entity and relation representations, which is achieved by a score function $f : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \rightarrow \mathbb{R}^+ \cup \{0\}$. It assigns a plausibility score $f_{r_k}(e_i, e_j)$ to a triple $(e_i, r_k, e_j) \in \mathcal{O}$, where $e_i, e_j \in \mathcal{E}$ and $r_k \in \mathcal{R}$. The higher the score, the higher confidence that the triple is a true fact.

The choice of scoring function plays a crucial role in modeling a diverse spectrum of relations in KGs, which exhibit different properties based on their different semantic nature. Some important properties of relations include:

- **Symmetric**: A relation $r_k$ is symmetric if $\forall e_i, e_j : r_k(e_i, e_j) \leftrightarrow r_k(e_j, e_i)$. For example, *Spouse* is a symmetric relation.

- **Asymmetric**: A relation $r_k$ is asymmetric if $\forall e_i, e_j : r_k(e_i, e_j) \rightarrow \neg r_k(e_j, e_i)$. For example, *Father* is an asymmetric relation.

- **Reflexive**: A relation $r_k$ is reflexive if $\forall e_i : r_k(e_i, e_i)$ is *True*. For example, *EqualTo* is a reflexive relation.

- **Ir-reflexive**: A relation $r_k$ is ir-reflexive if $\forall e_i : r_k(e_i, e_i)$ is *False*. For example, *LessThan* is

an ir-reflexive relation.

- **Transitive**: A relation $r_k$ is transitive if $\forall e_i, e_j, e_z : r_k(e_i, e_z) \wedge r_k(e_z, e_j) \rightarrow r_k(e_i, e_j)$. For example, *Similar* is a transitive relation.

There are also constraints between different relations:

- **Inversion**: Two relations $r_1$ and $r_2$ are inverse to each other if $\forall e_i, e_j : r_1(e_i, e_j) \leftrightarrow r_2(e_j, e_i)$. For example, *superclass* and *subclass* are inverse to each other.

- **Composition**: a relation $r_3$ is a composition of relations $r_1$ and $r_2$ if $\forall e_i, e_j, e_l : r_1(e_i, e_j) \wedge r_2(e_j, e_l) \rightarrow r_3(e_i, e_l)$. For example, $MotherInLaw(e_i, e_l)$ is a composition of $Spouse(e_i, e_j)$ and $Mother(e_j, e_l)$.

A powerful embedding method should be able to capture a wider range of properties. We will discuss which properties each KGE method can capture in the following subsections.

#### 3.3.1.3 The Loss function

The loss function aims to measure the consistency between the model-based score function and the "ground-truth" observation in KGs. There are two types of widely used loss functions. The first is to treat the problem as a classification task and tries to differentiate the positive triples and the sampled negative triples. The second is to treat the problem as a ranking task and tries to rank the positive triples higher than the sampled negative triples. More discussion will be presented later in Sec. 3.3.5.

Before we introduce the technical details of KGE methods, we first summarize notations in Table 3.2.

### 3.3.2 Geometric Transformation-based KGEs

In this line, entities are treated as points in a geometric space, and relations are treated as geometric operations that transform the head entity into the tail entity. Both entities and relations are

Table 3.2: Summary of Notations

| | |
|---|---|
| $(e_i, r_k, e_j)$ | A triple fact<br>where $e_i$ is head entity, $e_j$ is tail entity and $r_k$ is relation |
| $(\mathbf{e_i}, \mathbf{r_k}, \mathbf{e_j})$ | The embedding vectors corresponding to $(e_i, r_k, e_j)$ |
| $f_{r_k}(e_i, e_j)$ | The score function for triple $(e_i, r_k, e_j)$ |

associated with learnable embedding vectors, which belong to shallow embedding. The most commonly utilized geometric operations to represent relations in KGs are the *translational* and *rotational operations*.

#### 3.3.2.1 Translation-based Models: TransE and Its Variants

Translation-based methods treat entities as points in a $d$-dimensional Euclidean space, and relations as translational operations, which can be represented as a $d$-dimensional vector. TransE [13] is the most representative translation-based model, which proposed the translation idea for KG embedding for the first time. We have provided a figure to illustrate the translation idea in Fig. **??**(a). Formally, TransE assumes entities are points in the $\mathbb{R}^d$ space, i.e., $\mathbf{e}_i, \mathbf{e}_j \in \mathbb{R}^d$, and relations correspond to translation operations, which are represented as $d$-dimensional vectors, i.e., $\mathbf{r_k} \in \mathbb{R}^d$. For a triple $(e_i, r_k, e_j)$, the score function is defined as the negative distance between $\mathbf{e_i} + \mathbf{r_k}$ and $\mathbf{e_j}$, i.e., the distance between the projected tail entity and the true tail entity:

$$f_{r_k}(e_i, e_j) = \left\| \mathbf{e_i} + \mathbf{r_k} - \mathbf{e_j} \right\|_2^2 . \tag{3.8}$$

Despite its simplicity, TransE cannot address relations that are reflexive, symmetric, 1-to-N, N-to-1, and N-to-N due to the nature of the translation operation, which is a one-to-one mapping. Taking *"Friend"* as an example, it is a symmetric relation, i.e., if $Friend(e_i, e_j)$ then $Friend(e_j, e_i)$. But according to the translation operation, if $\mathbf{e}_i + \mathbf{r}_k = \mathbf{e}_j$, then $\mathbf{e}_j + \mathbf{r}_k \neq \mathbf{e}_i$ unless $\mathbf{r}_k = 0$. Taking *"hasSibling"* as another example, it is a 1-to-N relation, and we might observe multiple siblings for the same entity. TransE, however, can only project the entity to one point via the translation operation.

In order to overcome these limitations, different variants of TransE have been proposed, such as TransH [128], TransR [69], and TransD [57].

#### 3.3.2.2 Rotation-based Model: RotatE

Other than the translation-based methods, an alternative algorithm called RotatE [117] is proposed later, which is based on the rotation operation in the complex space. RotatE defines entities in the complex space, where each embedding dimension is a complex number, i.e., $\mathbf{e_i}, \mathbf{e_j} \in \mathbb{C}^d$. Relations are defined as rotation operations on each dimension, i.e., $\mathbf{r_k} \in \mathbb{C}^d$ and each dimension has a modulus 1 (a.k.a., $|r_{k,m}| = 1$ for every dimension $m$), which rotate each complex number in head entities into a different one in tail entities. The rotation operation for a complex number can be computed as a product of the two complex numbers. Formally, given a triple $(e_i, r_k, e_j)$, the rotation of head entity $e_i$ with $r_k$ is defined as $\mathbf{e_i} \circ \mathbf{r_k}$, where $\circ$ denotes the Hadamard (element-wise) product, which is expected to be close to the tail entity $\mathbf{e}_j$.

RotatE has the ability to model relations that exhibit different properties, such as *symmetry/asymmetry*, and accommodate relationships between relations such as *inversion* and *composition*. Relations with different types of properties require different $\mathbf{r_k}$. To be more precise, the requirements that need to be satisfied for each property are as follows:

- **Symmetric Relation**: Each dimension of a symmetric relation embedding satisfies $\mathbf{r}_{k,m} = e^{0/i\pi} = \pm 1$, which guarantees applying the rotation twice goes back to the same entity.
- **Asymmetric Relation**: At least one of $\mathbf{r_k}$'s dimensions does not satisfy $\mathbf{r}_{k,m} = e^{0/i\pi} = \pm 1$.[1]
- **Inversion**: For inverse relations, their embeddings should be conjugate to each other for every element, i.e., $\forall 1 \leq m \leq d, \mathbf{r}_{1,m} = \overline{\mathbf{r}_{2,m}}$.
- **Composition**: A composition's representation can be naturally computed as $\mathbf{r_3} = \mathbf{r_1} \circ \mathbf{r_2}$.

[1]In the original paper, it was referred as anti-symmetric relation. But since anti-symmetric relation has to be reflexive, which is not satisfied here, it should be asymmetric relation instead.

The score function of a triple $(e_i, r_k, e_j)$ then can be defined as:

$$-\left\|\mathbf{e_i} \circ \mathbf{r_k} - \mathbf{e_j}\right\| \tag{3.9}$$

where the $p$-norm of a complex vector is defined as $\|v\|_p = \sqrt[p]{\sum_m |v_m|^p}$.

Note similar to TransE, RotatE is not able to handle reflexive, 1-to-N, N-to-1, and N-to-N relations.

#### 3.3.2.3 Summary

Geometric operation-based models are typically more intuitive and interpretable compared to other KGE methods as they model relations between entities as geometric transformations. However, they may face difficulties when modeling complex relations that do not have a clear geometric interpretation.

### 3.3.3 Bilinear KGE Models

Instead of explicitly designing geometric transformations for relations and thus deriving a distance-based score function, we can directly define the score function for a triple using the bilinear models:

$$f_{r_k}(e_i, e_j) = \mathbf{e}_i^\top \mathbf{M}_{r_k} \mathbf{e}_j$$

where $\mathbf{M}_{r_k}$ is a $d \times d$ matrix that models the specific pair-wise interactions between the dimensions in entity embedding space for the relation $r_k$. The bilinear model can also be considered as projecting $e_j$ via a relation-specific linear transformation $\mathbf{M_{r_k}}$ and then the score function is calculated as the inner product of $\mathbf{e_i}$ and the projected entity $\mathbf{M_{r_k}} \mathbf{e_j}$. Linear transformation subsumes different types of geometric operations, including rotation, reflection, scaling, and shear.

#### 3.3.3.1 RESCAL and its Variants

RESCAL [83] is the first work that proposed the bilinear models. A vector $\mathbf{e_i} \in \mathbb{R}^d$ is used to capture the latent semantics of an entity $e_i$ and a relation $r_k$ is represented as a matrix $\mathbf{M_{r_k}} \in \mathbb{R}^{d \times d}$

to model pairwise interactions between latent factors. The score function for a triple $(e_i, r_k, e_j)$ is then defined as:

$$f_{r_k}(e_i, e_j) = \mathbf{e_i}^\top \mathbf{M}_{r_k} \mathbf{e_j} = \sum_p \sum_q [\mathbf{M}_{r_k}]_{pq} [\mathbf{e}_i]_p [\mathbf{e}_j]_q$$

Due to the expressivity of the matrix $\mathbf{M}_{r_k}$, bilinear model can model relations that are reflexive, irreflexive, transitive, symmetric, asymmetric, 1-N, N-1, and N-N, making the model very flexible and powerful. The downside of the approach is also caused by the matrix representation of relations, which makes the computational complexity very high. Its variants are exploring a good trade-off between *expressivity* and *efficiency*.

DistMult [135], which is the most popular bilinear model, simplifies RESCAL by restricting $\mathbf{M}_{r_k}$ to diagonal matrices, where the score function can be computed as:

$$f_{r_k}(e_i, e_j) = \sum_p [\mathbf{r_k}]_p [\mathbf{e}_i]_p [\mathbf{e}_j]_p$$

In this case, it reduces the time complexity from quadratic to linear with respect to the dimensionality of the embedding vectors. However, it can no longer model asymmetric relations, as for any relation $r_k$ we always have:

$$f_{r_k}(e_i, e_j) = \sum_p [\mathbf{r_k}]_p [\mathbf{e}_i]_p [\mathbf{e}_j]_p = \sum_p [\mathbf{r_k}]_p [\mathbf{e}_j]_p [\mathbf{e}_i]_p = f_{r_k}(e_j, e_i)$$

To address the limitations of DistMult, ComplEx [122] extends DistMult by defining entities as well as relations in the complex vector space to model asymmetric relations, where the score function is defined as:

$$f_{r_k}(e_i, e_j) = \mathrm{Re}(\sum_p [\mathbf{r_k}]_p [\mathbf{e}_i]_p [\overline{\mathbf{e}_j}]_p)$$

where $\overline{\mathbf{e}_j}$ is the conjugate of $\mathbf{e}_j$. In this way, the asymmetric relations can receive different scores due to the difference between head and tail entities when switching the order of the two. Note ComplEx can still model symmetric relations when only real part exists in those matrices.

There are other variants of bilinear models that have been proposed, such as SimplE [60] and HolE [82].

#### 3.3.3.2 ANALOGY: Introducing Constraints to Relation Matrices

Motivated by analogical reasoning, instead of using arbitrary linear maps to represent relations, ANALOGY [70] considers a special family of matrices called *normal matrices* and introduce additional constraints to capture the relation commutativity. Normal matrices satisfy $\mathbf{M}_{r_k}\mathbf{M}_{r_k}^\intercal = \mathbf{M}_{r_k}^\intercal\mathbf{M}_{r_k}$, and representative normal matrices include:

- Symmetric matrices, i.e., $\mathbf{M}_{r_k} = \mathbf{M}_{r_k}^\intercal$ (diagonal matrices as a special case), which can be used to model symmetric relations. An example of symmetric matrix is: $\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$.
- Anti-symmetric matrices, i.e., $\mathbf{M}_{r_k} = -\mathbf{M}_{r_k}^\intercal$, which can be used to model asymmetric relations. An example of anti-symmetric matrix is: $\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$.
- Rotation matrices, i.e., $\mathbf{M}_{r_k}\mathbf{M}_{r_k}^\intercal = I$, which can be used to model 1-1 mapping relations. An example of rotation matrix is: $\begin{pmatrix} 1/2 & -\sqrt{3}/2 \\ \sqrt{3}/2 & 1/2 \end{pmatrix}$.
- Circulant matrices, where each row contains the same set of elements, and the next row is to shift the rightmost element in the previous row to the leftmost. It can be used to model asymmetric relations with much fewer parameters than a full matrix. The form of a circulant matrix is: $\begin{pmatrix} x_1 & x_d & \dots & x_2 \\ x_2 & x_1 & \dots & x_3 \\ \vdots & \vdots & \vdots & \vdots \\ x_d & x_{d-1} & \dots & x_1 \end{pmatrix}$.

To model the analogical structures, ANALOGY requires *commutativity* for the composition of two relations, which translates to the commutativity for the two corresponding matrices under multiplication operation: $\mathbf{M}_{r_k}\mathbf{M}_{r_{k'}} = \mathbf{M}_{r_{k'}}\mathbf{M}_{r_k}$. The normal matrix constraint and the commutativity constraint are then incorporated into the standard loss function, which makes it an optimization problem with constraints.

It has been shown that the previously introduced methods, such as DistMult, ComplEx, and HolE, can all be subsumed by ANALOGY as special cases in a principled manner.

#### 3.3.3.3 Summary

Bilinear models have higher expressive power compared to geometric models as they can capture complex interactions between entities through a bilinear function. However, they may be more challenging to interpret than geometric models.

### 3.3.4 Deep Representation Learning-based KGE Models

All the previous embedding models belong to the category of shallow embedding, where the learnable embeddings for entities and relations are directly used to feed into the score function (the decoder). With the development of different deep learning models, such as convolutional neural networks (CNNs) [66], recurrent neural networks (RNNs) [107], Transformers [124] and graph neural networks (GNNs) [62], more complicated encoders are designed to further transform the initial learnable embeddings to deeper representations. They can be categorized based on their underlying neural network architectures, including (1) *general neural networks-based models*, (2) *convolutional neural networks (CNNs)-based models*, (3) *sequence models*, and (4) *graph neural networks (GNNs)-based models*.

#### 3.3.4.1 General Neural Networks-based Models

The general idea of neural network-based KGE models is to use neural networks to define the score function, which takes learnable embedding vectors as input and the plausible score as the output. This introduces more complicated interactions, such as non-linear transformations, between entities and relations.

**Multi-Layer Perceptron (MLP)** Multi-Layer Perceptron (MLP) [37] is the most straightforward neural architecture. Given a triple $(e_i, r_k, e_j)$, the vector embeddings $\mathbf{e_i}$, $\mathbf{r_k}$, and $\mathbf{e_j}$ are concatenated in the input layer and mapped to a non-linear hidden layer. The score is computed by a linear output layer.

$$f_{r_k}(e_i, e_j) = \sigma(\mathbf{w}^\top \tanh(\mathbf{M}[\mathbf{e_i}; \mathbf{r_k}; \mathbf{e_j}]))$$

The mapping matrices $\mathbf{M} \in \mathbb{R}^{k\times(3d)}$ are responsible for projecting the embedding vectors, where the $3d$ term comes from concatenating the $d$-dimensional vectors $\mathbf{e_i}$, $\mathbf{r_k}$, and $\mathbf{e_j}$. The final layer weights are represented by $\mathbf{w} \in \mathbb{R}^{k\times 1}$.

**Neural Tensor Network (NTN)** Neural Tensor Network (NTN) [113] proposes another neural network architecture for KG embedding learning. For each triple $(e_i, r, e_j)$, NTN combines entities $\mathbf{e}_i, \mathbf{e}_j \in \mathbb{R}^d$ by a 3-d relation-specific tensor $\mathbf{W}_r^{[1:k]} \in \mathbb{R}^{d\times d\times k}$ via $\mathbf{e_i}^\intercal \mathbf{W}_r^{[1:k]} \mathbf{e_j} \in \mathbb{R}^k$, together with linear projections to entities. Then a relation-specific linear output layer gives the final score of the triple. NTN essentially can be regarded as a combination of MLPs and bilinear models:

$$f_r(e_i, e_j) = \mathbf{r}^\intercal \tanh(\mathbf{e_i}^\intercal \mathbf{W}_r^{[1:k]} \mathbf{e_j} + \mathbf{M}_r^1 \mathbf{e_i} + \mathbf{M}_r^2 \mathbf{e_j} + \mathbf{b}_r)$$

where $\mathbf{M}_r^1, \mathbf{M}_r^2 \in \mathbb{R}^{k\times d}$ denote relation-specific projection matrices, $\mathbf{b}_r \in \mathbb{R}^k$ denotes relation-specific bias vector, and $\mathbf{r} \in \mathbb{R}^k$ denotes the final linear weights.

#### 3.3.4.2 Convolutional Neural Networks (CNNs)-based Model

Some studies aim to leverage the Convolutional Neural Networks (CNNs) to achieve better expressivity and thus higher performance.

**ConvE** ConvE [35] makes the first attempt to use the CNN framework for KG embedding learning. ConvE reshapes the head entity embedding and relation embedding into 2D embeddings and stacks them as the input matrix for the 2D convolution layer. Then, ConvE vectorizes and projects the output feature map tensor through a linear transformation. The score of a triple is defined as matching the output of the linear transformation and the tail entity embedding by their inner product:

$$f_{r_k}(e_i, e_j) = f(\mathrm{vec}(f([\overline{\mathbf{e_i}}; \overline{\mathbf{r_k}}] * \omega))\mathbf{W})\mathbf{e_j}$$

where $\omega$ is the convolutional filters and $*$ denotes convolution operation, vec is the vectorization operation reshaping a tensor into a vector, $f$ denotes a non-linear function, and $\overline{\mathbf{e_i}}$ and $\overline{\mathbf{r_k}}$ denote a

2D reshaping of vectors $\mathbf{e_i}$ and $\mathbf{r_k}$, respectively[2].

There are several extensions to ConvE, including ConvKB [81], ConvR [58], and InteractE [123], to further enhance the performance.

#### 3.3.4.3 Sequence Models

Most KGE methods learn KG embeddings mainly based on triple-level observation, which lacks the capability of capturing long-range relational dependencies of entities. To address this issue, several studies are proposed to integrate sequence models for long-term relational dependencies learning.

**Recurrent Neural Networks (RNNs)** A conventional choice to use recurrent neural networks (RNNs) to model long-range relational dependencies. Recurrent skipping networks (RSNs) [46] is the most representative work that integrates RNNs with residual learning for KG embedding. It utilizes a random walk sampler to sample an entity-relation chain $(e_{i_1}, r_{k_1}, e_{i_2}, r_{k_2}, \ldots, e_{i_l}, r_{k_l}, e_{i_{l+1}})$ from the KG as input. For example, (*"Thomas Edison", IsMarriedTo, "Mina Miller", LiveIn, "USA"*) is an entity-relation chain in Fig. 3.4. The chain is abstracted into $(x_1, x_2, \ldots, x_n)$ by ignoring the entity and relation type. The recurrent hidden state for each element in the entity-relation chain is calculated as $\mathbf{h}_t = \tanh(\mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{W}_x\mathbf{x}_t + \mathbf{b})$ following typical RNN architecture, where $t$ denotes $t_{th}$ element in the sequence.

Note that the elements in an entity-relation chain do have two different types: entity and relation, which carry different semantic information. For example, if $x_t$ is a relation, then $(x_{t1}, x_t, x_{t+1})$ is a triple in KG, where $x_{t1}$ is the head entity and $x_{t+1}$ is the tail entity. Since the head entity is significant in predicting its tail entity, RSN proposes a skipping operation to shortcut the head entity to let it directly participate in predicting its tail entity, which is conducted as:

[2]If $\mathbf{e_i}, \mathbf{r_k} \in \mathbb{R}^d$, then $\overline{\mathbf{e_i}}, \overline{\mathbf{r_k}} \in \mathbb{R}^{d_w \times d_h}$, where $d = d_w d_h$.

$$\mathbf{h}'_t = \begin{cases} \mathbf{h}_t & \text{if } x_t \in \mathcal{E} \\ \mathbf{S}_1\mathbf{h}_t + \mathbf{S}_2\mathbf{x}_{t-1} & \text{if } x_t \in \mathcal{R} \end{cases}$$

where $\mathbf{h}'_t$ denotes the output hidden state of the skipping operation at time step $t$, and $\mathbf{h}_t$ denotes the corresponding RNN output. $\mathbf{S}_1$, $\mathbf{S}_2$ are the weight matrices that are shared at different time steps. In this way, both head entity ($x_{t-1}$) and relation ($x_t$) determine the latent representation of a relation element ($x_t$), which will be used to update the latent representation of the next tail entity ($x_{t+1}$).

**Transformer-based Models** Transformer-based models have been proven to be a more powerful architecture for sequence modeling, and have also been leveraged to embed KG entities and relations. CoKE [126] is one of such examples. Different from the entity-relation chain sequence used in RSN, the sequence used in CokE $(x_1, x_2, \ldots, x_n)$ is a path connecting two entities, where the first and last elements are entities and the others in between are relations, in the form of $(e_{i_1}, r_{k_1}, r_{k_2}, \ldots, r_{k_l}, e_{i_{l+1}})$. Each element has an initial representation $\mathbf{h}_i^0 = \mathbf{x}_i^{\text{ele}} + \mathbf{x}_i^{\text{pos}}$, where $\mathbf{x}_i^{\text{ele}}$ is the element embedding and $\mathbf{x}_i^{\text{pos}}$ is the position embedding. CoKE feeds the sequence into a stack of $L$ successive Transformer encoders to encode the sequence, where $h_i^l = \text{Transformer}(h_i^{l-1})$ for $l = 1, \ldots, L$. Motivated by the masked language model (MLM) [36], CoKE proposes an entity prediction task to train the model, i.e., to predict the masked entities given a sequence. Different from the general mask strategy, CoKE only masks entities in a sequence. For instance, given a sequence $(x_1, x_2, \ldots, x_n)$ where $x_1$ and $x_n$ are two entities, CoKE associates the sequence with two training instances $(?, x_2, \ldots, x_n)$ and $(x_1, x_2, \ldots, ?)$ to predict the missing entities.

#### 3.3.4.4 Graph Neural Networks (GNNs)-based Models.

Graph Neural Networks (GNNs) [132] are a class of deep learning models that are designed for graphs, which leverage message passing to aggregate information from neighbors of a node. Graph Convolutional Network (GCN) [62] is the most cited GNN, which contains two steps in each layer: (1) feature propagation from neighbors; and (2) feature transformation with a linear weight matrix followed by a non-linear activation. Naturally, KGs are a special case of a graph, which contain

different types of relations and can benefit from GNNs to capture the graph structure. One of the most popular GNN-based KGE model is R-GCN [106]. RGCN extends GCN to relation-aware. Specifically, RGCN associates each relation type with a relation-specific weight matrix for the transformation:

$$h_i^{(l+1)} = \sigma(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{|N_i^r|}W_r^{(l)}h_j^{(l)} + W_0^{(l)}h_i^{(l)})$$

where $h_i^l \in \mathbb{R}^{d^{(l)}}$ is the hidden state of the $i$-th entity in $l$-th layer, $\mathcal{N}_i^r$ is the set of neighbors of $i$-th entity within relation $r \in \mathcal{R}$, $W_r^{(l)}$ and $W_0^{(l)}$ are the weight matrices for relation $r$ and self-influence at layer $l$. To predict the relationship between two entities, R-GCN uses DistMult [135] as the decoder to score a triple.

#### 3.3.4.5 Summary

In general, deep learning-based models are powerful and can model complex patterns and dependencies in KGs. However, they typically demand a large amount of data to obtain good performance and are therefore computationally expensive. In addition, these models are less interpretable compared to earlier carefully designed transformations for relations.

### 3.3.5 Model Training

Once the KGE models have been constructed, meaning the score function is well defined, the model parameters can be learned following standard machine learning training. We now discuss the commonly used loss functions and typical negative sampling strategies used in KGE model training.

#### 3.3.5.1 Training Objective

Let $\mathcal{O}$ be the set of observed triples in KG, let $(e_i', r_k, e_j')$ denote a negative triple to the positive triple $(e_i, r_k, e_j)$, and let the set of negative triples corresponding to $(e_i, r_k, e_j)$ be $\mathcal{N}(e_i, r_k, e_j)$, the general idea of training objective is either (1) differentiate the positive triples from the negative

ones; or (2) rank the positive triples higher than the negative ones.

**Classification-based Loss** The simplest loss function is cross-entropy loss, which aims to distinguish positive triples from negative ones:

$$-\sum_{(e_i,r_k,e_j)\in O}\left(\log\sigma(f_{r_k}(e_i,e_j))+\sum_{\substack{(e'_i,r_k,e'_j)\\ \in\mathcal{N}(e_i,r_k,e_j)}}\frac{1}{|\mathcal{N}(e_i,r_k,e_j)|}\log\sigma(-f_{r_k}(e'_i,e'_j))\right) \tag{3.10}$$

For instance, the bilinear models such as DistMult [135] and ANALOGY [70] employ this loss for model training. An extension to cross-entropy loss is to add a margin term $\gamma$ to adjust the probability:

$$-\sum_{(e_i,r_k,e_j)\in O}\left(\log\sigma(f_{r_k}(e_i,e_j)+\gamma)+\sum_{\substack{(e'_i,r_k,e'_j)\\ \in\mathcal{N}(e_i,r_k,e_j)}}\frac{1}{|\mathcal{N}(e_i,r_k,e_j)|}\log\sigma(-f_{r_k}(e'_i,e'_j)-\gamma)\right) \tag{3.11}$$

For instance, RotatE [117] employs this loss for model training.

**Ranking-based Loss** The most frequently used loss function is *margin-based pairwise ranking loss*:

$$\sum_{(e_i,r_k,e_j)\in O}\sum_{(e'_i,r_k,e'_j)\in\mathcal{N}(e_i,r_k,e_j)}\max(0,\gamma-f_{r_k}(e_i,e_j)+f_{r_k}(e'_i,e'_j)) \tag{3.12}$$

where $\gamma$ is a hyperparameter denoting the margin. The basic idea of pairwise ranking loss is to make the scores of positive triples higher than their corresponding negative ones with at least $\gamma$; otherwise, it will receive a penalty. For instance, TransE and its variants employ this loss for model training.

#### 3.3.5.2 Creating Negative Triples

We can see that negative triples play a critical role in defining the training objective. But KGs contain only positive triples. Conventional embedding models follow *Closed World Assumption* (CWA) (i.e., all triples that are not contained in the knowledge graph are *False*) to construct negative triples. As shown in Eq. (3.13), the set of negative triples of a positive triple $(e_i, r_k, e_j)$, denoted

as $\mathcal{N}(e_i, r_k, e_j)$, is constructed by corrupting the head or tail entities of the positive triple.

$$\mathcal{N}(e_i, r_k, e_j) = \{(e_i', r_k, e_j) | (e_i', r_k, e_j) \notin O\} \cup \{(e_i, r_k, e_j') | (e_i, r_k, e_j') \notin O\} \tag{3.13}$$

Given the fact that $\mathcal{N}(e_i, r_k, e_j)$ usually contains thousands of negative triples, it is impractical to include all of them in the computation. To alleviate the computational complexity, only a small portion of negative triples will be randomly sampled. However, some of the randomly generated negative samples might be too easy to be discriminated against from positive triples and thus make a little contribution towards the training.

Some recent work has proposed to incorporate adversarial learning idea, such as Generative Adversarial Network (GAN) [44] for better negative sampling. For example, KBGAN [15] proposes to employ GAN on knowledge representation learning for high-quality negative triplets sampling. The discriminator is trained to minimize the margin-based ranking loss in Eq. (3.12) as in the previous models, where negative samples are from the triple generator: $(e_i', r_k, e_j') \sim p_G(e_i', r_k, e_j' | e_i, r_k, e_j)$. The generator learns to generate high-quality negative triplets that can fool the discriminator, i.e., maximize the expectation of scores for generated triples or minimize the following loss:

$$\sum_{(e_i, r_k, e_j) \in O} \mathbb{E}[-f_D(e_i', r_k, e_j')] \tag{3.14}$$

where $f_D(e_i', r_k, e_j)$ is the score function of the discriminator. It is equivalent to maximizing the loss for the best discriminator and thus fooling the discriminator. For example, given an observed triple *(Google, LocatedIn, Mountain View)*, it can be corrupted by replacing its tail entity, i.e., *(Google, LocatedIn, Apple)*. However, most of the corrupted triples are usually "too easy" to recognize. The generator generates the probability distribution over all candidate negative triples to select the "most confusing" triple. For instance, *(Google, LocatedIn, Phoenix)* is a very useful negative triple because it cannot be proved wrong without understanding the context. Then, the discriminator takes the generated negative triple *(Google, LocatedIn, Phoenix)* as well as the original triple *(Google, LocatedIn, Mountain View)* as the input and distinguish the generated one from the observed one to improve the embedding performance.

RotatE [117] also proposes self-adversarial negative sampling for generating negative samples,

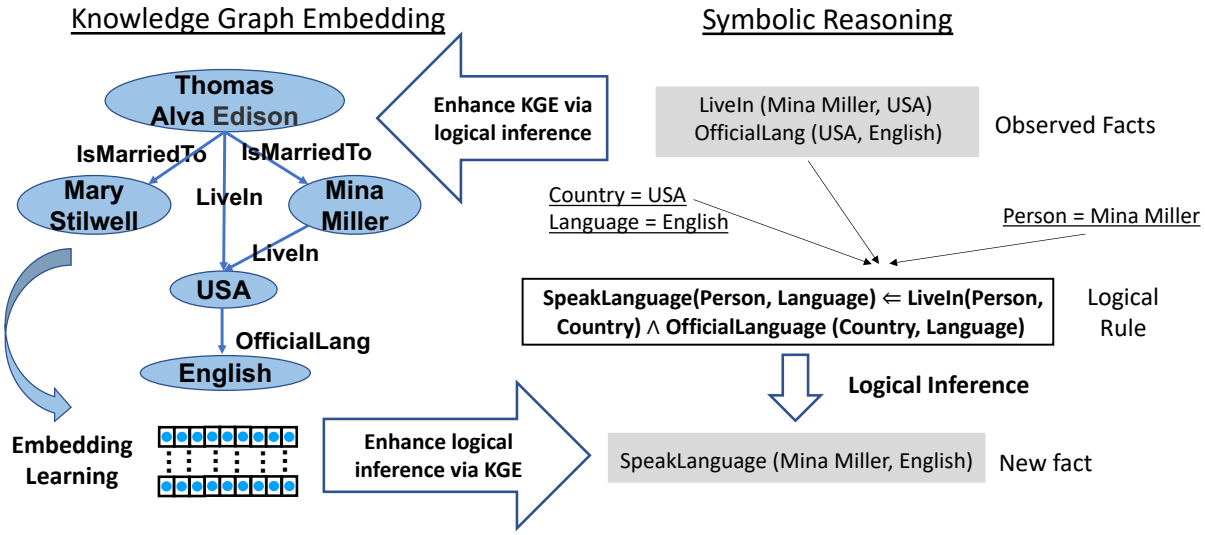



Figure 3.5: Neural-symbolic reasoning over a toy example of KG. **Left**: KG embedding model. **Right**: Traditional logical inference model. Integrating both approaches can lead to enhanced reasoning capabilities.

which samples negative triples according to the embedding model. Specifically, RotatE samples negative triples from the following distribution:

$$p((e'_i, r_k, e'_j)|(e_i, r_k, e_j)) = \frac{\exp\{\alpha f_{r_k}(e'_i, e'_j)\}}{\sum_{(e'_i, r_k, e'_j) \in \mathcal{N}(e_i, r_k, e_j)} \exp\{\alpha f_{r_k}(e'_i, e'_j)\}} \tag{3.15}$$

where $\alpha$ is the temperature of sampling. The probability $p((e'_i, r_k, e'_j)|(e_i, r_k, e_j))$ is taken as the weight of the negative sample. Therefore the final loss with self-adversarial training is given below:

$$-\sum_{(e_i, r_k, e_j) \in O} \left( \log \sigma(\gamma + f_{r_k}(e_i, e_j)) + \sum_{\substack{(e'_i, r_k, e'_j) \\ \in \mathcal{N}(e_i, r_k, e_j)}} p((e'_i, r_k, e'_j)) \log \sigma(-f_{r_k}(e'_i, e'_j) - \gamma) \right) \tag{3.16}$$

## 3.4 Neural-Symbolic Integration for KG Completion

Previous sections have introduced two main directions in solving the KG completion problem, i.e., traditional logical inference approaches in Section 3.2 and knowledge graph embedding (KGE) approaches in Section 3.3.

Traditional logical inference approaches attempt to implement reasoning via logic rules. They

have shown great interpretability and generalizability due to the power of symbolic representation. However, the discrete symbolic representation causes severe scalability issues and is unable to explore inherent correlations among entities and relations. Instead, KGE methods embed entities into continuous vector representations to encode the network structure of KGs. A score function can be learned by the neural network to make predictions of the missing knowledge. Although KGE methods have demonstrated their good scalability and strong ability in capturing correlation among atoms, most of them cannot model higher-order dependency of KG relations or take additional knowledge in terms of logical rules into consideration. As presented in Fig 3.5, by integrating both techniques into a unified framework, neural-symbolic reasoning provides a more efficient, generalizable, and interpretable way to perform KG reasoning.

In this chapter, we aim to introduce several studies that attempt to combine both approaches.Since the score function of KGE provides a soft truth value to triples in KG, *probabilistic programming frameworks* are widely used to combine the embedding model with pre-defined logic rules. Based on the types of probabilistic programming frameworks introduced in Section 3.2, existing neural-symbolic integration studies can be roughly divided into two categories: (1) *embedding-based variational distribution for variational inference of Markov Logic Network (MLN)*; and (2) *Fuzzy logic-based regularization to regularize embedding models*.

### 3.4.1 Embedding-based Variational Inference for Markov Logic Network (MLN)

Markov Logic Network (MLN) [100] combines probabilistic graphical model and FOL, which is able to define the joint probability for all the ground predicate variables. The knowledge graph completion task then becomes the inference problem in a probabilistic model. Unfortunately, constructing MLN involves the grounding for predicates and logical rules, and MLN inference subsumes probabilistic inference, which is #P-complete. Both the construction complexity and the inference complexity prevent the application of MLN in large-scale datasets.

Recently, variation inference [11] has been proposed to provide approximate solutions to the inference problem in a much more efficient way, where the key is to design a variational distribution to approximate the original posterior distribution. In the KG setting, KG embedding can be used to

define the probability for each triple $(e_i, r_k, e_j)$ to be *True*. Embedding-based distributions are then naturally proposed to serve as variational distributions to conduct variational inference of MLN. Earlier studies consider each triple independently with each other. Later studies use *graph neural network* as the KG embedding model to capture the dependency, which lead to more effective relational data modeling.

#### 3.4.1.1 pLogicNet: An Embedding-based Variational Inference Approach

pLogicNet [95] is the most representative embedding-based variational inference algorithm for MLN. According to MLN, the joint probability of all the predicate variables is $p_{\boldsymbol{w}}(\mathbf{v}_O, \mathbf{v}_H)$, where $\mathbf{v}_O$ denotes all the observed triples that take the value *True* and $\mathbf{v}_H$ denotes all the unseen triples that take either *True* or *False*. The KG completion task is to find the assignment $\mathbf{v}_H$ that maximizes the posterior probability $p_{\boldsymbol{w}}(\mathbf{v}_H|\mathbf{v}_O)$, i.e., to find the best truth values for the unobserved triples based on the observed ones. As discussed earlier, the inference problem is nontrivial to solve, and the approximation solutions are needed.

Variational inference is one of such approximation algorithms. The main idea is to find a simpler distribution denoted as $q_\theta(\mathbf{v}_H)$ that is close to the true posterior distribution $p_{\boldsymbol{w}}(\mathbf{v}_H|\mathbf{v}_O)$, where the distance of the two can be measured as the KL divergence between the two distributions: $D_{KL}(q_\theta(\mathbf{v}_H)||p_{\boldsymbol{w}}(\mathbf{v}_H|\mathbf{v}_O)) = \mathbb{E}_{q_\theta(\mathbf{v}_H)}(\log q_\theta(\mathbf{v}_H) - \log p_{\boldsymbol{w}}(\mathbf{v}_H|\mathbf{v}_O))$. KL-divergence is difficult to be minimized directly, as the true posterior distribution is unknown. Fortunately, we can maximize another term called Evidence Lower Bound (ELBO) denoted as $\mathcal{L}(q_\theta, p_{\boldsymbol{w}})$, which is defined as:

$$\mathcal{L}(q_\theta, p_{\boldsymbol{w}}) = \mathbb{E}_{q_\theta(\mathbf{v}_H)}[\log p_{\boldsymbol{w}}(\mathbf{v}_O, \mathbf{v}_H) - \log q_\theta(\mathbf{v}_H)] \tag{3.17}$$

It can be proven that KL-divergence of the distribution plus ELBO equals to the log of the marginal distribution $\log p_{\boldsymbol{w}}(\mathbf{v}_O)$, which is a constant with respect to the parameters in the variational distribution $q_\theta$, i.e., $D_{KL}(q_\theta(\mathbf{v}_H)||p_{\boldsymbol{w}}(\mathbf{v}_H|\mathbf{v}_O)) + \mathcal{L}(q_\theta, p_{\boldsymbol{w}}) = \log p_{\boldsymbol{w}}(\mathbf{v}_O)$. Therefore, minimizing the KL-divergence between the two distributions is equivalent to maximizing the ELBO.

In pLogicNet, the variational distribution $q_\theta(\mathbf{v}_H)$ is defined using a KGE model, by assuming the truth value of each triple independently follows a Bernoulli distribution, with parameters specified

by the embedding score function:

$$
\begin{aligned}
q_\theta(\mathbf{v}_H) &= \prod_{(e_i, r_k, e_j) \in H} q_\theta(r_k(e_i, e_j)) \\
&= \prod_{(e_i, r_k, e_j) \in H} f_{r_k}(r_k(e_i, e_j))
\end{aligned}
\tag{3.18}
$$

where $r_k(e_i, e_j) \sim \text{Ber}(f_{r_k}(e_i, e_j))$ follows the Bernoulli distribution and $f_{r_k}(e_i, e_j)$ is an embedding score denoting the probability of triple $(e_i, r_k, e_j)$ to be *True*. For example, in DistMult, $f_{r_k}(e_i, e_j)$ can be defined as $\sigma(\mathbf{e}_i^\top \text{diag}(\mathbf{r}_k)\mathbf{e}_j)$.

ELBO can be effectively optimized using variational EM algorithm [78], where (1) the variational E-step is corresponding to the inference problem when rule weights $\mathbf{w}$ are fixed and (2) the M-step can be considered as the learning for $\mathbf{w}$ when the assignments to $\mathbf{v}_H$ are fixed.

- In variational E-step, $p_{\boldsymbol{w}}$ is fixed and $q_\theta$ is updated to maximize ELBO. In pLogicNet, it is done by minimizing the KL divergence between $q_\theta(\mathbf{v}_H)$ and $p_{\boldsymbol{w}}(\mathbf{v}_H|\mathbf{v}_O)$ in an approximate way. Since exact inference is intractable, pLogicNet approximates the true posterior with a mean-field distribution. Alternative approaches such as stochastic variational inference can also be considered here [54].

- In M-step, $q_\theta$ is fixed and the weights of the rules $\boldsymbol{w}$ are updated to maximize the joint probability of both observed and hidden triples (i.e., $\mathbb{E}_{q_\theta(\mathbf{v}_H)}[\log p_{\mathbf{w}}(\mathbf{v}_O, \mathbf{v}_H)]$).

During training, pLogicNet iteratively performs the E-step and the M-step until convergence.

It is important to note that in real-world KG, there are numerous hidden triples (i.e., $|\mathcal{E}| \times |\mathcal{R}| \times |\mathcal{E}| \setminus |O|$). Optimizing all of them is not feasible due to computational constraints. To alleviate this burden, a smaller subset of hidden triples is sampled to construct the hidden set $\mathcal{H}$. One sampling approach is to select unobserved triples $(e_i, r_k, e_j)$ that serve as the head of a ground rule *body* $\rightarrow$ *head*, where the rule body is fully observed.

Other extensions to pLogicNet have been proposed to utilize Graph Neural Networks (GNNs) as the KG embedding model. Examples of such extensions include ExpressGNN [144] and pGAT [52].

#### 3.4.1.2 Summary

While embedding-based variational inference for MLN offers an elegant solution to integrating embedding models and logical rules, inference efficiency remains a significant challenge. Optimizing over all the hidden triples is impractical, so approaches in this category only sample a small subset of hidden triples to reduce computational complexity. However, this reduction in computational complexity also leads to information loss from the logical rule side.

### 3.4.2 Fuzzy Logic-based Regularization

Fuzzy logic provides an alternative approach to Markov logic networks (MLNs). Instead of constructing Markov networks based on KG and ground logical rules, PSL uses logical rules as additional constraints and forms regularization terms to the original KGE loss.

This idea is closely connected to probabilistic soft logic discussed in Sec. 3.2.3.2. Each predicate-based atom is associated with a soft truth value in the range of $[0, 1]$, which can be defined via embeddings. Each ground logical rule is corresponding to a logical formula, and the truth value can be evaluated based on fuzzy logic. Intuitively, the embeddings should be consistent with the logical rules, leading to higher truth values for the corresponding ground rules.

#### 3.4.2.1 Regularizing over Ground Rules

The bridge between the embedding-based approach and fuzzy logic is the soft truth value assignment to each ground predicate: $I(r_k(e_i, e_j))$. From embedding perspective, we can turn the score function for each triple $f_{r_k}(e_i, e_j)$ into the range $[0, 1]$. From fuzzy logic perspective, once we have $I(r_k(e_i, e_j))$ for each ground atomic formula, the truth value for any rule $f$ can be evaluated based on fuzzy logic introduce in Sec. 2.2.4. The goal is to find the embeddings that also give high truth values for the given rules.

We start by introducing the methods which add regularization to the entity and relation representations by instantiating the FOL rules. A typical integration is defined as follows: (1) mapping each related triple (i.e., ground predicate) into a soft truth value (i.e., $I(r_k(e_i, e_j))$); (2) sampling

ground logical rules given the template logical rules; (3) computing satisfaction of each ground rule based on the soft truth value of ground predicates in it; and (4) defining proper loss based on the satisfaction of all the ground rules.

**The first attempt** Rocktäschel [102] is among the first integration attempts in this line. Embedding-wise, instead of learning entity embeddings for the individual entity, they utilize matrix factorization to learn joint embeddings of pairs of entities $\mathbf{v}_{e_i,e_j}$ as well as embeddings of relations $\mathbf{v}_{r_k}$. For each triple $(e_i, r_k, e_j)$, the score function is defined as $\sigma(\mathbf{v}_{r_k} \cdot \mathbf{v}_{e_i,e_j})$, where $\sigma(\cdot)$ denotes the sigmoid function.

From embedding perspective, the likelihood of a possible world given embeddings $\mathbf{v}_{e_i,e_j}$ and $\mathbf{v}_{r_k}$ can be defined as:

$$\prod_{r_k(e_i,e_j)\in\mathbf{v}_O} I(r_k(e_i,e_j)) \prod_{r_k(e_i,e_j)\in\mathbf{v}_H} (1 - I(r_k(e_i,e_j))) \tag{3.19}$$

By maximizing the likelihood of all observed triples in KG with $\ell_2$ regularization [26], the method learns entity-pair and relation embeddings that reconstruct observed facts that are able to generalize to missing facts. This objective is equivalent to to the classification-based loss in KGE (Eq. 3.10).

To inject logical rules $\mathcal{F}$ into the embeddings of relations and entity pairs, the logistic loss is defined as the negation of logarithmic transformation of the likelihood that the formulas in $\mathcal{F}$ are true under the model:

$$-\sum_{f\in\mathcal{F}} \log(I(f)) \tag{3.20}$$

where the soft truth value of a ground rule $f \in \mathcal{F}$, $I(f)$, is computed following the *product logic*. The integration of the two losses can hence be written as:

$$-\sum_{r_k(e_i,e_j)\in\mathbf{v}_O} \log(I(r_k(e_i,e_j))) - \sum_{r_k(e_i,e_j)\in\mathbf{v}_H} \log(1 - I(r_k(e_i,e_j))) - \sum_{f} \log(I(f)) \tag{3.21}$$

**KALE** Compared to the first attempt, KALE [47] proposes to (1) use a different score function to assign truth values to each triple; and (2) introduce negative ground rules in addition to the positive ground rules.

KALE uses TransE-based score function for the truth value, i.e., $I(r_k(e_i, e_j)) = 1 - \frac{1}{3\sqrt{d}} \left\| \mathbf{e}_i + \mathbf{r}_k - \mathbf{e}_j \right\|_1$, where $\mathbf{e}_i$, $\mathbf{r}_k$, and $\mathbf{e}_j$ are embedding vectors to the corresponding entities and relations and $d$ is the dimensionality of the embeddings.

Then, a set of positive and negative ground rules (i.e., $f^+$ and $f^-$) are sampled given the template logical rules. For instance, consider a positive ground rule from Fig. 3.5 as shown below:

$$f^+ : \text{liveIn(Edison, USA)} \leftarrow \text{isMarriedTo(Edison, Miller)} \wedge \text{liveIn(Miller, USA)}$$

A negative ground rule can be constructed by replacing the relation in the conclusion of the positive ground rule with a random relation $r \in \mathcal{R}$. For example:

$$f^- : \text{bornIn(Edison, USA)} \leftarrow \text{isMarriedTo(Edison, Miller)} \wedge \text{liveIn(Miller, USA)}$$

Similar to Rocktäschel [102], the truth value for rules is computed based on product logic, denoted as $I(f^+)$ and $I(f^-)$.

In addition to the sample rules, the positive and negative triples are also sampled similarly to traditional KGE approaches, which are atomic formulas. Both categories are unified under the concept of *logic formulas*, and are denoted as $f^+$ and $f^-$.

KALE defines a loss function over all sampled formulas following margin-based classification loss, with the goal to differentiate positive formulas from negative ones:

$$\sum_{f^+} \sum_{f^- \in \mathcal{N}_{f^+}} [\gamma - I(f^+) + I(f^-)]_+ \tag{3.22}$$

where $\mathcal{N}_{f^+}$ denotes a set of negative rules constructed for the positive rule $f^+$, $[x]_+$ denotes $max(x, 0)$, and $\gamma$ denotes the margin. Note that, by removing ground rules from the whole set of sampled formulas, the loss degenerates to regular TransE-based embedding loss.

**UKGE: Modeling Probabilistic KG** Most of the existing methods focus on modeling deterministic KG while UKGE [20] extends existing methods so as to model probabilistic KG, where each triple is associated with a weight $c_{r_k(e_i,e_j)}$ indicating the confidence score of the triple to be *True*. UKGE employs two different mapping functions to transform the KGE score function to a

confidence value in the range of $[0, 1]$. Formally, let a transformation function $\phi(\cdot)$ denote the mapping from $f_{r_k}(e_i, e_j)$ to $I(r_k(e_i, e_j))$, the truth value of the triple is then:

$$I(r_k(e_i, e_j)) = \phi(f_{r_k}(e_i, e_j)), \phi : \mathbb{R}^+ \cup \{0\} \rightarrow [0, 1] \tag{3.23}$$

where $\phi$ can be defined as a logistic function:

$$\phi(x) = \frac{1}{1 + e^{-(\mathbf{w}x+\mathbf{b})}} \tag{3.24}$$

or a bounded rectifier:

$$\phi(x) = \min(\max(\mathbf{w}x + \mathbf{b}, 0), 1) \tag{3.25}$$

Specifically, DistMult score function is used in UKGE, i.e., $f_{r_k}(e_i, e_j) = \mathbf{r}_k \cdot (\mathbf{e}_i \circ \mathbf{e}_j)$.

To summarize, the interpretation of triples in KG can be defined as:

$$I(r_k(e_i, e_j)) = \begin{cases} c_{r_k(e_i,e_j)} & \text{if } r_k(e_i, e_j) \in \mathbf{v}_O \\ \phi(f_{r_k}(e_i, e_j)) & \text{if } r_k(e_i, e_j) \in \mathbf{v}_H \end{cases} \tag{3.26}$$

For each unseen triple, if it is covered by some logical rule, we can sample a set of ground rules with the unseen triple as the rule head. According to probabilistic soft logic introduced in Sec. 3.2.3.2, the distance to satisfaction of a rule is $d(f) = 1 - I(f)$, where $I(f)$ is computed based on Lukasiewicz logic. We wish the embedding-based interpretation for unseen triples could lead to a small distance to satisfaction for the sampled rules.

The goal of UKGE contains two parts: (1) to minimize the MSE between the ground truth weight $c_{r_k(e_i,e_j)}$ and the predicted soft truth value $I(r_k(e_i, e_j))$ of observed triples; (2) to minimize the weighted distance to satisfaction of sampled ground rules, in which unseen triple $r_k(e_i, e_j)$ is the rule head. The final loss can be written as:

$$\sum_{r_k(e_i,e_j)\in\mathbf{v}_O} |I(r_k(e_i, e_j)) - c_{r_k(e_i,e_j)}|^2 + \sum_{r_k(e_i,e_j)\in\mathbf{v}_H} \sum_{f\in\Gamma_{r_k(e_i,e_j)}} |w_f d(f)|^2 \tag{3.27}$$

where $\Gamma_{r_k(e_i,e_j)}$ is a set of ground rules with $r_k(e_i, e_j)$ as the head, and $w_f$ is the given weight associated with each rule.

#### 3.4.2.2 Regularizing over Template Rules

The above methods regularize relation and entity representations using the grounding of first-order logic rules. Nonetheless, scaling the grounding of first-order logic rules becomes challenging for large-scale KGs. To improve the scalability and generalizibility, several methods are proposed to add regularization at the template rule level. FSL [33] is one of the most representative models, which is an extension to Rocktäschel [102]. For any rule $f : r_h(x, y) \leftarrow r_b(x, y)$, such as $employeeAt(x, y) \leftarrow professorAt(x, y)$, the truth assignment for the head must be higher than or equal to the the body to make the rule satisfied, namely, for any $e_i, e_j$, $I(r_h(e_i, e_j)) \geq I(r_b(e_i, e_j))$. This can be easily checked for Boolean logic via truth table as well as three types of fuzzy logic mentioned in 2.2.4. A simple discussion has been provided for Lukasiewicz logic in probabilistic soft logic (Sec. 3.2.3.2). We will see this again when discussing logical laws in Sec. **??**.

Let $\mathbf{v}_{r_b}$ and $\mathbf{v}_{r_h}$ be the learnable embeddings of the rule body and rule head, enforcing the implication rule to be true requires $\forall \mathbf{v}_{e_i,e_j} : \mathbf{v}_{r_b}{}^\intercal \mathbf{v}_{e_i,e_j} \leq \mathbf{v}_{r_h}{}^\intercal \mathbf{v}_{e_i,e_j}$, where $\mathbf{v}_r^\intercal \mathbf{v}_{e_i,e_j}$ is the score function for triple $(e_i, r, e_j)$ used in FSL. The constraint can be turned into a (soft) loss, following Bayesian Personalized Ranking (BPR) loss [99] used in recommender systems:

$$- \sum_{(e_i,*,e_j)\in \mathbf{v}_O} \log \sigma(-[\mathbf{v}_{r_b} - \mathbf{v}_{r_h}]^\intercal \widetilde{\mathbf{v}}_{e_i,e_j}) \tag{3.28}$$

where $\widetilde{\mathbf{v}}_{e_i,e_j} = \mathbf{v}_{e_i,e_j} / \left\|\mathbf{v}_{e_i,e_j}\right\|_1$, and $(e_i, *, e_j) \in \mathbf{v}_O$ denotes all the entity pairs that are observed in KG.

But how can we drop the entities so that there is no need to go through all the entity pairs? FSL smartly leverages Jensen's inequality to generate an upper bound of the above loss:

$$\begin{aligned} & - \sum_{(e_i,*,e_j)\in \mathbf{v}_O} \log \sigma(-[\mathbf{v}_{r_b} - \mathbf{v}_{r_h}]^\intercal \widetilde{\mathbf{v}}_{e_i,e_j}) \\ \leq & - \sum_{(e_i,*,e_j)\in \mathbf{v}_O} \sum_d (\widetilde{\mathbf{v}}_{e_i,e_j})_d \log \sigma(-[\mathbf{v}_{r_b} - \mathbf{v}_{r_h}]_d) \\ \leq & - \sum_d \log \sigma(-[\mathbf{v}_{r_b} - \mathbf{v}_{r_h}]_d) \sum_{(e_i,*,e_j)\in \mathbf{v}_O} (\widetilde{\mathbf{v}}_{e_i,e_j})_d \end{aligned} \tag{3.29}$$

where $d$ is used to the denote the $d_{\text{th}}$ dimension of the corresponding vector. Let $\beta = \sum_{(e_i,*,e_j)\in \mathbf{v}_O}$ be

the upper bound of $(\widetilde{\mathbf{v}}_{e_i,e_j})_i$, say $|\mathbf{v}_O|$, the upper bound can be written as: $-\beta \sum_d \log \sigma(-[\mathbf{v}_{r_b} - \mathbf{v}_{r_h}]_d)$, which is called *lifted loss* and has dropped all the concrete triples successfully.

Combining the lifted loss with the matrix factorization model for KG embedding learning, the overall loss can hence be written as

$$-\sum_{\substack{r_k(e_i,e_j)\in\mathbf{v}_O,\\ r_k(e_i',e_j')\in\mathbf{v}_H}} \log\sigma(-\mathbf{v}_{r_k}^{\intercal}[\mathbf{v}_{e_i',e_j'} - \mathbf{v}_{e_i,e_j}]) - \beta \sum_{(r_b,r_h)\in\mathcal{F}} \sum_d \log\sigma(-[\mathbf{v}_{r_b} - \mathbf{v}_{r_h}]_d) \tag{3.30}$$

where the first term learns the KG embedding such that positive triples receive higher scores than negative ones, while the second term denotes a lifted loss for every rule in a set $\mathcal{F}$ of implication rules.

#### 3.4.2.3 Regularizing over Unseen Triples

Different from the methods which define loss over the logical rules, RUGE [48] directly defines a loss function over triples. It employs the score function in ComplEx [122]

$$\sigma(\mathrm{Re}(\langle \mathbf{e}_i, \mathbf{r}_k, \bar{\mathbf{e}}_j \rangle)) \tag{3.31}$$

to model triples. Triples are divided into two categories, the observed triples (i.e., $\mathbf{v}_O$) and the hidden triples (i.e., $\mathbf{v}_H$). Observed triples have hard labels $y_{r_k(e_i,e_j)} = 1$ whereas sampled negative triples used to have labels $y_{r_k(e_i,e_j)} = 0$. To inject logical rules, RUGE uses product logic to predict the soft label of hidden triples $s_{r_k(e_i,e_j)} \in [0,1]$ in the sampled rule instances by maximizing the soft truth value of the sampled rules. With the ground truth labels $y_{r_k(e_i,e_j)}$ of the observed triples and soft label $s_{r_k(e_i,e_j)}$ of the hidden triples, RUGE learns embedding by enforcing triples to be consistent with their labels.

$$\frac{1}{|O|} \sum_{r_k(e_i,e_j)\in\mathbf{v}_O} l(I(r_k(e_i,e_j)), y_{r_k(e_i,e_j)}) + \frac{1}{|H|} \sum_{r_k(e_i,e_j)\in\mathbf{v}_H} l(I(r_k(e_i,e_j)), s_{r_k(e_i,e_j)}) \tag{3.32}$$

where $l(x,y) = y \log x (1y) \log(1x)$ is the cross entropy.

#### 3.4.2.4 Summary

The logical rule-based regularization methods typically separate the loss into two parts: the embedding-based loss and the logic-based loss. Although logical rule-based regularization approaches provide an efficient and effective way to combine embedding and logical rules, they also suffer from information loss from the logical rule side, as most of them sample only a small portion of ground rules to approximate the inference process. In addition, most methods in this category make only a one-time injection of logical rules to enhance embedding, ignoring the interactive nature between embedding learning and logical inference [47, 102]

## 3.5 UniKER: A Recent Advance of Neural-Symbolic Integration for KG Completion

Despite several attempts made to combine KGE and logical rules for KG reasoning, they either use a probabilistic model to approximate the exact logical inference (i.e., MAX-SAT) [95, 144, 52] (i.e., embedding-based variational inference of MLN) or simply treat logical rules as additional constraints into KGE loss [47, 102, 33] (i.e., logical rule-based regularization approaches). Moreover, these methods rely on ground rules, the total number of which can be intractable in practice as logical rules with $n$ variables can result in $|\mathcal{E}|^n$ possible combinations of entities for grounding the rule. To tackle the scalability issue, only a small portion of ground predicates/ground rules are sampled to approximate the inference process, which causes further *information loss from the logic side*.

To overcome the above issues, we introduce a state-of-the-art **Uni**fied framework for combining **K**nowledge graph **E**mbedding with logical **R**ules (**UniKER**) [24], to handle a special type of first-order logic, i.e., the definite Horn rules. First, UniKER combines logical rule reasoning and KG embedding in an iterative manner, to make sure the inferred knowledge via both techniques can benefit each other as shown in Fig. 3.6. Second, UniKER proposes an iterative grounding algorithm to extend the classic forward chaining algorithm that is designed for definite Horn rule reasoning in an extremely efficient way. Consequently, UniKER can fully exploit the knowledge

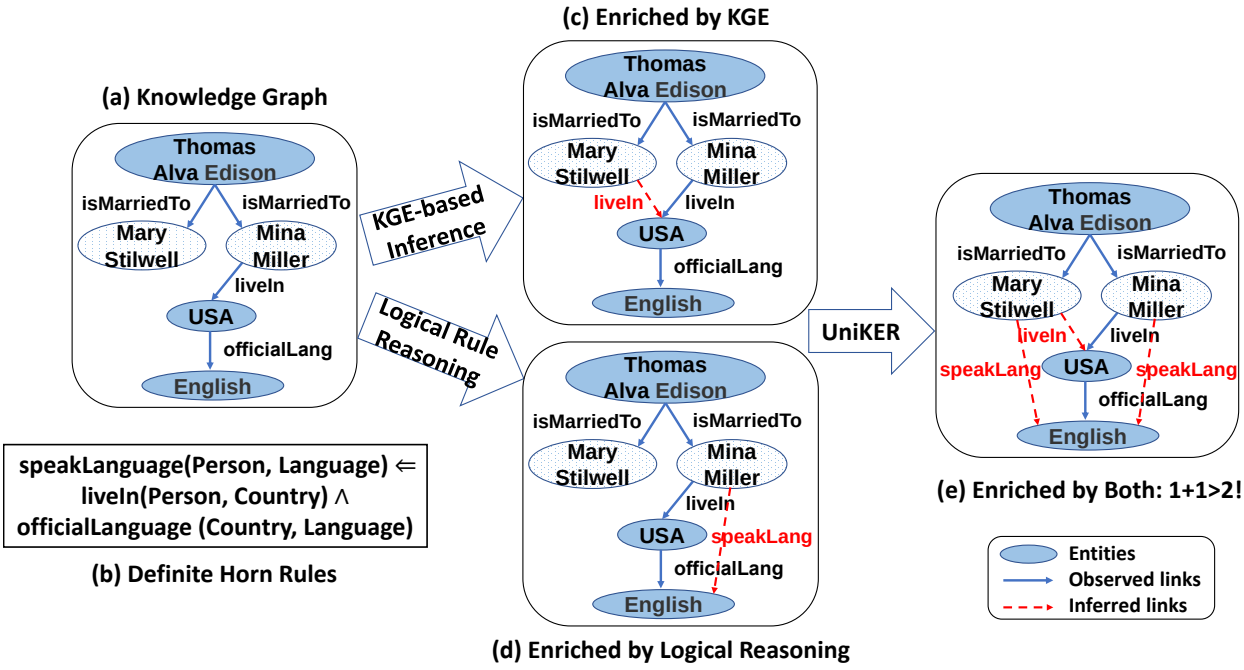



Figure 3.6: Given (a) a KG with observed facts and (b) a set of definite Horn rules, different inference results can be obtained using (c) KGE, (d) logical inference, and (e) the proposed UniKER approach. The synergy of KGE and logical reasoning via UniKER is more powerful than a simple union of (c) and (d).

contained in logical rules and enrich KGs for better embedding. Meanwhile, KGE enhances the forward chaining by including more potentially useful hidden facts (See Fig. 3.6(c)). In this way, the two procedures mutually enhance each other.

### 3.5.1 Framework of UniKER

Rather than follow probabilistic logic to integrate logical rules and KGE, UniKER shows that by leveraging the nice properties of definite Horn rules, there is a much simpler way to directly derive an optimal boolean solution for MAX-SAT problem. Definite Horn rules have been introduced previously in Sec. 2.3. Their nice properties and two efficient inference algorithms, forward chaining and backward chaining, have been discussed in Sec. 3.2.2.1. To capture the mutual interaction between KGE and logical inference, UniKER proposes an iterative mechanism, which

ensures that it is more potent than a simple union of KGE and logical rules.

#### 3.5.1.1 Enhance KG via Forward Chaining-based Logical Reasoning for KGE

Note that there exists a truth assignment that satisfies all ground rules when restricting logical rules to be definite Horn rules [55]. The question is how to conduct such an assignment efficiently. Let $\mathbf{v}_H^{T^*}$ (hidden triples that are true) and $\mathbf{v}_H^{F^*}$ (hidden triples that are false) denote the satisfying truth assignment, i.e., $\mathbf{v}_H^{T^*} = \{r_k(e_i, e_j) = 1 \mid r_k(e_i, e_j) \in \mathbf{v}_H\}$ and $\mathbf{v}_H^{F^*} = \{r_k(e_i, e_j) = 0 \mid r_k(e_i, e_j) \in \mathbf{v}_H\}$. An existing algorithm called **forward chaining** [104] can derive $\mathbf{v}_H^{T^*}$ efficiently, which has been discussed previously in Section 3.2.2.1.

Starting from known facts (e.g., *liveIn (Mina Miller, USA)*, *officialLanguage (USA, English)*), the forward chaining algorithm triggers all ground rules whose premises are satisfied (e.g., *speakLanguage (Mina Miller, English)* ← *liveIn (Mina Miller, USA)* ∧ *officialLanguage (USA, English)*), and adds their conclusion (e.g., *speakLanguage (Mina Miller, English)*) to the known facts until no facts can be added anymore.

As illustrated in Fig. 3.2, unlike other neural-symbolic integration algorithms, which require all ground predicates (including both observed and unobserved ground predicates) into the calculation, forward chaining adopts "lazy inference" instead. It involves only a small subset of "active" ground predicates/rules, and activates more if necessary as the inference proceeds. The mechanism dramatically improves inference efficiency by avoiding the computation for massive ground predicates/rules that are never used. Moreover, considering that definite Horn rules which can be extracted efficiently via modern rule mining systems are usually chain-like Horn rules. The conjunctive body of a ground chain-like Horn rules is essentially a path in a KG, which can be extracted efficiently using sparse matrix multiplication. To illustrate, let us consider the logical rule $r_0(x, y) \leftarrow r_1(x, z) \wedge r_2(z, y)$. We can represent relations $r_1$ and $r_2$ as sparse matrices $\mathbf{M}_1$ and $\mathbf{M}_2$, respectively. In these matrices, each row and column represent an entity in the KG, and the non-zero entries represent the existence of a relation between two entities. To find all the entity pairs $(x, y)$ that satisfy the body of the rule (i.e., $r_1(x, z) \wedge r_2(z, y)$), we perform sparse matrix multiplication between $\mathbf{M}_1$ and $\mathbf{M}_2$, which give us new facts in the form of $(x, r_0, y)$.

The newly generated facts denoted as $\mathbf{v}_H^{T^*}$ then will be added to KG. Since $\mathbf{v}_H^{T^*}$ and $\mathbf{v}_H^{F^*}$ are satisfying truth assignments derived by forward chaining, the knowledge contained in definite Horn rules is guaranteed to be fully exploited.

#### 3.5.1.2 Enhance KG via Embedding-based Inference for Logical Reasoning

Although forward chaining can find the satisfying truth assignment for all hidden triples efficiently, its reasoning ability is severely limited by the coverage of rules, the incompleteness of KGs, and the errors/noise contained in KGs. Considering its strong reasoning ability and robustness, KGE models are not only useful to (1) prepare a more complete KG by adding useful hidden triples but also helpful to (2) eliminate incorrect triples in both KGs and inferred results.

**Update the KGE model** Once the KG has been updated by logical reasoning, UniKER updates KGE by (1) treating both observed triples $\mathcal{O}$ and the newly inferred triples $\mathbf{v}_H^{T^*}$ as positive triples and (2) treating $\mathbf{v}_H^{F^*}$ as negative triples to form the objective function of KGE model: Following the margin-based cross entropy loss discussed in Sec. 3.3.5, the new KGE loss can be defined as:

$$-\sum_{\substack{(e_i,r_k,e_j)\\ \in\{\mathcal{O}\cup\mathbf{v}_H^{T^*}\}}}\left(\log\sigma(\gamma+f_{r_k}(e_i,e_j))+\sum_{\substack{(e_i',r_k,e_j')\\ \in\mathcal{N}(e_i,r_k,e_j)}}\frac{\log\sigma(-f_{r_k}(e_i',e_j')-\gamma)}{|\mathcal{N}(e_i,r_k,e_j)|}\right) \tag{3.33}$$

where $\mathbf{v}_H^{T^*}$ denotes the inferred triples derived by forward chaining, $(e_i', r_k, e_j')$ denotes their corresponding negative samples, and $\gamma$ is a margin to separate them. The score $f_{r_k}(e_i, e_j)$ of a triple $(e_i, r_k, e_j)$ can be calculated following any score functions of KGE models. To reduce the effects of randomness, UniKER samples multiple negative triples for each positive sample, which is denoted as $\mathcal{N}(e_i, r_k, e_j)$. To ensure *True* but unseen triples not be sampled, the selection of $\mathcal{N}(e_i, r_k, e_j)$ is restricted to $\mathbf{v}_H^{F^*}$.

**Update the KG** Now let us discuss how to use the embeddings to efficiently and effectively update the KG, by adding high-quality new triples and removing noisy triples.

**(1) Including Potential Useful Hidden Triples ($\Delta$+).** A straightforward solution to adding hidden true triples would be computing the score for *every* hidden triple and adding the most

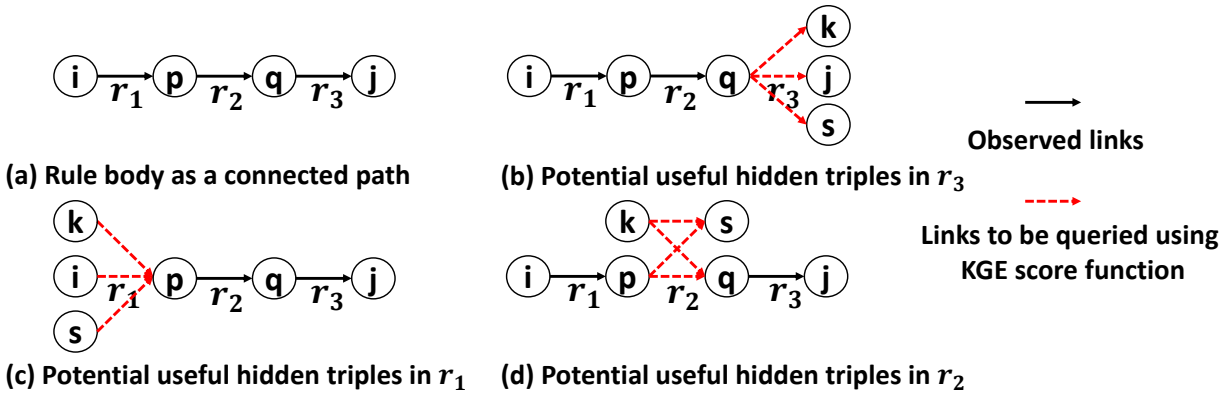



Figure 3.7: Illustration of potential useful hidden triples by taking Eq.(3.34) as an example.

promising ones with the highest scores to the KG. Unfortunately, the number of hidden triples is quadratic to the number of entities (i.e. $O(|\mathcal{R}||\mathcal{E}|^2)$), thus it is too expensive to compute scores for all of them. Instead, UniKER adopts the "lazy inference" strategy to select only a small subset of "potentially useful" triples.

Considering the ground rule below:

$$r_0(e_i, e_j) \leftarrow r_1(e_i, e_p) \wedge r_2(e_p, e_q) \wedge r_3(e_q, e_j), \tag{3.34}$$

if $r_1(e_i, e_p) \in \mathbf{v}_O$, $r_3(e_q, e_j) \in \mathbf{v}_O$, and $r_2(e_p, e_q) \in \mathbf{v}_H$, we would not be able to infer the head $r_0(e_i, e_j)$ as whether $r_2(e_p, e_q)$ is *True* or not is unknown. Thus, $r_2(e_p, e_q)$ becomes the crux to determine the truth value of the head, which is called "potentially useful". In general, given a ground rule whose body includes only one unobserved ground predicate, this unobserved ground predicate can be regarded as a "potentially useful" triple. The set of all "potentially useful" triples is denoted as $\Delta_+$. According to their positions, "potentially useful" triples can be divided into two categories: (1) triples that are the first or the last predicate in a ground rule; and (2) triples that are not in the two ends of the rule (neither the first nor the last).

UniKER identifies both types of "potentially useful" triples as illustrated in Fig. 3.7 by taking the above logical rule (Eq. 3.34) with length 3 as an example. Let every relation $r_k$ in KG associate with an $|E| \times |E|$ matrix $\mathbf{M}^{(k)}$, in which the element $\mathbf{M}^{(k)}_{ij} = 1$ if the triple $(e_i, r_k, e_j) \in O$, and 0 otherwise. The algorithms to identify both types of "potential useful" triples are sketched as

follows.

- When the "potential useful" triple is the first or the last predicate in a ground rule, other observed triples in a chain-like definite Horn rule still constitute a complete path, which can be extracted efficiently by sparse matrix multiplication. Take Fig. 3.7 (c) as an example, to identify the "potential useful" triple $r_1(e_i, e_p)$, we have to first extract all connected path $r_2(e_p, e_q) \wedge r_3(e_q, e_j)$ by calculating $\mathbf{M} = \mathbf{M}^{(2)}\mathbf{M}^{(3)}$, where $\mathbf{M}^{(2)}$ and $\mathbf{M}^{(3)}$ are adjacency matrices corresponding to relations $r_2$ and $r_3$. Each nonzero element $\mathbf{M}_{pj}$ indicates a connected path between $e_p$ and $e_j$. We denote all indexes correspond to nonzero rows in $\mathbf{M}$ as $\delta = \{p|(\sum_j \mathbf{M}_{pj}) \neq 0\}$, which indicates that there is always a connected path starting at $p$. For specific $p \in \delta$, $\Delta_p = \{(e_i, r_1, e_p)|e_i \in E\}$ defines a set "potential useful" triples. If $(e_i, r_1, e_p)$ in $\Delta_p$ is predicted to be true via KGE, the head predicates $r_0(e_i, e_j)$ can be inferred.

- Otherwise, the path corresponds to the conjunctive body of the ground rule get broken into two paths by the "potential useful" triple, which we have to extract separately. As shown in Fig. 3.7 (d), when identifying "potential useful" triple $r_2(e_p, e_q) \in v_H$, two paths to be extracted are essentially two single relations $r_1$ and $r_3$, whose corresponding matrices are $\mathbf{M}^{(1)}$ and $\mathbf{M}^{(3)}$, respectively. We denote all indexes correspond to nonzero columns in $\mathbf{M}^{(1)}$ as $\delta_1 = \{p|(\sum_i \mathbf{M}^{(1)}_{ip}) \neq 0\}$ and all indexes correspond to nonzero rows in $\mathbf{M}^{(3)}$ as $\delta_2 = \{q|(\sum_j \mathbf{M}^{(3)}_{qj}) \neq 0\}$. $\Delta_{12} = \{(e_p, r_2, e_q)|p \in \delta_1, q \in \delta_2\}$ defines a set "potential useful" triples. If $(e_p, r_2, e_q)$ in $\Delta_{12}$ is predicted to be true via KGE, the head predicates $\{r_0(e_i, e_j)|\mathbf{M}^{(1)}_{ip} \neq 0, \mathbf{M}^{(3)}_{qj} \neq 0\}$ can be inferred.

The score $f_{r_k}(e_i, e_j)$ will be computed by KGE model to predict whether a "potentially useful" triple is *True*. If $f_{r_k}(e_i, e_j)$ is larger than the given threshold $\Psi$, the triple is classified as *True*. Otherwise, the triple is classified as *False*.

**(2) Excluding Potential Incorrect Triples ($\Delta-$).** In addition, due to their symbolic nature, logical rules cannot handle noisy data as well. If the KGs contain any error, based on incorrect observations, forward chaining will not be able to make the correct inference. Even worse, it may contribute to the propagation of the errors by including incorrectly inferred triples into KGs.

Therefore, a clean KG is significant for logical inference. Since KGE models show great power in capturing the network structure of KGs, incorrect triples usually result in contradictions and get lower prediction scores in KGE models compared to correct ones. Therefore, the score $f_{r_k}(e_i, e_j)$ computed by KGE model is able to measure the reliability of triple $(e_i, r_k, e_j)$ in $\mathcal{O} \cup \mathbf{V}_H^{T\,*}$. The bottom $\theta\%$ triples with the lowest prediction scores are denoted as $\Delta_-$. It will be excluded from $\mathcal{O} \cup \mathbf{V}_H^{T\,*}$ to alleviate the impact of noise.

**Algorithm 1:** Learning Procedure of UniKER

**Input:** Observed facts in knowledge bases $O$; threshold to eliminate noise $\theta\%$; threshold to include useful hidden triples $\Psi$; a set of definite Horn rules $\mathcal{F}$

**Output:** KG embeddings

```
for t = 1 : MAX_ITER do
    // Update KG via Logical Reasoning
    Derive v_H^{T t*} from O and update O ← O ∪ v_H^{T t*}
    // Update KG via Embedding-based Inference
    KG embedding learning based on O;
    Compute Δ_ and update O ← O − Δ_;
    Compute Δ_+ and update O ← O ∪ Δ_+;
end
```

#### 3.5.1.3 Integrating Embedding and Logical Rule Reasoning in an Iterative Manner

Since logical rules and KGE can mutually enhance each other, UniKER proposes a unified framework to integrate KGE and definite Horn rules-based inference iteratively. The pseudo-code of UniKER can be found in Algorithm 1. MAX_ITER is the user specified max iterations to run the algorithm, which highly depends on the given KG. UniKER usually sets MAX_ITER as 2 to 4. For each iteration of UniKER, it is comprised of two steps. First, it conducts logical reasoning to update KG. Following forward chaining algorithm, by triggering all rules whose premises are satisfied, UniKER derives entailed triple set $\mathbf{v}_H^{T\,t*}$ at $t$-th iteration, which is a subset of $\mathbf{v}_H^{T\,*}$ ($\mathbf{v}_H^{T\,*} = \cup_{t=1}^{+\infty} \mathbf{v}_H^{T\,t*}$).

Then, the newly inferred triples $\mathbf{v}_H^{T\,t*}$ are added to KG by updating $\mathcal{O} = \mathcal{O} \cup \mathbf{v}_H^{T\,t*}$. Second, UniKER focuses on embedding-based inference to update KG. KGE is learned based on the updated KG after the first step. With the learned embeddings, $\Delta_-$, which is the bottom $\theta\%$ triples with lowest prediction scores, are eliminated from $\mathcal{O}$. Meanwhile, $\Delta_+$, which are potentially useful hidden triples, are added to $\mathcal{O}$.

### 3.5.2 Connection to Existing Approaches

We categorize all existing methods according to two aspects: (1) whether they capture mutual interaction between KGE and logical inference; and (2) whether they conduct exact logical inference. The summary is given in Table 3.3. For the first aspect, both embedding-based variational inference to MLN methods and UniKER provide the interaction between embedding and logical inference while most fuzzy logic-based regularization approaches make only a one-time injection of logical rules to enhance embedding. For the second aspect, both embedding-based variational inference to MLN methods and fuzzy logic-based regularization methods follow the framework of fuzzy logic to combine logical rule and KGE, which can only approximate the optimal solution of MAX-SAT problem. UniKER is the first to use forward chaining to conduct exact inference, which provides an optimal solution to the original MAX-SAT problem. The detailed connection between UniKER and both categories of methods is discussed below.

**Comparison to Embedding-based Variational Inference to MLN.** The general objective of embedding-based variational inference for MLN can be written as:

$$\mathcal{L}_{\text{ELBO}}(q_\theta, p_w) + \lambda \mathcal{L}_{\text{KGE}}(q_\theta) \tag{3.35}$$

where the variational distribution $q_\theta$ is defined using a KGE model and $p_w$ is the true posterior defined over MLN. $\mathcal{L}_{\text{KGE}}(q_\theta)$ denotes the loss of the base KGE model. By optimizing $\mathcal{L}_{\text{ELBO}}(q_\theta, p_w)$, the KL divergence between $q_\theta$ and $p_w$ can be minimized. In this way, the knowledge contained in rules can be transferred into the embeddings. Due to the nature of the approximate solution provided by variational inference and the information loss caused by the sampling procedure, $q_\theta$ can only approximate the optimum of the MAX-SAT problem, and no guarantees are provided on the quality

Table 3.3: Comparison of different neural-symbolic methods for KG completion.

| Categories | Methods | Interactive | Exact Logical Inference |
|---|---|---|---|
| Embedding-based | pLogicNet [95] | ✓ | × |
| Variational | ExpressGNN [144] | ✓ | × |
| Inference to MLN | pGAT [52] | ✓ | × |
| Fuzzy logic-based Regularization | KALE [47] | × | × |
| | RUGE [48] | ✓ | × |
| | Rocktäschel et al. [102] | × | × |
| UniKER | | ✓ | ✓ |

of the solutions obtained. Instead of guiding the learning of the embedding model via variational inference, UniKER directly solves the MAX-SAT problem and use the derived knowledge $\mathbf{v}_H^{T^*}$ to train the embedding model, which leads to superior reasoning.

**Comparison to Fuzzy Logic-based Regularization Approaches.** The general objective of fuzzy logic-based regularization approaches can be written as:

$$\mathcal{L}_{\text{KGE}} + \lambda \mathcal{L}_{\text{Logic}} \tag{3.36}$$

where $\mathcal{L}_{\text{KGE}}$ denotes the loss of the base KGE model while $\mathcal{L}_{\text{Logic}}$ corresponds to the satisfaction loss of the sampled ground rules. When we follow Lukasiewicz logic to define the satisfaction loss and incorporate all ground rules into the calculation of $\mathcal{L}_{\text{Logic}}$, $\mathcal{L}_{\text{Logic}}$ becomes a convex program that reasons over a relaxed version of the MAX-SAT problem, as previously discussed in Section 3.2.3.2, which only approximates the exact logical inference. Instead, UniKER solves the MAX-SAT problem exactly using forward chaining, thanks to the definite Horn rules. The better exploitation of logical rules lead to better KG and thus more powerful KGE and thus logical reasoning itself. Moreover, $\mathcal{L}_{\text{Logic}}$ makes only a one-time injection of logical rules to enhance embedding, where logical reasoning will not be further enhanced even after the KGE gets improved. On the contrary, UniKER is able to capture the interactive nature between embedding and logical

inference.

**Theoretical Computational Complexity Analysis of UniKER.** To theoretically demonstrate the superiority of our proposed UniKER in terms of efficiency, we compare the space and time complexity of UniKER and other methods that combine KG embedding and logical rules. More precisely, we focus solely on logical rule-based regularization approaches as the complexity of embedding-based variational inference for MLN can vary widely and is heavily dependent on the number of iterations needed to achieve convergence. Obviously, they have a much higher computational cost than UniKER does. As both logical rule-based regularization approaches and UniKER consists of two parts, grounding logical rules and embedding learning, we include the complexity of both parts in Table 3.4. Note that grounding only contributes to the time complexity without affecting space complexity. Let $n_e$ denote the number of entities, $n_r$ denote the number of relations, and $n_t$ denote the number of observed triples; additionally, let $l$ represent the length of the rule body, $n_l$ denote the number of template rules, and $\theta$ represent the sampling ratio; finally, let $a$ denote the average degree of entities and $d$ represent the dimension of the embedding space. We can see that: (1) For space complexity, UniKER is the same as other logical rule-based regularization approaches; (2) For time complexity, considering $a \ll n_e$, if the sampling ratio is not small enough, UniKER is much smaller than other logical rule-based regularization approaches.

### 3.5.3 Experiments

#### 3.5.3.1 Experimental Setup

**Datasets.** UniKER conducts experiments on three real-world KGs (i.e., Family [34], FB15k-237 [13] and WN18RR [13]). The input logical rules are generated by AMIE+ [41] automatically. The detailed statistics of these KGs are provided in Table 3.5. FB15K237 and WN18RR are the most widely used large-scale benchmark datasets for KGE models, which do not suffer from test triple leakage in the training set. The Family dataset is selected due to better interpretability and high intuitiveness.

**Compared Methods.** UniKER is evaluated against SOTA algorithms, including (1) basic KGE

Table 3.4: Comparison of space and time complexity for model training.

| Method | | Space Complexity | Time Complexity | |
|---|---|---|---|---|
| | | | Grounding | Embedding |
| KGE | TransE [13] | $O(n_e d + n_r d)$ | - | $O(n_t d)$ |
| | Dismult [135] | $O(n_e d + n_r d)$ | - | $O(n_t d)$ |
| Logical Rule-based Regularization | KALE [47] | $O(n_e d + n_r d)$ | $O(\theta n_l {n_e}^{l+1})$ | $O(n_t d + \theta n_l {n_e}^{l+1} d)$ |
| | Rocktäschel et al. [102] | $O(n_e d + n_r d + n_e n_r)$ | $O(n_l n_e a^l)$ | $O(n_e n_r d + n_e d^2 + n_r d^2 + n_l n_e a^l d)$ |
| | RUGE [48] | $O(n_e d + n_r d)$ | $O(\theta n_l {n_e}^{l+1})$ | $O(n_t d + \theta n_l {n_e}^{l+1} d)$ |
| Our Model | UniKER | $O(n_e d + n_r d)$ | $O(n_l n_e a^l)$ | $O(n_t d + n_l n_e a^l d)$ |

Table 3.5: Data Statistics.

| Dataset | Type | #Entity | #Relation | #Triple | #Rule |
|---|---|---|---|---|---|
| Family | Family network | 3007 | 12 | 28356 | 41 |
| FB15k-237 | Freebase knowledge | 14541 | 237 | 310116 | 300 |
| WN18RR | Lexical network | 40943 | 11 | 93003 | 11 |

models (e.g., RESCAL [83], SimplE [60], HypER [5], TuckER [6], TransE [13], DistMult [135] and RotatE [117]), (2) traditional logical rule-based methods (e.g., MLN [100] and BLP [32]), (3) two classes of approaches combining embedding model with logical rules (two representative methods KALE [47] and RUGE [48] for PSL-based regularization approaches, and pLogicNet [95], ExpressGNN [144] and pGAT [52] for embedding-based variational inference of MLN), and (4) other approaches to combining embedding model with logical rules (e.g., BoxE [1]). To show that UniKER can be easily adapted to various KGE models, TransE [13], DistMult [135], and RotatE [117] are chosen as the base models for UniKER.

**Evaluation Metrics.** To compare the reasoning ability of UniKER and the aforementioned baseline algorithms, UniKER masks the head or tail entity of each test triple, and requires each method to predict the masked entity. During the evaluation, UniKER uses the filtered setting [13][3] and three evaluation metrics, i.e., Hit@1, Hit@10, and MRR. Hit@k measures the proportion of test triples for which the correct head or tail entity is ranked among the top $k$ predictions made by the model. For example, Hit@1 measures whether the correct answer is ranked at the very top of the list of predicted answers, while Hit@10 measures whether the correct answer is ranked within the top 10 predicted answers. MRR measures the average reciprocal rank of the correct entity among all the ranked entities for each test triple, where the reciprocal rank is 1/rank.

The dataset is partitioned randomly into a training set and a test set with a 8:2 ratio. In order to show the reasoning also benefit from KGE, any triples that can be directly inferred from the training set via logical rule reasoning are deliberately left out of the test set.To fairly compare all baseline methods, this same setting is consistently applied to all of them. We compare different algorithms on the KG inference task.

[3]Although test triples and training triples have no overlap, it is still possible that a test triple and a training triple share head and relation (or relation and tail). It is possible that the model predicts the entity where the entire triple is seen in the training, which should be correct but unfortunately will be considered as false as it is not included in the test. In the filtered setting, any entity that appears in the training set is removed from the candidate entities to rank to avoid this issue.

Table 3.6: Effectiveness on KG completion task

| Model | Family | | | FB15k-237 | | | WN18RR | | |
|---|---|---|---|---|---|---|---|---|---|
| | Hit@1 | Hit@10 | MRR | Hit@1 | Hit@10 | MRR | Hit@1 | Hit@10 | MRR |
| RESCAL | 0.489 | 0.894 | 0.639 | 0.108 | 0.322 | 0.179 | 0.123 | 0.239 | 0.162 |
| SimplE | 0.335 | 0.888 | 0.528 | 0.150 | 0.443 | 0.249 | 0.290 | 0.351 | 0.311 |
| HypER† | 0.364 | 0.903 | 0.551 | 0.252 | 0.520 | 0.341 | 0.436 | 0.522 | 0.465 |
| TuckER† | 0.373 | 0.898 | 0.567 | 0.266 | 0.544 | 0.358 | **0.443** | 0.526 | 0.470 |
| BLP† | - | - | - | 0.062 | 0.150 | 0.092 | 0.187 | 0.358 | 0.254 |
| MLN | 0.655 | 0.732 | 0.694 | 0.067 | 0.160 | 0.098 | 0.191 | 0.361 | 0.259 |
| Forward Chaining | **0.919** | 0.919 | **0.919** | **0.586** | **0.586** | **0.586** | 0.323 | 0.323 | 0.323 |
| KALE | 0.433 | 0.869 | 0.598 | 0.131 | 0.424 | 0.230 | 0.032 | 0.353 | 0.172 |
| RUGE | 0.495 | 0.962 | 0.677 | 0.098 | 0.376 | 0.191 | 0.251 | 0.327 | 0.280 |
| ExpressGNN | 0.105 | 0.282 | 0.164 | 0.150 | 0.317 | 0.207 | 0.036 | 0.093 | 0.054 |
| pLogicNet | 0.683 | 0.874 | 0.768 | 0.261 | 0.567 | 0.364 | 0.301 | 0.410 | 0.340 |
| pGAT† | - | - | - | 0.377 | 0.609 | 0.457 | 0.395 | 0.578 | 0.459 |
| BoxE† | - | - | - | - | 0.538 | 0.337 | - | 0.541 | 0.451 |
| TransE | 0.221 | 0.874 | 0.453 | 0.231 | 0.527 | 0.330 | 0.007 | 0.406 | 0.165 |
| UniKER-TransE | 0.873 | **0.971** | 0.916 | 0.463 | **0.630** | 0.522 | 0.040 | 0.561 | 0.307 |
| DistMult | 0.360 | 0.885 | 0.543 | 0.220 | 0.486 | 0.308 | 0.304 | 0.409 | 0.338 |
| UniKER-DistMult | 0.770 | 0.945 | 0.823 | **0.507** | 0.587 | 0.533 | 0.432 | 0.538 | 0.485 |
| RotatE | 0.787 | 0.933 | 0.862 | 0.237 | 0.526 | 0.334 | 0.421 | 0.563 | 0.469 |
| UniKER-RotatE | **0.886** | **0.971** | **0.924** | 0.495 | 0.612 | **0.539** | 0.437 | **0.580** | **0.492** |

† Results on FB15k-237 and WN18RR are taken from the original papers.

#### 3.5.3.2 Results on KG Completion Task

To compare different algorithms on the KG completion task, we mask the head or tail entity of each test triple and require each method to predict the masked entity. The results of BLP, MLN, and pLogicNet are taken from the original papers that are summarized in [95]. As the results are only reported on the FB15k-237 and WN18RR datasets for these baselines, we compare UniKER with them on these two datasets.

Table 3.6 shows the comparison results, from which we find that: (1) UniKER outperforms KGE models in most cases with significant performance gain, which is due to the utilization of additional knowledge from logical rules; (2) UniKER also achieves better performance than existing two classes of approaches that combine the embedding model with logical rules as it provides an exact optimal solution to a satisfiable problem defined over all ground rules rather than employing sampling strategies to do approximation.

#### 3.5.3.3 Efficiency Analysis

Besides the promising results on KG reasoning, our UniKER is superior in terms of efficiency. We have theoretically analyzed the computational complexity of UniKER in Section 3.5.2. Learning procedure for UniKER consists of two components, (1) finding the optimal truth assignment problem defined over all ground Horn rules using forward chaining algorithm and (2) optimize the embedding model according to the optimal truth assignment. The efficiency of forward chaining highly depends on KG datasets. Thus, we provide a further investigation of the scalability of forward chaining on real-world datasets in this section. Note that forward chaining learns the optimal truth assignment for the satisfiable problem iteratively, the number of iterations required to achieve the optimal solution may influence its scalability. To compare the scalability of forward chaining against a number of SOTA (State-Of-The-Art) inference algorithms for MLN, three small-scale datasets (e.g., RC1000, sub-YAGO3-10, and sub-Family) are included in our experiments due to the high time complexity of the MLN inference algorithms. The RC1000 dataset is a commonly used benchmark dataset for inference in MLN, while sub-Family and sub-YAGO3-10 are subsets

Table 3.7: Efficiency Analysis on Family Dataset.

| Model | Time per Epoch | #Epochs for Convergence |
|---|---|---|
| KALE | > 1000 s | 500 |
| RUGE | > 1000 s | 800 |
| ExpressGNN | 168 s | 200 |
| pLogicNet | 7.2 s | 600 |
| UniKER-TransE | 6.5 s | 400 |

of the Family and YAGO3-10 datasets, respectively. We first conduct two experiments on the datasets. (1) As presented in Fig. 3.8, we record the proportion of inferred triples accumulated in every iteration over all inferred triples. The result shows that forward chaining can achieve the optimal solution within 12 iterations and infer over 70% correct triples within only 4 iterations. (2) We evaluate the scalability of forward chaining against a number of SOTA inference algorithms for MLN (e.g., MCMC [17], MC-SAT [91], BP [140], liftedBP [109] and Tuffy [85]). Forward chaining runs $100 - 100,000$ times faster than them. Some widely used algorithms MCMC and MC-SAT cannot even handle RC1000 dataset, which indicates the scalability of UniKER.

Then, we compare the overall run time of our proposed UniKER with other methods. As shown in Table 3.7, UniKER, which is able to conduct exact logical inference, is still faster than other neural-symbolic methods.

#### 3.5.3.4 Mutual Enhancement between KGE and Logical Inference

We have further investigated the potential benefits brought by the mutual interaction between KGE and logical inference.

- **Enhancement of Logical Inference via KGE.** On one hand, high-quality embedding learned by KGE models is useful to prepare more complete KGs via including useful hidden triples, which the performance of logical inference highly depends on. To show the benefit brought by KGE over logical inference, we evaluate UniKER-TransE against forward chaining on

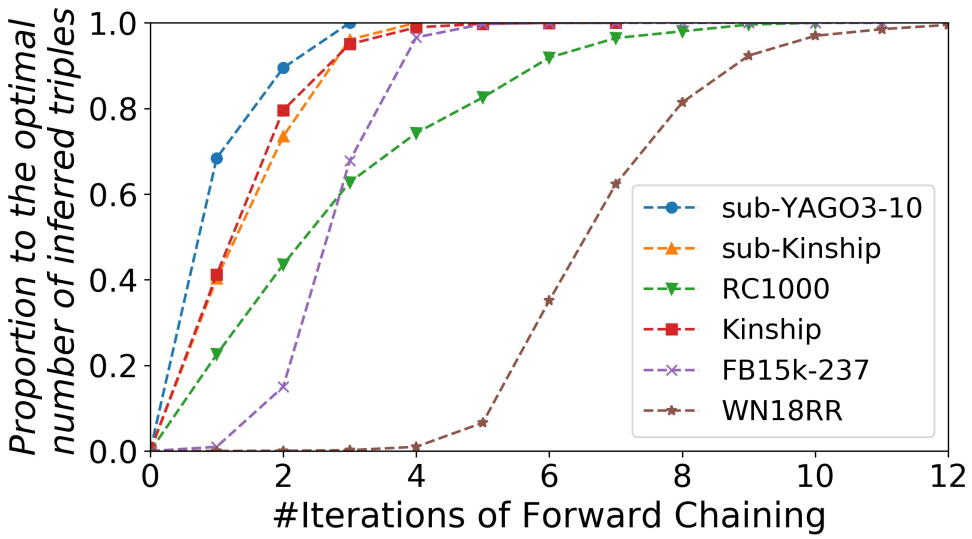



Figure 3.8: Proportion to the optimal number of inferred triples w.r.t. #iterations for efficiency analysis of Forward Chaining.

Table 3.8: UniKER-TransE vs. Forward Chaining on Family dataset (whose test set only retains triples that can be inferred by logical rules) on triple *True*/*False* classification task.

| Model | Precision | Recall | F1 |
|---|---|---|---|
| Forward Chaining | **1.000** | 0.919 | 0.958 |
| UniKER-TransE | 0.991 | **0.955** | **0.973** |

Family Dataset with the triple classification task, which aims to predict correct facts in the testing data. In order to create a testing set for classification, we randomly corrupt relations of correct testing triplets for negative triples construction. It results in a total of $2 \times$ #Test triplets with an equal number of positive and negative examples. During the evaluation, we adopt three evaluation metrics, i.e., precision, recall, and F1. As shown in Table 3.8, we can observe that although the precision slightly decreases, UniKER outperforms forward chaining with significant performance gain in terms of recall and F1, which validates the enhancement brought by the KGE model over logical inference.

- **Enhancement of KGE via Logical Inference.** On the other hand, logical rules are useful

Table 3.9: Results of reasoning of UniKER-TransE v.s. TransE on Family dataset (whose test set eliminates triples that can derive from logical rules).

| Model | Hit@1 | Hit@3 | Hit@10 | MRR |
|---|---|---|---|---|
| TransE | 0.267 | 0.651 | 0.803 | 0.476 |
| UniKER-TransE | **0.710** | **0.866** | **0.904** | **0.816** |

to gather more reliable triples for KGE by exploiting the symbolic compositionality of KG relations, which leads to the enhancement of the reasoning ability of the KGE model. To investigate the added value brought by logical rules over KGE, we evaluate UniKER-TransE against TransE on the Family dataset on the KG completion task. As some triples in the test dataset can be directly derived from logical rules, to ensure the improvement comes from the reasoning ability enhancement of the KGE model, we exclude the triples derived directly from rules from the test data. As presented in Table 3.9, we can observe that UniKER-TransE outperforms the TransE model with huge performance gain, especially in terms of Hit@1, which can be ascribed to the added value brought by logical rules over KGE.

## 3.6 Summary and Discussions

Knowledge graph (KG) completion refers to the task of predicting missing or incomplete information in a knowledge graph. Symbolic methods and neural methods are two broad categories of approaches used for KG completion.

Symbolic methods rely on logical reasoning and rule-based approaches to infer new facts based on existing ones. One of the main advantages of symbolic methods is that they can be easily interpreted and validated by human experts. However, symbolic methods have limited scalability and they struggle to capture complex patterns in the data, as logical rules may not be able to cover all possible cases.

On the other hand, neural methods use neural networks to learn representations of entities and

relations in the KG and use these representations to predict missing links. Neural methods have the advantage of being able to learn complex and non-linear patterns in the data. However, they are also difficult to interpret and validate and may require large amounts of data and computational resources.

Neural-symbolic methods aim to combine the strengths of symbolic and neural methods by integrating logical reasoning and representation learning. These methods typically use neural networks to learn representations of entities and relations and use logical rules or constraints to guide the learning process. This allows neural-symbolic methods to leverage the interpretability of symbolic methods, while also benefiting from the learning capabilities of neural methods.

However, neural-symbolic methods are still an early area of research, and their effectiveness for KG completion is still being explored. For example, although there have been attempts to integrate neural and symbolic methods using probabilistic programming frameworks, scalability issues have limited their effectiveness. To overcome these limitations, the UniKER algorithm has been introduced as a cutting-edge method that successfully combines symbolic reasoning and representation learning to tackle KG completion tasks.

We will only be able to cover the main approaches for KG completion due to space constraints. It is worth noting that there are other promising directions in this area, which are discussed below.

**Path-based KG Completion** Path-based methods concentrate on discovering paths between entities in a KG and use the existence or absence of specific paths to predict missing links. The reasoning paths found by path-based KG completion methods inherently provide an interpretable basis for their predictions. Representative works in this domain can be broadly categorized into traditional *path ranking algorithm* and recent *reinforcement learning-based methods*.

- **Path Ranking Algorithm**: The Path Ranking Algorithm is an early approach that employs random walks to generate paths between entities. It then ranks these paths as features for a binary classifier to predict missing links in KG. Although the method is simple, it relies on random walks to generate paths between entities, which can be computationally intensive, particularly for large-scale KGs with millions of nodes and edges. Notable methods in

this category include Path Ranking Algorithm (PRA) [64], and its extensions such as Path-RNN [79] and Chains of reasoning [30].

- **Reinforcement Learning Based Methods**: Since enumerating all possible relation paths is impractical, reinforcement learning-based methods have been proposed to avoid enumerating all possible relation paths in the KB by searching within a small neighborhood around the query entity. The most representative methods in this category include: DeepPath [134], MINERVA [29], MWalk [108] and MultiHop [68]

# CHAPTER 4

# Logical Rule Learning

## 4.1 Overview

Logical rules are widely used to represent domain knowledge and hypothesis, which is fundamental to symbolic reasoning-based human intelligence [2]. Very recently, it has been demonstrated that integrating logical rules into regular learning tasks can further enhance learning performance in a label-efficient manner, such as KG completion tasks discussed in Chapter 3. Despite the promising reasoning ability of symbolic methods, they require the set of logical rules to be specified by hand, which are usually *labor-intensive*. To reduce the demand on human efforts, *automatically* learning logical rules from examples in KGs becomes critical [56, 31].

The learning of logical rules from data can lead to the discovery of novel knowledge that was previously unknown. This is particularly significant for tasks where domain knowledge is incomplete. Furthermore, logical rules can facilitate the *generalization* of knowledge from specific examples or scenarios, enabling AI systems to apply learned knowledge to novel situations. This ability to generalize is crucial for transfer learning, which enables AI systems to adapt and reuse knowledge from one domain to another. Additionally, learning logical rules allows for knowledge representation in a format that is easily understandable by humans. Logical rules are often presented as "if-then" statements, which are easy to comprehend, interpret, and justify. This interpretability is valuable for decision-making processes and fosters trust in AI systems.

Formally, logical rule learning aims to learn logical rules in the following form:

$$r_h(x_h, y_h) \leftarrow r_{b_1}(x_{b_1}, y_{b_1}) \wedge \cdots \wedge r_{b_l}(x_{b_l}, y_{b_l}) \tag{4.1}$$

where the predicate $r_h(x_h, y_h)$ is called *rule head* (or *conclusion*), and $r_{b_1}(x_{b_1}, y_{b_1}) \wedge \cdots \wedge$

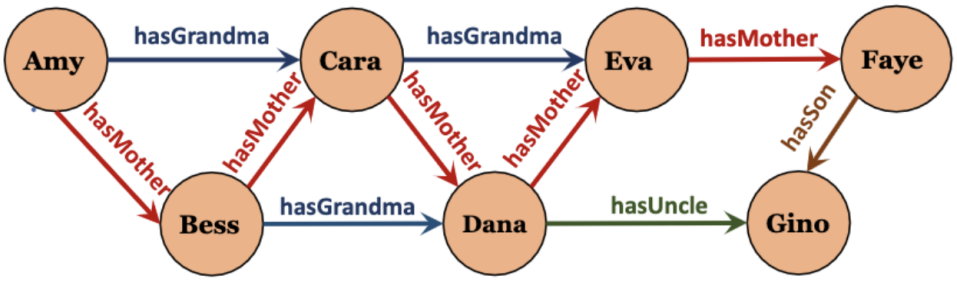



Figure 4.1: An instance of a kinship knowledge graph is depicted, where various relationships are denoted by arrows of distinct colors.

$r_{b_l}(x_{b_l}, y_{b_l})$ is called *rule body* (or *premise*). Combining rule head and rule body, we denote a logical rule as $(Head, Body)$. The length of a rule is defined as $l$, which is the number of predicates appearing in its body. For a more comprehensive overview of logical rules, please refer to Sec. 2.3. An example of rule learning is shown in Fig. 4.1, where two rules can be extracted from the observed KG:

$$\begin{aligned} \delta_1 &:= \text{hasGrandma}(x, y) \leftarrow \text{hasMother}(x, z) \wedge \text{hasMother}(z, y) \\ \delta_2 &:= \text{hasUncle}(x, y) \leftarrow \text{hasMother}(x, z_1) \wedge \text{hasMother}(z_1, z_2) \wedge \text{hasSon}(z_2, y) \end{aligned} \tag{4.2}$$

The fundamental idea of the logical rule learning methods is to assign a plausibility score $s(Head, Body)$ to each rule $(Head, Body)$ within a designated *rule space*. This score function, denoted as $s(\cdot)$, is used to evaluate the likelihood of each rule. Further explanation regarding the rule space will be provided in Sec. 4.2.

There are two primary categories of logical rule learning: (1) *traditional* search-based methods and (2) *recent* neural-symbolic integration approaches.

- **Search-Based Methods**: These methods are a traditional approach to logical rule learning. They involve searching through the rule space to identify the optimal set of logical rules that explain the data. Typically, these methods rely on heuristics and pruning techniques to improve the efficiency of the search process. They may struggle with larger or more complex datasets.

- **Neural-Symbolic Integration**: Neural-symbolic integration is a modern approach to logical rule learning that combines the strengths of rule-based symbolic reasoning and connectionist neural networks. Neural networks are used to learn representations of the data, while symbolic logic is used to derive high-level rules from the learned representations. This approach is more promising as it can learn patterns that may be difficult to express using symbolic logic alone.

In the following subsection, we will provide a detailed explanation of these two categories of methods, including a comparison of their advantages and disadvantages.

## 4.2 Search-based Logical Rule Learning

Search-based methods are traditional approaches to rule learning. These methods treat logical rules as discrete structures within KGs and model rule learning as a discrete optimization problem. The goal is to search over a discrete rule space to find the best set of rules that can effectively explain a given dataset.

Formally, we can define a *rule space* for rule learning in a KG as follows: Given a knowledge graph $\mathcal{G} = \{\mathcal{E}, \mathcal{R}, \mathcal{O}\}$, let $\mathcal{L}$ be the set of possible Horn rules that can be learned from $\mathcal{G}$, where each rule is represented as a pair $(Head, Body)$, denoting a Horn rule in the form $Head \leftarrow Body$, where $Head = r_h(x_h, y_h)$ is a single atom (a.k.a., atomic formula) representing the head of the rule (i.e., the conclusion) and $Body = r_{b_1}(x_{b_1}, y_{b_1}) \wedge \cdots \wedge r_{b_l}(x_{b_l}, y_{b_l})$ is a conjunction of atoms representing the body of the rule (i.e., the premise). A formal definition of Horn rules can be found in Sec. 2.3.

The rule space for logical rules with a length of $n$ can be formally defined as a structured space that encompasses all possible ways of constructing $r_{b_1}(x_{b_1}, y_{b_1}), \ldots, r_{b_l}(x_{b_l}, y_{b_l})$. For instance, in a KG with four relations $\{hasMother, hasGrandma, hasUncle, hasSon\}$, the search space for chain-like Horn rules with a length of 2 and a head relation *hasGrandma* can be visualized as the tree in Fig. 4.2. Here, a chain-like Horn rule is a special form of Horn rule where the rule body has

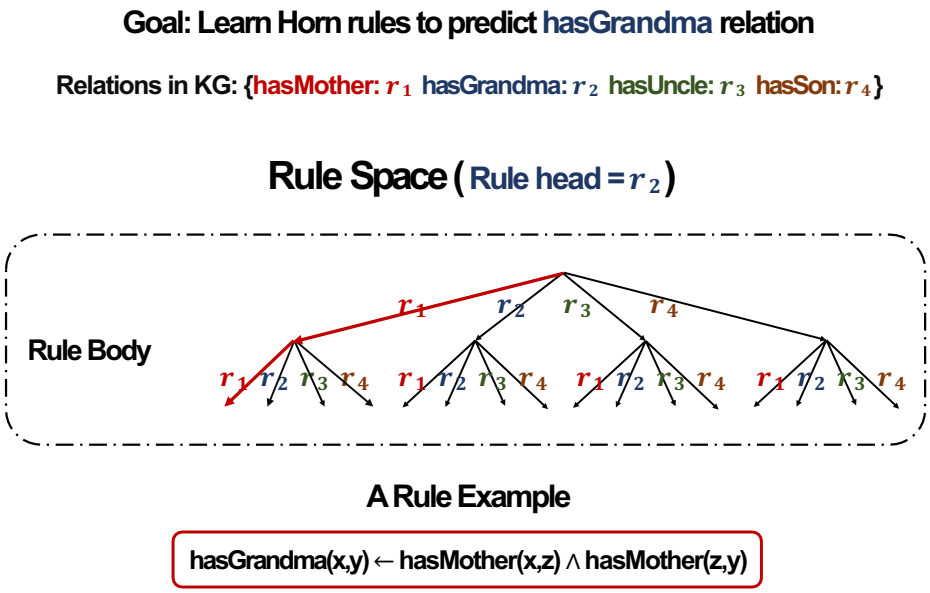



Figure 4.2: The search space for rules with head relation $r_2$ and body length 2. A path from the root to the leaf corresponds to a possible rule body. Together with the head relation, it forms a candidate rule. The figure highlights an example of a candidate rule that corresponds to the path in red.

a chain-like structure (a path in the KG). An example of a chain-like horn rule is given below:

$$r_h(x, y) \leftarrow r_{b_1}(x, z_1) \wedge r_{b_2}(z_1, z_2) \wedge \cdots \wedge r_{b_l}(z_{l-1}, y) \tag{4.3}$$

For a more comprehensive overview of chain-like Horn rules, please refer to Sec. 2.3. Each path in Fig. 4.2 from the root to the leaf corresponds to a candidate rule body, which, when combined with the head relation, forms a candidate rule. The highlighted path in red shows an example of a candidate rule in rule space.

$$\text{hasGrandma}(x, y) \leftarrow \text{hasMother}(x, z) \wedge \text{hasMother}(z, y) \tag{4.4}$$

The rule space size of a chain-like Horn rule is $|\mathcal{R}|^l$, where $|\mathcal{R}|$ denotes the number of relations in the KG and $l$ denotes the rule length. For instance, considering Horn rules with a length of 2 and a head relation *hasGrandma* in Fig. 4.2, the rule space comprises $4^2$ potential rules. It is clear that the search space for rule learning methods can be vast, especially as rule lengths increase. Due to the potentially large rule space, the primary objective of search-based methods is to efficiently navigate the rule space while minimizing computational complexity.

Search-based methods are part of traditional symbolic AI, in which knowledge is expressed using symbols, making it more comprehensible for humans. Furthermore, unlike neural methods, search-based methods do not need a significant number of examples for learning.

Existing search-based methods for rule learning can be broadly divided into two categories: (1) *Inductive Logic Programming (ILP)*, which is a subfield of logic programming. ILP involves learning from examples to induce a set of rules or hypotheses that can be used to make predictions or classify new examples. (2) *Association Rule Mining*, which draws inspiration from the data mining domain. This technique is widely used to identify frequent patterns and correlations in the data. In the following section, we will explore the most representative methods in each category.

### 4.2.1 Inductive Logic Programming (ILP)

Inductive Logic Programming (ILP) [65, 77] is a subfield of Artificial Intelligence (AI) that was first proposed in the late 1980s by Stephen Muggleton. The main objective of ILP is to induce

logical rules or hypotheses from a given set of examples or data, typically represented in the form of predicates (e.g., triples in KG). The formal definition of the ILP problem is provided as follows:

Given the background knowledge base (KB) $B$ [1], which contains facts stored as predicates, a set of positive examples denoted as $E^+$, as well as a set of negative examples denoted as $E^-$, ILP aims to learn Horn rules (i.e., $\mathcal{L}$) which are able to entail all the positive examples while excluding any of the negative examples:

$$\forall P \in E^+ : B \wedge \mathcal{L} \models P$$
$$\forall P \in E^- : B \wedge \mathcal{L} \not\models P$$

where the examples are ground predicates denoted as $P$ and $\models$ denotes logical entailment.

ILP algorithms typically involve a search strategy in the rule space and a scoring function $s(\cdot)$ to evaluate the quality of each rule. ILP has been successfully applied to a variety of domains, such as bioinformatics and natural language processing. We now use an example to illustrate how ILP works.

**Example 4.2.1.** Given a KB on kinship consists of the following facts:

{parent(Antony, Bess), parent(Antony, Charlotte), parent(Diana, Bess),
father(Antony, Bess), female(Charlotte), father(Antony,Charlotte),
mother(Diana, Bess), male(Antony), female(Diana)}

Two positive examples are given, i.e., $E^+$ = {father(Antony, Bess), father(Antony, Charlotte)}, and ten negative examples $E^-$ are listed below:

{father(Antony, Diana), father(Bess, Antony), father(Bess, Charlotte),
father(Bess, Diana), father(Charlotte, Antony), father(Charlotte, Bess),
father(Charlotte, Diana), father(Diana, Antony), father(Diana, Bess),
father(Diana, Charlotte)}

[1]Knowledge base (KB) is a more general concept than knowledge graph, which contains (1) facts that are beyond binary predicates and (2) rules.

In this case, our goal is to learn rules to infer the target relation (i.e, rule head) - $father(x, y)$. In order to learn rules, the general ILP approach begins with the target relation $father(x, y)$ and an empty rule body. To construct the rule body, ILP adds a literal that provides maximal coverage of the positive examples $E^+$ and minimal coverage of the negative examples $E^-$. The possible candidate literals to be added to the body are given as follows, which are all possible predicates in $B$ filled with variables $x, y, z$:

$$\{\text{male}(x), \quad \text{male}(y), \quad \text{female}(x), \quad \text{female}(y), \quad \text{parent}(x, y),$$
$$\text{parent}(x, z), \quad \text{parent}(y, x), \quad \text{parent}(y, z), \quad \text{parent}(z, x), \quad \text{parent}(z, y)\}$$

To simplify this example, we will only focus on literals that have at least one variable in common with the target relation $father(x, y)$.

In order to choose the most useful literal from the candidate set, we utilize the *gain heuristic* as the score function $s(\cdot)$ to evaluate the contribution of a literal $\phi$ to the target relation, which is defined as:

$$\text{gain}(\phi) = T * (\log_2(E^+_{\text{post}}/(E^+_{\text{post}} + E^-_{\text{post}})) - \log_2(E^+_{\text{pre}}/(E^+_{\text{pre}} + E^-_{\text{pre}}))) \tag{4.5}$$

where $E^+_{\text{post}}$ and $E^-_{\text{post}}$ respectively refer to the positive and negative examples that satisfy the rule *after* adding the literal, and $E^+_{\text{pre}}$ and $E^-_{\text{pre}}$ respectively refer to the positive and negative examples satisfied by the rule *before* adding the literal. $T$ is the number of positive examples that satisfied the rule before adding the new literal, and are still covered after adding the literal. The gain heuristic formula measures the contribution of a literal to the target relation by evaluating the difference between the information content of the positive and negative examples satisfied by the rule before and after adding the literal. The higher the information gain of a literal, the more useful it is in increasing its coverage of the target relation.

According to the definition of the gain, we compute the gain value for all the literals. As an example, let us compute the gain value for the literal $male(x)$, given the starting rule with rule head $father(x, y)$ and an empty body. Before adding the literal $male(x)$, the rule covers 2 positive examples and 10 negative examples, denoted as $E^+_{\text{pre}} = 2$ and $E^-_{\text{pre}} = 10$, respectively. After adding the literal, the rule $father(x, y) \leftarrow male(x)$ covers 2 positive examples and 4 negative examples,

denoted as $E^{+}_{\text{post}} = 2$ and $E^{-}_{\text{post}} = 4$, respectively. Since the number of positive examples that satisfy the rule remains constant before and after the literal is added, we have $T = 2$. Using the gain heuristic formula, we can compute the gain value for the literal $male(x)$ as:

$$\text{gain}(\text{male}(x)) = 2 * (\log_2(2/6) - \log_2(2/12)) = 2.0 \tag{4.6}$$

Similarly, the gain values for all the other literals can be computed in the same way as follows:

$$\begin{array}{cccc} \text{male}(x) : 2.0 & \text{male}(y) : 0.0 & \text{female}(x) : 0.0 & \text{female}(y) : 0.0 \\ \text{parent}(x, y) : 4.0 & \text{parent}(x, z) : 2.83 & \text{parent}(y, x) : 0.0 & \text{parent}(y, z) : 0.0 \\ & \text{parent}(z, x) : 0.0 & \text{parent}(z, y) : 2.0 & \end{array}$$

We observe that the literal $parent(x, y)$ has the highest gain value. We add $parent(x, y)$ to the rule body and get $father(x, y) \leftarrow parent(x, y)$. As this rule still entails one negative example (father(Diana, Bess)), we keep looking for additional literals. The gain value for all the literals is given as:

$$\begin{array}{cccc} \text{male}(x) : 1.17 & \text{male}(y) : -0.41 & \text{female}(x) : 0.0 & \text{female}(y) : 0.59 \\ \text{parent}(x, y) : 0.0 & \text{parent}(x, z) : 0.53 & \text{parent}(z, y) : 0.0 & \end{array}$$

As the literal $male(x)$ has the highest gain and the resulting rule is satisfied by all the positive and negative examples. Hence, we have successfully learned the rule:

$$\delta_1 := \text{father}(x, y) \leftarrow \text{parent}(x, y) \wedge \text{male}(x)$$

Algorithm 2 presents a general algorithm for the ILP approach. The algorithm takes as input a set of positive and negative examples (denoted as $E^{+}$ and $E^{-}$, respectively), a background knowledge base $B$ (which may contain previously learned rules or logical axioms), and a specific target relation (i.e., rule head). The algorithm initializes the rule body as empty and iteratively adds a literal to the rule body until the logical rule entails all positive examples and none of the negative examples. In each iteration, the algorithm selects the literal that maximally covers the positive examples and minimally covers the negative examples (Algorithm 3). Upon completion of the iterations,

the algorithm outputs the learned rules. The specific implementation of this algorithm may vary depending on the ILP system and the type of data being learned. However, this general framework provides a guideline for the ILP learning process.

**Algorithm 2:** General Algorithm of ILP Approach

**Input:** Background knowledge base $B$, positive examples $E^+$, negative examples $E^-$, target relation $r_h(x, y)$

**Output:** A set of Horn rules $\mathcal{L}$

$\mathcal{L} \leftarrow \emptyset$

**while** $E^+ \neq \emptyset$ **do**

  $(r_h(x, y), Body) \leftarrow$ Learn New Rule$(B, E^+, E^-, r_h)$

  $E^+ \leftarrow E^+ -$ positive examples satisfied by $r_h(x, y) \leftarrow Body$

  $\mathcal{L} \leftarrow \mathcal{L} \cup (r_h(x, y), Body)$

**end**

return $\mathcal{L}$

**Algorithm 3:** Learn New Rule

**Input:** Background knowledge base $B$, positive examples $E^+$, negative examples $E^-$, target relation $r_h(x, y)$

**Output:** An Horn rule $r_h(x, y) \leftarrow Body$

$Body \leftarrow \emptyset$

**while** $E^- \neq \emptyset$ **do**

  literal $\leftarrow$ choose_literal$(B, E^+, E^-, Body)$

  $Body \leftarrow$ literal $\wedge\, Body$

  $E^- \leftarrow E^- -$ negative examples satisfied by $r_h(x, y) \leftarrow Body$

**end**

return $r_h(x, y) \leftarrow Body$

**Summary** Inductive Logic Programming (ILP) is a specialized area within Artificial Intelligence (AI) that concentrates on deriving logical rules from given examples. ILP can incorporate domain-specific knowledge or constraints (i.e., represented as the background knowledge base in the input)

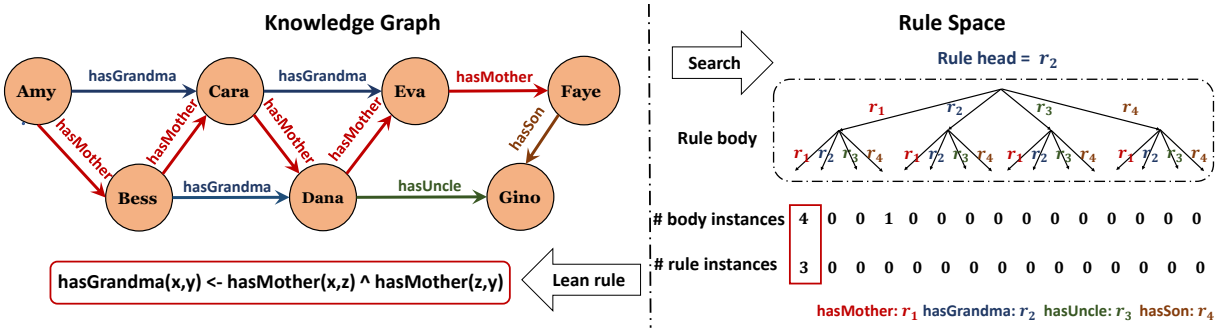



Figure 4.3: Left: An example KG and a detected rule with length 2. Right: A search tree for rules with length 2 and head relation $r_2$. A path from the root to the leaf corresponds to a rule body. Together with the head relation, it forms a candidate rule.

and produces comprehensible rule sets that can be easily interpreted and understood by humans, making it ideal for applications where human-understandable and interpretable rules are essential, such as expert systems and legal reasoning. Nevertheless, ILP techniques for rule learning utilize greedy search strategies, which make locally optimal decisions without taking into account the global optimality.

### 4.2.2 Association Rule Mining-based Approaches

ILP shares common ground with data mining techniques, as both aim to uncover hidden patterns in data. Traditional data mining methods have been widely applied for logical rule learning as well. The association rule mining-based approach, inspired by association rule mining in the data mining field, offers an alternative to ILP methods. This approach assesses the plausibility of Horn rules in KGs using predefined metrics such as support and confidence. Since these metrics involve assessing the overall number of occurrences of entities and predicates, they can be easily adjusted to filter out noisy patterns. As a result, when compared to ILP, whose objective is to learn rules that infer *all* positive examples and exclude *all* negative examples, the association rule mining-based approach typically exhibits greater resilience to noise in the data.

However, calculating support and confidence requires the counting of ground rules (or path instances) for all potential rules in the rule space as shown in Fig. 4.3. Consequently, the most

significant challenge in utilizing the association rule mining-based approach is the substantial computational complexity, stemming from two primary factors: (1) The need to evaluate the plausibility of all potential rules in the rule space to select the most reliable logical rules (big search space). As discussed in Sec. 4.2, the rule space can be quite extensive. With increasing rule length, the search space experiences exponential growth. (2) The need to identify all ground rules that conform to a candidate rule template to evaluate the plausibility of a single candidate rule (expensive grounding process). As an example, in Fig. 4.3, given the candidate rule:

$$\text{hasGrandma}(x, y) \leftarrow \text{hasMother}(x, z) \wedge \text{hasMother}(z, y) \tag{4.7}$$

we observe three rule instances following the rule:

$$\begin{aligned} &\text{hasGrandma(Amy, Cara)} \leftarrow \text{hasMother(Amy, Bess)} \wedge \text{hasMother(Bess, Cara)} \\ &\text{hasGrandma(Bess, Dana)} \leftarrow \text{hasMother(Bess, Cara)} \wedge \text{hasMother(Cara, Dana)} \\ &\text{hasGrandma(Cara, Eva)} \leftarrow \text{hasMother(Cara, Dana)} \wedge \text{hasMother(Dana, Eva)} \end{aligned} \tag{4.8}$$

The number of body instances and rule instances that conform to each candidate rule are counted and presented under the search tree in Fig. 4.3. The computational complexity of grounding a chain-like Horn rule is $|\mathcal{E}|^{(l+1)}$, which depends on the total number of entities in the KG, denoted by $|\mathcal{E}|$, and the rule length, denoted by $l$. To alleviate the immense computational complexity, most research in this area incrementally prunes the search space while learning logical rules.

The most representative approach in this category is *AMIE* [41, 42], which presents a formal model for the process of rule mining, following the idea of association rule mining.

AMIE involves two main steps to explore logical rules. The first step in the AMIE approach is called *rule extending*, which involves generating candidate rules by iteratively extending existing rules using three types of mining operators.

- *Add dangling atom*, where a dangling atom includes one new variable and one shared variable with another atom of the rule. It can be considered as extending a body path with another relation.

- *Add instantiated atom*, where an instantiated atom instantiates one variable with an entity

and shares the other variables with the rule. It can be considered as constraint a variable with a certain property, e.g., gender($x$, female).

- *Add closing atom*. Both variables of a closing atom are shared with the rule. This can be considered as the body path is closed with the shared two variables.

The second step in the AMIE approach is called *rule pruning*, which involves applying metrics such as *head coverage* and *confidence*, to evaluate the rules generated by the first step and output only plausible rules. The head coverage and confidence are defined as follows:

- *Head Coverage*: Head coverage of a rule is measured by the proportion of pairs from the head relation that are covered by the predictions of the rule. Taking the rule in Eq. 4.3 as an example, its head coverage can be calculated as:

$$\text{hc}(r_h(x, y) \leftarrow Body) = \frac{\text{supp}(r_h(x, y) \leftarrow Body)}{|\{(x', y') : r_h(x', y')\}|} \tag{4.9}$$

  Here, $Body := r_{b_1}(x, z_1) \wedge \cdots \wedge r_{b_n}(z_{n-1}, y)$ and $r_h(x', y')$ represents the collection of observed triples that have the relation $r_h$. The support $\text{supp}(r_h(x, y) \leftarrow Body)$ is defined as the number of $(x, y)$ pairs with relation $r_h$ that can be covered by the rule:

$$\text{supp}(r_h(x, y) \leftarrow Body) = |\{(x, y) : \exists z_1, \ldots, z_{n-1} : r_h(x, y) \leftarrow Body\}| \tag{4.10}$$

  where $z_1, \ldots, z_{n-1}$ are the variables of the rule body $r_{b_1}(x, z_1) \wedge \cdots \wedge r_{b_n}(z_{n-1}, y)$ apart from $x$ and $y$.

- *Standard Confidence*: Standard confidence is a metric that measures the strength of a rule by quantifying the proportion of times the rule is closed by relation $r_h$ among all the instances where the premise occurs. Using the same example as in Eq. 4.3, its standard confidence can be calculated as:

$$\text{conf}(r_h(x, y) \leftarrow Body) = \frac{\text{supp}(r_h(x, y) \leftarrow Body)}{|\{(x, y) | \exists z_1, \ldots, z_{n-1} : Body\}|} \tag{4.11}$$

  where $|\{(x, y) | \exists z_1, \ldots, z_{n-1} : Body\}|$ denotes the number of the body instances observed in the KG.

These metrics evaluate the plausibility of logical rules, they can be considered as the score function $s(\cdot)$ in AMIE. By utilizing these metrics, AMIE prunes the rules that fail to meet predefined thresholds for head coverage and confidence. The rules that remain are output as the final set of learned logical rules. Meanwhile, AMIE leverages the monotonicity of the two metrics with respect to the rule length, similar in frequent pattern mining, and prune the longer rule candidates if the shorter ones already violate the threshold. Overall, the rule-pruning step in AMIE plays a critical role in improving the efficiency and accuracy of the rule-learning process by filtering out unpromising or unreliable rules.

Putting together, AMIE utilizes a bottom-up approach to generate candidate rules from a KG. It starts by initializing a queue containing only the empty rule. Then, the algorithm dequeues a rule from the queue and checks whether the rule is closed or not. If the dequeued rule is closed, it is output as a candidate rule; otherwise, the algorithm applies all available operators to the rule. The available operators include "Add dangling atom", "Add instantiated atom", and "Add closing atom" that we discussed earlier. These operators are used to add atoms to the rule, and the resulting rules are added to the queue unless they are pruned out. The algorithm iteratively applies these steps to generate a set of candidate association rules. The top-$k$ rules are selected from this set and presented to the user.

**Summary** The association rule mining-based approach is particularly effective at generating simpler rules that highlight co-occurrence or correlation within the data. When compared to ILP, the association rule mining-based approach tends to exhibit greater resilience to noise in the data, as it measures a rule with metrics such as coverage or confidence. Also, it does not need negative examples for the learning process. Nevertheless, determining suitable thresholds for those metrics can be a complex task, as these thresholds significantly influence the quantity of discovered rules and the balance between their generality and specificity.

## 4.3 Neural Symbolic Integration for Logical Rule Learning

Despite being the most studied approach to rule learning since the earliest days of AI, searching-based methods suffer from several limitations, including:

- **Lack of Scalability:** Searching-based methods are not well-suited for large datasets or complex problems as fundamentally they are addressing the combinatorial discrete optimization problem, which is computationally expensive. Most searching-based methods for rule learning employ greedy search strategies that lead to local optimal yet still hard to scale [65].
- **Sensitivity to Incompleteness and Noises in KG:** Searching-based methods for rule learning can be sensitive to incompleteness and noise in the data, which can lead to missing important rules or the production of logical rules that fit the noise. This is particularly true for inductive logic programming (ILP) methods, where the objective is to learn rules that infer all positive examples and exclude all negative examples.

Neural-symbolic integration offers an alternative method to learning logical rules by combining symbolic logical rule learning with neural network models, leveraging the strengths of both approaches to handle the challenges of learning from real-world KGs.

One of the key challenges presented by real-world data is its large scale. Neural networks can handle large-scale data by enabling end-to-end optimization through gradient-based methods. By exploiting the differentiability of neural networks, it is possible to apply continuous optimization techniques to the task of learning logical rules. This transition from a discrete to a continuous problem allows for a more effective exploration of the space of rules, which in turn enables handling the difficulties associated with managing large datasets.

Another challenge presented by real-world data is incompleteness and noise. Representation learning-based neural network methods can largely alleviate the incompleteness and noise issues by capturing the correlation between entities and relations, as evidenced by knowledge graph completion (Chapter 3). This enables neural-symbolic integration to learn complex patterns that may be difficult to express using symbolic logic alone.

Existing neural-symbolic integration approaches can be roughly divided into two categories: (1) *differentiable searching-based methods*, which directly extend traditional searching-based methods by transforming the logical rule learning problem from a discrete to a continuous problem using a differentiable loss function; (2) *learning based methods*, which combine various learning paradigms to learn rules from data, where the rules are represented in a symbolic form.

### 4.3.1 Differentiable Searching-based Methods

Differentiable search-based methods combine traditional search-based logical rule learning with modern neural network models, transforming the rule learning problem from a discrete optimization problem to a continuous optimization problem. The differentiability of neural networks is exploited to enable this transformation. By formulating the rule learning problem as an end-to-end continuous optimization problem, these methods make the searching process differentiable, enabling the use of gradient-based optimization techniques such as stochastic gradient descent (SGD). Gradient-based optimization makes the learning process more efficient compared to combinatorial search techniques used in traditional searching-based rule learning methods, enabling better scalability for large-scale KGs. Furthermore, with the integration of deep learning architectures, these methods can effectively take advantage of representation learning capabilities. As a result, they are capable of learning complex patterns and exhibit greater robustness to noise and incompleteness in the data compared to traditional searching-based rule learning methods.

Existing neural-symbolic integration approaches can be broadly classified into two categories based on the types of traditional searching-based methods employed: (1) *Differentiable ILP*, which combines Inductive Logic Programming (ILP) with deep learning principles to learn logical rules; (2) *Matrix-based reasoning*, which integrates association rule mining-based approaches with deep gradient-based learning systems to learn logical rules.

#### 4.3.1.1 Differentiable ILP

Differentiable Inductive Logic Programming (DILP) is a type of machine learning algorithm that combines the principles of Inductive Logic Programming (ILP) and deep learning to learn logic rules. In particular, DILP methods use differentiable logic programs that can be trained using gradient-based optimization techniques. Differentiable logic programs are similar to traditional logic programs but include additional operations, such as differentiable logic gates, that can be used to construct neural network-like structures. The most representative method in this category is Neural Theorem Provers (NTPs).

**Neural Theorem Provers (NTPs)** Neural Theorem Provers (NTPs) [101] extends the traditional backward chaining algorithm to learn logic rules. The backward chaining algorithm, as previously discussed in Chapter 3.2.2.2, is a top-down reasoning approach used in logic programming and theorem proving. It begins with the goal (or conclusion) and works backward, recursively breaking the goal into subgoals and searching for rules that can satisfy those subgoals.

The traditional backward chaining algorithm faces limitations when dealing with incomplete or noisy KGs. NTPs enhance the backward chaining algorithm by using continuous vector space embeddings to represent entities, relations, and rules, which makes it possible to capture similarities between different entities and relations. NTPs model logical rules in a continuous vector space, transforming the backward chaining procedure into a sequence of differentiable operations, which allows for end-to-end learning using gradient-based optimization methods. Following the principle of Inductive Logic Programming (ILP), these embeddings are learned end-to-end by maximizing the scores of facts in the knowledge graph (KG), while minimizing the score of facts not in the KG.

To illustrate how NTPs extend the backward chaining algorithm, we provide an example to compare the backward chaining algorithm and NTPs:

**Example 4.3.1. Backward Chaining Algorithm.** Consider a simple KG with the following two

facts and one rule:

$$\text{Father(Anthony, Bob)}$$

$$\text{Father(Bob, Christian)}$$

$$\text{GrandFather}(x, y) \leftarrow \text{Father}(x, z) \wedge \text{Father}(z, y)$$

Our goal is to prove the query *"GrandFather(Anthony, Christian)."* Backward chaining would proceed as follows: The query *GrandFather(Anthony, Christian)* can be verified by mapping it with the rule head *GrandFather(x, y)*. After replacing $x$ with *"Anthony"* and $y$ with *"Christian"* in the rule body, the task of proving the goal *GrandFather(Anthony, Christian)* is transformed into proving two subgoals: *Father(Anthony, z)* and *Father(z, Christian)*. By substituting $z$ with *"Bob"*, the subgoals *Father(Anthony, Bob)* and *Father(Bob, Christian)* can be recursively demonstrated as true. Thus, the backward chaining algorithm can successfully confirm the query *GrandFather(Anthony, Christian)*.

Using the same example, we now show how NTPs extend the backward chaining algorithm for rule learning.

**Example 4.3.2. Neural Theorem Provers (NTPs)**

Let us continue with the previous backward chaining example. In addition to entities Anthony, Bob, and Christian, assume we also have other two entities Tony and Chris, relations $GrandFather$ and $Grandpa$, and other related facts in the KG, where Tony is a nickname for Anthony and Chris is a nickname for Christian. Our goal now has changed slightly. Instead of proving the query *GrandFather(Anthony, Christian)*, we aim to prove the query *GrandPa(Tony, Chris)*. Using the backward chaining algorithm, we cannot prove *GrandPa(Tony, Chris)* because there is no proof path that leads to the conclusion *GrandPa(Tony, Chris)*, where a *proof path* refers to the sequence of literals (i.e., ground predicates) that are used to derive a specific ground instance of a rule from the background knowledge. For instance, *GrandFather(Anthony, Christian)* $\leftarrow$ *Father(Anthony, Bob)* $\wedge$ *Father(Bob, Christian)* is a proof path leading to the conclusion *GrandFather(Anthony, Christian)*. We now demonstrate how NTPs prove *GrandPa(Tony, Chris)*. Additionally, we will demonstrate how NTPs can learn new rules by utilizing *parameterized rule*.

Different from the backward chaining algorithm, Neural theorem provers (NTPs) learns em-

beddings for all relations and entities using a bilinear KG embedding model - ComplEx, which has been introduced ComplEx previously in Sec. 3.3.3. For example, embeddings for relations $\mathbf{r}_{\text{GrandFather}}$ and $\mathbf{r}_{\text{GrandPa}}$, as well as for entities $\mathbf{e}_{\text{Anthony}}$ and $\mathbf{e}_{\text{Tony}}$, are represented in $\mathbb{R}^d$. With these embeddings, NTPs can compare symbols using a similarity measure between their respective embeddings, denoted as $f : \mathbb{R}^d \times \mathbb{R}^d \rightarrow [0, 1]$. For example, $e_{\text{Anthony}}$ might be similar to $e_{\text{Tony}}$ because they share similar neighboring nodes in the KG.

To prove the query *GrandPa(Tony, Chris)*, NTPs first compute its score via embedding-based score function. If the query cannot be directly inferred from the KG embedding model, NTPs performs backward chaining, which involves using human-predefined logical rules such as the rule *GrandFather(x, y)* $\leftarrow$ *Father(x, z)* $\wedge$ *Father(z, y)* in the background knowledge, or using a *parameterized rule* to represent an unknown rule needs to be learned. We will discuss how to use parameterized rules for rule learning later in this example.

To prove the query *GrandPa(Tony, Chris)*, a candidate proof path is created using the facts *Father(Anthony, Bob)* and *Father(Bob, Christian)* and the rule *GrandFather(x, y)* $\leftarrow$ *Father(x, z)* $\wedge$ *Father(z, y)*. The system calculates a score for each step in the backward chaining process, representing how well the rules and facts satisfy the current goal or subgoal. For instance, when attempting to unify the query *GrandPa(Tony, Chris)* with the rule head *GrandFather(x, y)*, NTPs compute the similarity score $f(\mathbf{r}_{\text{GrandFather}}, \mathbf{r}_{\text{GrandPa}})$. It then replaces variables $x$ and $y$ in the rule body with *"Tony"* and *"Chris"*. The task of proving the goal *GrandPa(Tony, Chris)* is then transformed into proving two subgoals: *Father(Tony, z)* and *Father(z, Chris)*. By substituting $z$ with *"Bob"*, the subgoals *Father(Tony, Bob)* and *Father(Bob, Chris)* are recursively evaluated using $f$(Father(Tony, Bob), Father(Anthony, Bob)) and $f$(Father(Bob, Chris), Father(Bob, Christian)). Finally, the NTP aggregates these scores and generates a final score for the query *GrandPa(Tony, Chris)*, indicating how likely it is that *GrandPa(Tony, Chris)* holds given the current KG.

The examples discussed above have demonstrated how NTPs can use the predefined logical rules in the background knowledge to infer new facts. In the next section, we will illustrate how rule learning can be seamlessly integrated into this reasoning process. Specifically, we aim to learn

a rule of the form:

$$r_h(x, y) \leftarrow r_{b_1}(x, z) \wedge r_{b_2}(z, y) \tag{4.12}$$

where $r_h$, $r_{b_1}$, and $r_{b_2}$ are unknown predicates represented as vectors in $\mathbb{R}^d$, and we want to learn their embeddings from the training data. We refer to Eq.(4.12) as a *parameterized rule* because the rules are unknown, and their representations are learned from the data. During training, the representations of parameterized rules are optimized along with all other symbolic representations. As a result, the model can adapt parameterized rules to ensure successful proofs for known facts while preventing proofs for sampled negative facts. After training, we can decode the parameterized rule by searching for the closest representations of known predicates.

NTPs can be used for Inductive Logic Programming (ILP) by leveraging gradient descent, rather than conducting a combinatorial search over the space of rules. The advantage of using NTPs for rule learning is that they learn continuous embeddings for entities and relations, which can handle noisy, incomplete, or inconsistent KGs. Moreover, NTPs can identify equivalent entities and relations using similarity measures between embeddings, making them more flexible in handling queries that involve synonyms or aliases. However, a drawback of NTPs is that the continuous embeddings learned may not be easily interpretable by humans. The decoding process from parameterized rules to interpretable rules can make it difficult to understand the underlying reasoning process.

**Connect differentiable ILP to traditional ILP** Differentiable ILP (DILP) methods extend ILP by incorporating deep learning ideas, offering a differentiable framework for joint learning of embeddings of symbols and interpretable rules. DILP methods can place similar symbols in close proximity in a vector space, enhancing logical rule learning by leveraging a single rule for many proofs of queries with symbols that have a similar representation. DILP methods have several advantages over traditional ILP methods. First, they can learn more complex logic programs than traditional ILP methods, which are often limited by the expressiveness of the logic language used. Second, DILP methods can learn from continuous and noisy data, whereas traditional ILP methods are typically designed for discrete and noise-free data. However, as DILP is an extension

of ILP approaches, it still requires all possible proof paths to answer a given query, leading to computational inefficiency, even for a small KG. Although efforts have been made to reduce the proof paths, this remains a central drawback of DILP.

#### 4.3.1.2 Matrix-based Reasoning

Matrix-based reasoning is another differentiable searching-based method that builds upon association rule learning-based approaches. By using sparse matrix multiplication for logical inference, matrix-based reasoning turns the discrete searching problem into a differentiable problem, allowing logical rules to be learned from data using gradient-based optimization methods. Neural-LP is one of the most representative methods in this category. It is inspired by a recently developed differentiable logic called TensorLog [25]. We will begin by introducing TensorLog.

**TensorLog** As discussed in Chapter 4.2.2, traditional association rule mining-based approaches require identifying all ground rules conforming to a candidate rule template to evaluate the plausibility of a single candidate rule. This process, known as logical inference following the candidate rule, involves discrete counting and can be computationally expensive when dealing with large datasets or complex rules, limiting its scalability to real-world applications. To address this challenge and perform logical inference more efficiently, TensorLog [25] was introduced. TensorLog makes the first attempt to establish a connection between logical inference and sparse matrix multiplication. As discussed in Sec. 2.3.2, the body of a chain-like Horn rule is essentially a path in the KG. The number of instances of such a path can be computed using matrix multiplication.

TensorLog defines an operator $\mathbf{M}^{(k)}$ for each relation $r_k$ and a one-hot vector $\mathbf{e}_i$ for each entity $e_i$, where $\mathbf{M}^{(k)}$ is a matrix in $\{0,1\}^{|\mathcal{E}|\times|\mathcal{E}|}$ such that its $(i,j)$ entry is 1 if and only if the triple $(e_i, r_k, e_j)$ is in the KG and $\mathbf{e}_i$ is a one-hot encoded vector $\{0,1\}^{|\mathcal{E}|}$ such that only the $i$-th entry is 1. The scalar $\mathbf{e}_i^\intercal \mathbf{M}^{(b_1)}\mathbf{M}^{(b_2)}\ldots\mathbf{M}^{(b_l)}\mathbf{e}_j$ is then equal to the number of paths connecting entity $e_i$ to $e_j$ through the chain-like rule body $r_{b_1}(x,z_1) \wedge r_{b_2}(z_1,z_2) \wedge \cdots \wedge r_{b_l}(z_{l-1},y)$. During inference, given a set of logical rules with $r_h$ as the head relation, TensorLog can answer the query $(?, r_h, e_j)$ via a score function $s(e_i, r_h, e_j)$. The score of each head entity is equivalent to the entries in the

vector $\mathbf{v}$:

$$\mathbf{v} = \sum_{\mathbf{r}_b \in Body(r_h)} s(r_h, \mathbf{r}_b) \prod_{r_{b_i} \in \mathbf{r}_b} \mathbf{M}^{(r_{b_i})} \mathbf{e}_j \ , \tag{4.13}$$

where $Body(r_h)$ is the set of potential rule bodies whose rule head is $r_h$, $\mathbf{r}_b = [r_{b_1}, \ldots, r_{b_l}]$ is a chain-like rule body and $r_{b_i}$ is a relation in the body, $s(r_h, \mathbf{r}_b)$ denotes the confidence score associated with the rule $(r_h, \mathbf{r}_b)$, $\mathbf{e}_i$ and $\mathbf{e}_j$ are the one-hot encoded vectors for entities $e_i$ and $e_j$, respectively. The score function is then computed as: $s(e_i, r_h, e_j) = \mathbf{e}_i^\top \mathbf{v}$.

Although TensorLog improves the efficiency of logical inference on large datasets using sparse matrix multiplication, it is limited as it can only learn the confidence score $s(r_h, \mathbf{r}_b)$, not rules. It is still a challenging task to extend TensorLog for differentiable logical rule learning. A straightforward approach is to reformulate Eq.(4.13) to learn $s(r_h, \mathbf{r}_b)$ in such a way that the score $s(e_i, r_h, e_j)$ for observed triples in KG is maximized. However, since each confidence score $s(r_h, \mathbf{r}_b)$ is associated with a specific rule $(r_h, \mathbf{r}_b)$, and the process of rule enumeration is intrinsically a discrete task, such a learning process is not suitable for gradient-based optimization.

To overcome this challenge, Neural-LP [136] turns the discrete searching problem into a differentiable problem, allowing logical rules to be learned from data using gradient-based optimization methods. In the following section, we will explain the technical details of Neural-LP.

**Neural-LP** To allow the rule searching process to be differentiable, Neural-LP [136] proposed exchanging the order of the summation and product in Eq.(4.13) to reduce the number of the parameter from $|\mathcal{R}|^L$ to $|\mathcal{R}|L$ :

$$\prod_{l=1}^{L} \sum_{k}^{|R|} \alpha_l^k \mathbf{M}^{(r_k)} \tag{4.14}$$

where $L$ is the max length of rules and $|\mathcal{R}|$ is the number of relations in the KG. Note that Eq.(4.13) and Eq.(4.14) differ in their parameterization, with the latter associating each relation in the rule with a weight, enabling rule enumeration and confidence score learning simultaneously within a differentiable framework. This innovation enables the use of gradient-based optimization methods for learning the confidence score $s(r_h, \mathbf{r}_b)$, which would otherwise be infeasible due to the discrete nature of the rule-searching process.

However, the parameterization in Eq.(4.14) has limited expressiveness since it assumes that all rules are of the same length. To address this limitation, Neural-LP proposes a recurrent formulation. In the recurrent formulation, Neural-LP employs auxiliary memory vectors $\mathbf{u}_l$, which are initially set to the one-hot encoded vectors of entity $e_i$, i.e., $\mathbf{e}_i$. To sequentially compose the differentiable tensor multiplication, Neural-LP designs a neural controller system based on an attention mechanism.

$$\begin{aligned}
\mathbf{u}_0 &= \mathbf{e}_j \\
\mathbf{u}_l &= \sum_{k}^{|\mathcal{R}|} a_l^k \mathbf{M}^{(r_k)} \Big(\sum_{\tau=0}^{l-1} b_l^\tau \mathbf{u}_\tau\Big) \text{ for } 1 \le l \le L \\
\mathbf{u}_{L+1} &= \sum_{\tau}^{L} b_{L+1}^\tau \mathbf{u}_\tau
\end{aligned} \tag{4.15}$$

To include all previous partial inferences, the Neural-LP model computes a weighted average of previous memory vectors using the memory attention vector $\mathbf{b}_l$ and softly applies the TensorLog operators using the operator attention vector $\mathbf{a}_l$ at each step. The final inference result is given by the last vector in memory, $\mathbf{u}_{L+1}$. The objective of Neural-LP is to learn rules that can best predict the facts in the KG, which is formulated as maximizing the function:

$$\log \mathbf{e}_i^\intercal \mathbf{u}_{L+1} \tag{4.16}$$

To recover logical rules from the neural controller system, Neural-LP expresses rules and their confidence score $s(r_h, \mathbf{r}_b)$ in terms of the attention vectors $\mathbf{a}_l, \mathbf{b}_l$ for each query.

**Connect matrix-based reasoning to traditional association rule learning** Similar to association rule learning-based approaches, neural logic programming computes the confidence score of every candidate rule in the rule space to select Horn rules with the highest confidence score. However, to make the framework differentiable, neural logic programming replaces the discrete counting of ground rules with sparse matrix multiplication. This transformation enables logical inference to be compiled into sequences of differentiable tensor multiplication, allowing for the use of neural networks to learn logical rules. Despite the benefits of this approach, neural logic programming can be computationally inefficient due to its reliance on large matrix multiplication,

and it makes fewer efforts in reducing the large search space of logical rules. As a result, while it can be applied to some real-world KGs such as WN18 [35] and FB15K [121], it is still not efficient enough to handle larger KGs, such as YAGO3-10 [116]. The detailed statistics of these datasets are summarized in Table 4.2. Additionally, unlike Neural Theorem Provers (NTPs), matrix-based reasoning methods are unable to learn embeddings of entities and relations, which are essential for identifying equivalent entities and relations. This limitation restricts their ability to handle queries that involve synonyms or aliases.

### 4.3.2 Learning Based Methods

While differentiable searching-based methods offer an elegant approach to combining symbolic logical rule learning with neural network models by searching for rules that optimize a differentiable loss function using gradient-based optimization algorithms, they also present several limitations:

- **Limited expressiveness** Differentiable searching-based methods suffer from limited expressiveness compared to traditional rule-based systems. This is because they can only learn rules that are expressible in the search space of differentiable functions that the optimization algorithm can handle. For instance, Neural-LP is only capable of learning chain-like Horn rules since the body of these rules can be effectively modeled using matrix multiplication. In contrast, AMIE can learn Horn rules in a more general form.

- **Difficulty with complex rules**: Although differentiable searching-based methods often scale better than traditional searching-based methods, they still struggle to learn complex rules involving multiple variables and intricate interactions between them. This is because, when dealing with complex rule sets, the search space can be extensive. For instance, in order to learn long rules, NTPs have to involve deep proof paths, leading to computational inefficiency. Generally, the larger the search space, the more computationally expensive the optimization process becomes.

- **Limited interpretability of neural network components**: While the rules learned by differentiable searching methods are interpretable, the neural network components employed in the

optimization may not be easily understandable. This can make it challenging to comprehend how the rules were learned, limiting the method's usefulness in certain applications. For example, the continuous embeddings learned by NTPs may not be easily interpretable by humans, making it difficult to understand the underlying reasoning process.

Learning-based methods offer an alternative approach to combining symbolic logical rule learning with neural network models [39]. Rather than simply extending and improving traditional searching-based methods, learning-based methods can be integrated with a broader range of learning paradigms, which increases their versatility for learning different types of rules. For instance, learning-based methods can be combined with sequential generative models, which are useful for generating *chain-like Horn rules* in KGs, or with reinforcement learning models, which are particularly effective for modeling *compositional rules*. We have summarized and compared the capabilities of differentiable searching-based methods and learning-based methods in Table 4.1. We can observe that while differentiable searching-based methods offer some advantages, such as end-to-end learning and continuous search spaces, learning-based methods can excel in flexibility by supporting various learning paradigms, making them suitable for a diverse range of tasks.

RNNLogic [94] and R5 [71] are two recent learning-based methods for logical rule learning. Each method adopts a unique learning paradigm, offering different advantages in the rule learning process. RNNLogic separates rule generation and rule weight learning by introducing a rule generator and a reasoning predictor. The rule generator uses a Recurrent Neural Network (RNN) to generate rules, while the reasoning predictor learns the rule weights. RNNLogic can effectively learn logical rules from data while maintaining interpretability. On the other hand, R5 formulates rule learning as a sequential decision-making process and employs deep reinforcement learning to learn the logical rules. By treating rule discovery as a series of decisions made over steps, R5 can explore the search space more efficiently, potentially leading to better rule discovery and improved performance. In the following sections, we will provide a more detailed introduction to these two learning-based methods.

Table 4.1: Capabilities of neural-symbolic methods: differentiable searching-based methods v.s. learning-based methods.

| Capabilities | Differentiable Searching-Based Methods | Learning-Based Methods |
|---|---|---|
| End-to-end | Yes | May vary |
| Differentiable | Yes | Yes |
| Complexity | High | May vary |
| Flexibility in learning paradigms | Limited | High |

**RNNLogic** RNNLogic [94] is the most representative learning-based method for logical rule learning. Since the inefficiency of existing rule learning methods usually comes from searching in a large search space, to reduce search space and learn better rules, RNNLogic is proposed to separate rule generation and rule weight learning by introducing a *rule generator* and a *reasoning predictor* respectively. Before delving into the technical details of the rule generator and reasoning predictor, we will formally define the rule learning problem in a probabilistic way.

In RNNLogic, a set of logic rules is considered as a latent variable denoted by $\mathbf{z} \in \mathcal{R}^2 \cup \mathcal{R}^3 \cup \ldots \cup \mathcal{R}^L$ where $L$ is the maximum length of rules. The *rule generator*, denoted by $p_\theta(\mathbf{z}|\mathbf{q})$, defines a prior distribution over a set of latent rules $\mathbf{z}$ conditioned on a query $\mathbf{q}$. On the other hand, the *reasoning predictor*, denoted by $p_w(a|\mathcal{G}, \mathbf{q}, \mathbf{z})$, provides the likelihood of the answer $a \in \mathcal{E}$ given the latent rules $\mathbf{z}$, the query $\mathbf{q}$, and the KG $\mathcal{G}$. By utilizing a rule generator and a reasoning predictor, a KG reasoning task can be formulated as predicting the correct answer $a$ for a given query $\mathbf{q}$ and KG $\mathcal{G}$. This can be represented formally as the probability distribution $p(a|\mathcal{G}, \mathbf{q})$, which can be computed as follows:

$$p(a|\mathcal{G}, \mathbf{q}) = \sum_{\mathbf{z}} p_w(a|\mathcal{G}, \mathbf{q}, \mathbf{z}) p_\theta(\mathbf{z}|\mathbf{q}) = \mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{q})}[p_w(a|\mathcal{G}, \mathbf{q}, \mathbf{z})] \tag{4.17}$$

where $\theta$ and $w$ are the parameters for the rule generator and the reasoning predictor, respectively. To achieve the best performance for KG reasoning, RNNLogic aims to learn logic rules by jointly

training the rule generator and reasoning predictor to maximize the likelihood of the training data. This is formally defined as follows:

$$\begin{aligned}\max_{\theta,w} O(\theta, w) &= \mathbb{E}_{(\mathcal{G},\mathbf{q},a)\sim p_{\text{data}}}[\log p_{w,\theta}(a|\mathcal{G},\mathbf{q})] \\ &= \mathbb{E}_{(\mathcal{G},\mathbf{q},a)\sim p_{\text{data}}}[\log \mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{q})}[p_w(a|\mathcal{G},\mathbf{q},\mathbf{z})]]\end{aligned} \tag{4.18}$$

After formally defining the problem, we will proceed to present the technical specifications of the rule generator and the reasoning predictor.

**Rule Generator.** The rule generator is responsible for defining the distribution $p_\theta(\mathbf{z}|\mathbf{q})$. Given a query $\mathbf{q} = (e_i, r_h, ?)$, the rule generator aims at generating a set of latent rules $\mathbf{z}$ for answering the query. Specifically, a chain-like Horn rule $r_h(x, y) \leftarrow r_{b_1}(x, z_1) \wedge \cdots \wedge r_{b_n}(z_{n-1}, y)$ can be viewed as a sequence of relations $[r_h, r_{b_1}, r_{b_2}, \ldots, r_{b_n}, r_{\text{END}}]$, where $r_h$ is the query relation or the rule head, $\mathbf{r_b} = [r_{b_1}, \ldots, r_{b_n}]$ represents the rule body, and $r_{\text{END}}$ is a special relation indicating the end of the relation sequence. Combining the rule head and rule body, we denote a chain-like Horn rule as $(r_h, \mathbf{r_b})$. RNN is the natural choice to model a relation sequence. Therefore, we have

$$p_\theta(\mathbf{z}|\mathbf{q}) = \text{Mult}(\mathbf{z}|N, \text{RNN}_\theta(\cdot|r_h)), \tag{4.19}$$

where Mult stands for multinomial distributions, $N$ is the size of the set of logical rules $\mathbf{z}$.

**Reasoning Predictor.** The reasoning predictor aims to predict the answer $a$ for a query $\mathbf{q} = (e_i, r_h, ?)$ using a set of rules $\mathbf{z}$ to preform reason. Let $\mathcal{A}$ be the set of candidate answers which can be discovered by any logic rule in the set $\mathbf{z}$. For each candidate answer $e_j \in \mathcal{A}$, we compute the following scalar score for that candidate:

$$\text{score}_w(e_j) = \sum_{\mathbf{r_b}\in\mathbf{z}} \text{score}_w(e_j|\mathbf{r_b}) = \sum_{\mathbf{r_b}\in\mathbf{z}} \sum_{p\in\mathcal{P}(e_i,\mathbf{r_b},e_j)} \psi_w(\mathbf{r_b}) \tag{4.20}$$

where $\mathcal{P}(e_i, \mathbf{r_b}, e_j)$ is the set of grounding paths that start at $e_i$ and end at $e_j$ following the rule body $\mathbf{r_b}$. $\psi_w(\mathbf{r_b})$ are the scalar weights of rule $(r_h, \mathbf{r_b})$. Once we have the score for every candidate's answer, we can further define the probability that the answer $a$ of the query $\mathbf{q}$ is entity $e_j$ by using

a softmax function as follows:

$$p_w(a = e_j|\mathcal{G}, \mathbf{q}, \mathbf{z}) = \frac{\exp(\text{score}_w(e_j))}{\sum_{e' \in \mathcal{A}} \exp(\text{score}_w(e'))} \tag{4.21}$$

RNNLogic optimizes the reasoning predictor and rule generator in order to maximize the objective defined in Eq. 4.18. In each training iteration, the algorithm first maximizes the objective in Eq. 4.18 with respect to the reasoning predictor $p_w$. The parameter $w$ of the reasoning predictor $p_w(a|\mathcal{G}, \mathbf{q}, \mathbf{z})$ is updated to maximize the log-likelihood of the answer $a$ based on rules $\hat{\mathbf{z}} \sim p_\theta(\mathbf{z}|\mathbf{q})$ generated by the rule generator. Then, it updates the rule generator using the Expectation-Maximization (EM) algorithm. In the *E-step*, a set of high-quality rules $\mathbf{z}_I$ is identified from all generated rules through posterior inference $p_{\theta,w}(\mathbf{z}_I|\mathcal{G}, \mathbf{q}, \mathbf{z}) \propto p_w(a|\mathcal{G}, \mathbf{q}, \mathbf{z}_I)p_\theta(\mathbf{z}_I|\mathbf{q})$, using the rule generator as the prior and the reasoning predictor as the likelihood. In the *M-step*, the parameters $\theta$ of the rule generator $p_\theta$ are updated to be consistent with the high-quality rules $\mathbf{z}_I$ selected in the E-step.

Despite introducing a rule generator to reduce the search space, the reasoning predictor in RNNLogic still requires the counting of all ground rules or path instances based on the generated rules. This limits the efficiency of the algorithm. As a result, scaling RNNLogic to handle KGs with hundreds of relations (such as FB15K-237) or millions of entities (such as YAGO3-10) remains a challenging task.

**R5** Although the RNNLogic model has been successful in predicting missing links in benchmark KG datasets, it lacks a key aspect of machine intelligence known as systematicity. Systematicity involves a model's capacity to recombine established parts and rules to create novel sequences while reasoning over relational data. To illustrate this concept concretely, consider the scenario depicted in Figure 4.13. A robustly systematic model can effectively learn rule ③ by integrating logical rules ① and ②. This ability is critical for real-world applications, where data may be complex and varied, and the model must be able to reason over a wide range of scenarios.

Recently, R5 [71] has been proposed as a solution to overcome the lack of systematicity in RNNLogic. R5 formulates the task of relational reasoning as a sequential decision-making problem and employs *deep reinforcement learning* in combination with a *dynamic rule memory*

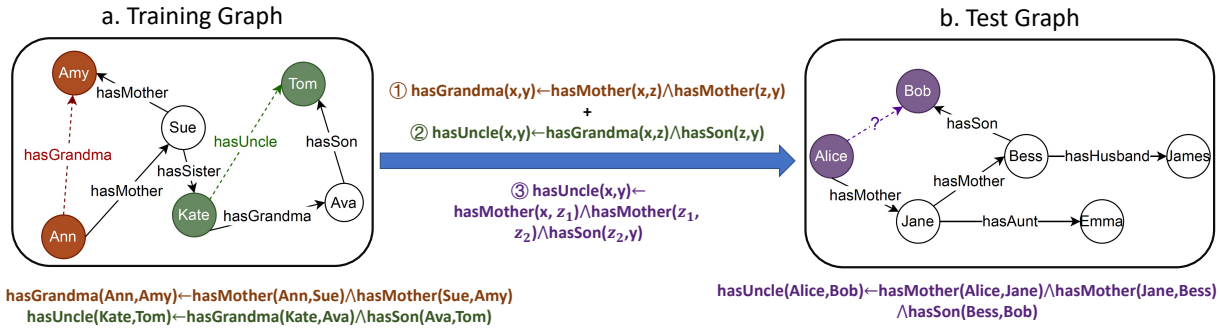



Figure 4.4: Illustration of how the compositionality of logical rules helps improve systematic generalization. (a) logical rule extraction from the observed graph (i.e., training stage) and (b) Inference on an unseen graph (i.e., test stage). The train and the test graphs have disjoint sets of entities. By combining logical rules ① and ② we can successfully learn rule ③ for prediction on unseen graphs.

to perform logical reasoning and rule extraction. Specifically, it initially converts a KG into a collection of paths linking the queried entity pairs. Subsequently, R5 iteratively employs learned rules to combine a pair of relations in the paths, resulting in a composite relation, until it derives a final relation connecting the two queried nodes.

**Recurrent Relational Reasoning with Deep Reinforcement Learning** To begin, we introduce the deep reinforcement learning-based reasoning agent, which models logical rule learning as a sequential decision-making problem. The training process of the reasoning agent involves utilizing the extracted relation paths linking the queried entity pairs as input and combining a policy value network with Monte Carlo Tree Search (MCTS) to perform iterative reasoning over these paths until it derives a final relation directly connecting the two queried nodes.

- **Action**: At each step of an episode, the MCTS outputs an action denoted as aB, which represents a relation pair $a_B = (a_{B,0}, a_{B,1})$. The model references the dynamic rule memory to obtain a rule of the form $a_H \leftarrow (a_{B,0}, a_{B,1})$, which implies that the relation pair $(a_{B,0}, a_{B,1})$ in the path will be replaced with $a_H$. We will address the implementation of the dynamic rule memory at a later point.

- **State**: Considering that an informative state should encode walking paths between query nodes, which can be represented as all the possible relations pairs among the current paths, $s \in \mathbb{R}^{(m+n)\times(m+n)\times k}$, where $m$ is the number of observed relations, $n$ is the number of invented

relations, and $k$ is the dimension of features of each relation pair.

- **Reward**: Within an episode, actions are iteratively applied to a relation path until the path is reduced to a single relation $r_h$ and a reward value, $z_T \in \{1, 0, +1\}$ is assigned to all states in the episode, where

$$z_T = \begin{cases} 1 & \text{if } r_h \text{ is a known relation, but is not the target relation} \\ 0 & \text{if } r_h \text{ is an invented relation} \\ 1 & \text{if } r_h \text{ is the target relation} \end{cases} \tag{4.22}$$

- **Policy Network**: The policy value network $(\boldsymbol{\rho}, v) = f_\theta(s)$, is a neural network with $\theta$ representing its parameters. Given the current state $s$ as input, the network outputs both an action probability distribution $\boldsymbol{\rho}$ and a state value $v$. MCTS utilizes this policy network to guide its simulations and produce a vector $\boldsymbol{\pi}$ that represents the improved search probabilities of the available actions. During training, the policy network is optimized to minimize the difference between the predicted state value $v$ and the actual reward $z$ received after each episode. Additionally, it is also trained to maximize the similarity between two probability distributions $\boldsymbol{\rho}$ and $\boldsymbol{\pi}$.

$$\mathcal{L} = (z - v)^2 - \boldsymbol{\pi}^\intercal \log \boldsymbol{\rho} + a \|\theta\|^2 \tag{4.23}$$

**Dynamic Rule Memory Module** As mentioned earlier, the combination of deep reinforcement learning and dynamic rule memory enables logical reasoning and rule extraction. While we have already discussed deep reinforcement learning, in this section we will focus on introducing the dynamic rule memory module.

Dynamic rule memory module aims to store and update candidate rules and interact with the reasoning agent during training, whose design can be interpreted as two hash tables: one hash table $D_{rl}$ in the form of $(r_{B,0}, r_{B,1}) : r_h$ is used to memorize the candidate rules, and another hash table $D_{rls}$ in the form of $(r_{B,0}, r_{B,1}) : \mathit{score}$ to track the rule scores. Specifically, the MCTS obtains a rule $a_H \leftarrow (a_{B,0}, a_{B,1})$ from the hash table $D_{rl}$, and replaces the current relation pair $(a_{B,0}, a_{B,1})$ in the path with the rule head $a_H$. If the rule is not found in the hash table $D_{rl}$, the MCTS adds an

invented relation to the buffer $B_{\text{unkn}}$ and uses it as the rule head $a_H$. During training, the dynamic rule memory module is updated to keep track of the score of each rule, where

$$\text{score} = \begin{cases} v_0 & \text{if } a_B \in D_{rl}, \forall r \in a_B, r \in M \\ v_1 & \text{if } a_B \in D_{rl}, a_H \in M, \exists r \in a_B, r \in N \\ v_2 & \text{if } a_B \in D_{rl}, a_H \in N \\ v_3 & \text{if } a_B \notin D_{rl}, \forall r \in a_B, r \in M \\ v_4 & \text{if } a_B \notin D_{rl}, \exists r \in a_B, r \in N \end{cases} \tag{4.24}$$

We have $v_0 > v_1 > 0 > v_2 > v_3 > v_4$. $M$ denotes the set of known relations and $N$ denotes the set of invented relations. This way R5 penalizes the usage of invented relations to prevent overfitting.

Despite providing a promising approach for learning first-order logical rules, R5 faces limitations in its scalability and ability to generalize to the KG completion task. This is due to the requirement for pre-sampling the paths that entail the query, which can be impractical even for small-scale KGs due to the large number of paths that need to be considered. Furthermore, R5 employs a hard decision mechanism for merging a relation pair into a single relation, which makes it challenging to handle the uncertainty that exists in KGs. For example, given the rule body $\mathit{hasAunt}(x, z) \wedge \mathit{hasSister}(z, y)$, both $\mathit{hasMother}(x, y)$ and $\mathit{hasAunt}(x, y)$ can be derived as the rule head. The inaccurate merging of a relation pair may lead to error propagation when generalizing to longer paths.

**Summary** Learning-based methods provide a flexible and adaptable approach to combining symbolic logical rule learning with neural network models by leveraging various learning paradigms. The effectiveness of the rule learning process depends heavily on the choice of a learning algorithm, as these methods rely on the algorithm to discover rules from the data. Hence, it is crucial to carefully select an appropriate algorithm that is tailored to the specific application and characteristics of the data being analyzed.

## 4.4 RLogic: A Representation-learning Based Model For Logical Rule Learning

Although many efforts have been made to learn logical rules automatically, there are two limitations to the existing studies. Most existing methods entirely rely on observed rule instances to define the score function for rule evaluation. They are unable to mine rules that have no support from rule instances. For example, due to the lack of supportive evidence, the following rule cannot be learned from Fig. 4.3:

$$\delta_3 \coloneqq \text{hasUncle}(x, y) \leftarrow \text{hasGrandma}(x, z) \wedge \text{hasSon}(z, y) \tag{4.25}$$

Note that the number of rule instances is extremely large for a big KG, scalability is another central challenge. Instead of completely relying on rule instances for rule evaluation, RLogic [23] proposes to learn logical rules directly at the *schema level* via a representation learning-based model. We have introduced template rules (i.e., logical rules at the schema level) and ground rules (i.e., instance rules) in Sec. 2.3. As a small amount of sampled closed paths is enough for training such a model, it greatly improves the efficiency. To push deductive reasoning deeper into rule learning, RLogic breaks a big sequential model into small atomic models in a recursive way, which is essential for us to detect rules without direct support from rule instances.

### 4.4.1 Framework of RLogic

RLogic aim to learn chain-like Horn rules in the following form:

$$r_h(x, y) \leftarrow r_{b_1}(x, z_1) \wedge \cdots \wedge r_{b_n}(z_{n-1}, y) \tag{4.26}$$

We previously introduced chain-like Horn rules in Sec. 2.3, where $r_h(x, y)$ is called **rule head** and $r_{b_1}(x, z_1) \wedge \cdots \wedge r_{b_n}(z_{n-1}, y)$ is called **rule body**. Combining rule head and rule body, we denote a Horn rule as $(r_h, \mathbf{r_b})$ where $\mathbf{r_b} = [r_{b_1}, \ldots, r_{b_n}]$. The fundamental idea of RLogic is to assign a plausibility score $s(r_h, \mathbf{r_b})$ to each rule $(r_h, \mathbf{r_b})$ in **rule space**. $s(\cdot)$ is called the score function.

#### 4.4.1.1 From Counting-based Measure to Model-based Measure for Rule Evaluation

The ratio that a body path can be closed is usually utilized to define the score function $s(r_h, \mathbf{r_b})$ (e.g., the confidence for association rule mining as discussed in Sec. 4.2.2 and the percentage of triples of the rule head to be satisfied for Neural-LP as discussed in Sec. 4.3.1.2. During rule extraction, the top $k$ rules with the highest score will be selected as the learned rules.

**Existing Counting-based Measure for Rule Evaluation** Among several possible measures, *confidence* is the most representative one widely used by association rule mining. As mentioned in Sec. 2.3.2, a chain-like Horn rule instance can be easily identified as a "closed path" in a KG. Here, the rule body represents a path in the KG, and the rule head represents the target relation that connects the starting and ending entities in the rule body. Consequently, the confidence measure is defined as the ratio of the number of times the target relation can close a body path. Given an arbitrary rule defined in Eq. 4.3, its confidence can be calculated as:

$$s(r_h, \mathbf{r_b}) = \frac{|\{(x, z_1, \ldots, z_{n-1}, y) : r_h(x, y) \leftarrow r_{b_1}(x, z_1) \wedge \cdots \wedge r_{b_n}(z_{n-1}, y)\}|}{|\{(x, z_1, \ldots, z_{n-1}, y) : r_{b_1}(x, z_1) \wedge \cdots \wedge r_{b_n}(z_{n-1}, y)\}|} \tag{4.27}$$

where the numerator is the number of its rule instances (i.e., closed paths) and the denominator is the number of its body instances (i.e., relation paths). We have discussed that a chain-like Horn rule instance can be considered as a "closed path" in a graph, where the rule body corresponds to a path in the KG (called "relation path"), and the rule head corresponds to the edge connecting the starting entity and ending entity in the rule body. The confidence measure is purely based on counting on observed rule instances, and it is very sensitive to the quality and completeness of the KG. For example, if the KG contains noises, it might bring in false-positive rules. If the KG contains similar entities and relations with different names (e.g., "Anthony" vs. "Tony"), it might miss some rules due to undercounting the rule instances. Similarly, if the KG misses any facts, it might miss some rules due to the low support of the rule instances.

**Proposed Model-based Measure for Rule Evaluation** As discussed above, the calculation of confidence entirely relies on observed rule instances, which is sensitive to the quality of the KG. In addition, enumerating rule instances is usually time-consuming. Instead, we propose a new measure

for rule evaluation based on the model-based probability that the rule head can be predicted by the rule body:

$$s(r_h, \mathbf{r_b}) = q(r_h = r_i | \mathbf{r_b}) \tag{4.28}$$

For a rule with body length $l$, we need a tensor with $|\mathcal{R}|^{(l+1)}$ dimension to store all the probabilities. It is too expensive and impossible to get enough data points to estimate each entry in the tensor using counting-based approach. Model-based approaches can alleviate this issue by fitting a function with observed instances, which can infer the probability for any entry with the body and head inputs. For example, a sequential model, such as RNN, is a natural option to learn $q(r_h = r_i | \mathbf{r_b})$ by encoding the relation sequence $\mathbf{r_b}$ into a representation with fixed dimension. However, a sequential model such as RNN still heavily relies on the observed instances.

#### 4.4.1.2 Framework - RLogic

In RLogic, we propose to incorporate the *deductive nature* of the logical rules into the modeling process, which significantly improves the performance due to its capability in combining learning and reasoning, especially when the evidence is sparse. Deductive nature, as one of the most significant properties of logical rules, allows us to decompose the inference of long rules on the basis of the shorter rules. Given a short Horn rule that is in the form $r_k \leftarrow r_i \wedge r_j$, we can reduce the long logical body $r_{b_1} \wedge r_{b_2} \wedge \cdots \wedge r_{b_n}$ by replacing the relation pair $r_i \wedge r_j$ with their head $r_k$. By recursively applying different short Horn rules to a relation path (body-based path), it will be transformed into a single head at the end. Following the same idea, we can use $q(r_h | r_i, r_j)$ to compute $q(r_h | \mathbf{r_b})$ in a recursive way. For example, given a relation path $[r_{b_1}, r_{b_2}, r_{b_3}]$, $q(r_h | r_{b_1}, r_{b_2}, r_{b_3})$ can be computed as follows if we follow the order from left to right to reduce the path:

$$q(r_h | r_{b_1}, r_{b_2}, r_{b_3}) = \sum_k q(r_h | r_k, r_{b_3}) q(r_k | r_{b_1}, r_{b_2}) \tag{4.29}$$

In other words, we reduce the first two relations $r_{b_1}, r_{b_2}$ into some possible relation $r_k \in \mathcal{R} \cup \{r_0\}$, where $r_0$ denotes an unknown relation. Then we further reduce the intermediate relation $r_k$ with $r_{b_3}$ into the head relation $r_h$. We aggregate all the possible cases for $r_k$, and the final probability for the body to predict the head is a weighted average over each case. As for every step we only need

to model a sequence with length 2, this significantly reduces the computational burden caused by long sequence modeling, such as RNN.

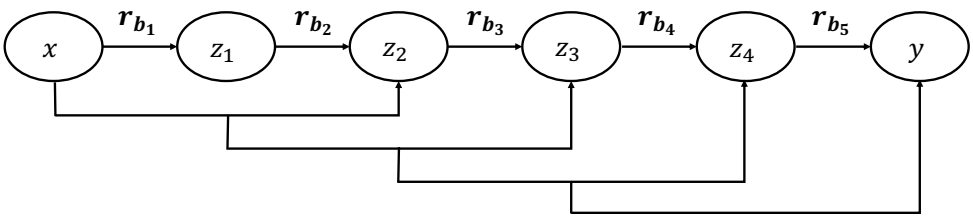
Although we can follow logical deduction to reduce a relation path into one single head, this head relation may not always be observed due to the sparsity of real-world KGs. We thus introduce $p(r_t|r_h)$ to bridge the gap between the logical rule-based "ideal prediction" and the "real observation" in KGs, which is the probability that we can observe a triple with relation $r_t$ given $r_h$ is true for the same head and tail entity. The the ratio that a relation path $\mathbf{r_b}$ can be closed with relation $r_t$ can be defined as:

$$p(r_t|\mathbf{r_b}) = \sum_h p(r_t|r_h)q(r_h|\mathbf{r_b}) \tag{4.30}$$

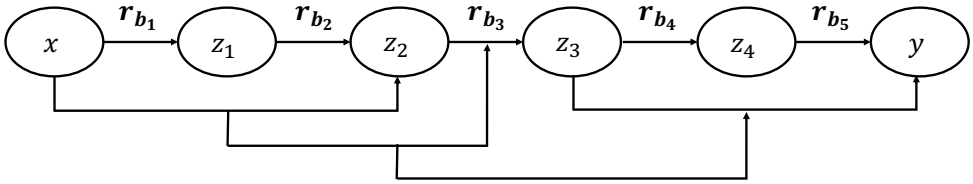
Following the proposed idea, we introduce a (1) *recursive relation path reduction module* for relation path encoding and a (2) *close ratio predictor* to predict the ratio that a path will close.

In our model, each relation $r_i$ is associated with a learnable embedding vector $\mathbf{r_i} \in \mathbb{R}^d$. With the representation of relations, RLogic can identify similar relations like *GrandPa* and *Grand-Father* even though they are lexically different. It is important to note that it is not always possible to predict a single relation from a relation path. For example, consider the relation path [hasMother, hasMother, hasDaughter], which is predicted to be hasAunt, but there is a possibility that the relation hasAunt is not present in the KG. To accommodate unseen relations, we introduce a "null" predicate into the prediction set $\mathcal{R}$, which is denoted as $r_0$. The score function $s(r_h, \mathbf{r_b})$ in Eq. 4.28 can be computed based on the representation of relations.

**(1) Recursive Relation Path Reduction ($q(r_h|\mathbf{r_b})$).** The goal here is to model the score for any relation path $\mathbf{r_b}$ for a given head relation $r_h$, which can be interpreted as the probability of the head to be true if the body is true. As the score function has $|\mathcal{R}|^{l+1}$ entries for a rule with a length $l$ body, it is impractical to estimate each entry independently. Model-based approach can capture the correlation between these entries with a much smaller parameter space. A natural sequence model is RNN and Transformer, but they still heavily relied on the observed closed paths. If no instances are observed for a rule, there is no way for these models to produce a high score for the body and head combination.




(a) Left-wise decomposition




(b) Decomposition following an irregular order

Figure 4.5: Different order to deduct a relation path. (a) left-wise decomposition; (b) irregular decomposition.

We propose to reduce long relation path into shorter ones by pushing reasoning into the prediction. Specifically, we recursively reduce two adjacent relations in the body into one, until two relations are left. In this case, we only need to model the probability for a length-2 body to a relation head, $q(r_k|r_i, r_j)$, for which it is easy to collect more evidences due to their relative short length.

This idea requires (1) deciding the order to deduct a relation path (i.e., selecting the appropriate relation pair in each step) and (2) learning the probability of predicting a single relation from a relation pair (i.e., $q(r_k|r_i, r_j)$).

- **Deciding the next pair of relations to deduct.** As shown in Fig. 4.5, there are many different ways to deduct a relation path. Determining the best order involves searching over a potentially large problem space. Given a relation path with length $l$, there are $C_{l-1} = \prod_{k=2}^{l-1} \frac{l-1+k}{k}$ (Catalan number) different ways to decompose the relation path. To alleviate the high computational complexity, instead of enumerating all possible deduction orders to find the global optimal, we adopt a greedy algorithm to select the optimal relation pair at each step, which reduces the complexity into $(l-1) + (l-2) + \cdots + 1$.

Considering that the ground truth deduction order is unavailable in our problem, we propose to use entropy to measure the uncertainty of the reduction prediction. By minimizing entropy, the confidence of predictions can be encouraged. The entropy of a relation pair $(r_i, r_j)$ implying $r_k$ is defined as follows:

$$E(r_k \leftarrow (r_i, r_j)) = \sum_{r_k \in R} -q(r_k|r_i, r_j) \log q(r_k|r_i, r_j) \tag{4.31}$$

When the entropy is lower, we can be more certain that a relation can be predicted by the relation pair. To maximize the confidence of deduction, we select the relation pair with the lowest entropy at every step.

- **Probabilistic deduction to a single relation from a relation path.** Given the order to deduct a relation path, the next step is to learn $q(r_k|r_i, r_j)$ to deduce the relation path recursively. $q(r_k|r_i, r_j)$ can be approximated via a multilayer perceptron (MLP) classifier. The MLP classifier $f_\theta(\mathbf{r_i}, \mathbf{r_j})$ is a two-layer fully-connected neural network with parameters $\theta$. It takes the embedding of predicate $r_i, r_j$ as input, and outputs the probability of predicting each relation in the KG, including the "null" predicate (i.e., unknown relation). Considering that $q(r_k|r_i, r_j)$ follows categorical distribution, the MLP classifier uses softmax as the activation function for the final layer.

  For a rule body $\mathbf{r}_b = [r_{b_1}, r_{b_2}, \ldots, r_{b_l}]$, assuming $r_{b_1}, r_{b_2}$ is selected to deduce first, they lead to the prediction of each relation in KG with different probability (i.e., $q(r_k|r_{b_1}, r_{b_2})$). For example, given the rule body $hasAunt(x, z) \wedge hasSister(z, y)$, both $hasMother(x, y)$ and $hasAunt(x, y)$ can be derived as the rule head. We propose to encode the relation pair $r_{b_1}, r_{b_2}$ as the weighed average over different relations, including the unknown relation $r_0$:

$$\tilde{\mathbf{r}} = \sum_{k=0}^{|\mathcal{R}|} q(r_k|r_{b_1}, r_{b_2}) \cdot \mathbf{r_k}, \tag{4.32}$$

  The relation path encoder will recursively deduce the next best pairs, where the pair may involve the newly generated relation. For example, a relation path in their embedding $[\mathbf{r}_{b_1}, \mathbf{r}_{b_2}, \mathbf{r}_{b_3}]$ may be deduced to $[\tilde{\mathbf{r}}, \mathbf{r}_{b_3}]$, and then the final representation for the entire relation path $\tilde{\mathbf{r}}^* = \sum_{k=0}^{|\mathcal{R}|} q(r_k|\tilde{\mathbf{r}}, \mathbf{r}_{b_3}) \cdot \mathbf{r_k}$.

When there are only two relations left in the rule body, we can use the MLP to predict the probability for it to reduce to any relation $r_h$. For example, a relation path in their embedding $[\mathbf{r}_{b_1}, \mathbf{r}_{b_2}, \mathbf{r}_{b_3}]$ may be reduced to $[\tilde{\mathbf{r}}, \mathbf{r}_{b_3}]$, and the probability for the body to imply any relation can be computed as:

$$q(\cdot|\mathbf{r}_{b_1}, \mathbf{r}_{b_2}, \mathbf{r}_{b_3}) = q(\cdot|\tilde{\mathbf{r}}, \mathbf{r}_{b_3}) = f_\theta(\tilde{\mathbf{r}}, \mathbf{r}_{\mathbf{b_3}}) \tag{4.33}$$

**(2) Close Ratio Predictor ($p(r_t|r_h)$).** Although we can follow logical deduction to reduce a relation path into one single head relation probabilistically, this head relation may not always be observed due to the sparsity of real-world KGs. To bridge the gap between "ideal prediction" following logical rules and "real observation", the close-ratio predictor is proposed to predict the ratio that a path will close. A two-layer fully-connected neural network (MLP) is introduced to model $p(r_t|r_h)$. In particular, it uses ReLU as the activation function for the first layer and adds sigmoid as the activation function for the second layer. The "weighted average" $\tilde{\mathbf{r}}$ learned by the relation path encoder at the final step and the embedding $\mathbf{r_t}$ are taken as input to decide the close ratio.

#### 4.4.1.3 Model Training

This section discusses the training procedure for RLogic. We first introduce our proposed closed path sampler for training data generation. Then we give the objective functions for training relation path encoder and close ratio predictor.

**Closed Path Sampler for Training Data Generation.** Rather than enumerate all closed paths in KG, we utilize a closed path sampler to sample only a small portion of closed paths to train the model. We propose a random walk [115] based procedure to efficiently sample the close paths. Formally, given a source entity $x_0$, we simulate a random walk of fixed length $n$. Let $x_i$ denote the $i$th node in the walk. Nodes $x_i$ are generated by the following distribution:

$$p(x_i = e_i|x_{i-1} = e_j) = \begin{cases} \frac{1}{|\mathcal{N}(e_j)|}, & \text{if } (e_i, e_j) \in E \\ 0, & \text{otherwise} \end{cases} \tag{4.34}$$

where $|\mathcal{N}(e_j)|$ is the neighbor size of entity $e_j$. Different from a random walk, each time after we sample the next node $x_i$, we add all edges which can directly connect $x_0$ and $x_i$ in KG to construct

the closed paths.

**Objective function for training relation path encoder** The relation path encoder aims to predict a head relation $r_h$ for a given relation path $\mathbf{r_b}$. Each sampled closed path can be regarded as a positive example, whose target relation gives the ground truth of head relation $r_h$. Negative examples can be generated by corrupting positive examples. Note that KGs operate under the Open World Assumption (OWA). A statement that is not contained in the KG is not necessarily false. It is just unknown. Rather than assuming negative examples are necessarily false, we can only infer that they are more invalid than those positive ones. To make the scores of positive examples higher than those of negative ones, we employ a pairwise margin-based ranking loss function to learn $q(r_k|\mathbf{r_b})$.

$$\sum_{(r_h,\mathbf{r_b})\in\mathcal{P}} \sum_{(r_h{}',\mathbf{r_b})\in\mathcal{N}(r_h,\mathbf{r_b})} [q(r_h|\mathbf{r_b}) + \gamma - q(r_h{}'|\mathbf{r_b}))]_+ \tag{4.35}$$

where $[x]_+ = \max\{0, x\}$ and the scoring function $s(\cdot)$ can be parameterized by the relation path encoder. $\gamma > 0$ is the margin hyperparameter separating positive and negative closed paths. To reduce the effects of randomness, we sample multiple negative examples for each positive closed path. We denote the set of negative samples of a closed path $(r_h, \mathbf{r_b})$ as $\mathcal{N}(r_h, \mathbf{r_b})$, which is constructed by replacing the head relation of the closed path $(r_h, \mathbf{r_b})$ with a relation sampled randomly from relation set $R$:

$$\mathcal{N}(r_h, \mathbf{r_b}) \subset \{(r_h{}', \mathbf{r_b})|r_h{}' \in R\}\} \tag{4.36}$$

**Objective function for training close ratio predictor** Close ratio predictor bridges the gap between "ideal prediction" following logical rules and "real observation" by predicting the ratio that a path $\mathbf{r_b}$ will close. Each closed path in KG can be regarded as a positive example while the paths that fail to be closed by any relation are regarded as negative examples. The training objective for learning $p(r_t|r_h)$ can be formulated as the binary cross-entropy loss:

$$\sum_{(r_t,r_h)\in\mathcal{P}} \log p(r_h|\mathbf{r_b}) + \sum_{(r_t,r_h)\in\mathcal{N}} \log(1 - (p(r_h|\mathbf{r_b})) \tag{4.37}$$

where $\mathcal{P}$ stores only positive examples and $\mathcal{N}$ stores only negative examples.

#### 4.4.1.4 Rule Extraction

To recover logical rules from RLogic, we calculate the score $s(r_h, \mathbf{r_b})$ for each rule $(r_h, \mathbf{r_b})$ in rule space when the model has finished training. Unlike most of the existing neural symbolic methods for rule learning, such as Neural-LP and RNNLogic, which can only rank rules conditioned on the head relation, our method can rank rules globally. With such scores, we can select the most important rules to interpret the whole KG rather than infer a certain relation. The detailed procedure to extract rules is presented below. Given a candidate rule $(r_h, \mathbf{r_b})$, we reduce the rule body $\mathbf{r_b}$ into a single head $r_k$ by recursively merge relation pairs in path $\mathbf{r_b}$ according to $q(r_k|r_i, r_j)$. Every step, the relation pair $(r_i, r_j)$ with minimum entropy will be selected and replaced by an intermediate representation $\tilde{\mathbf{r}}$ according to Eq. (4.32). At the end of the deduction, we obtain the vector $[q(r_0|\mathbf{r_b}), q(r_1|\mathbf{r_b}), \ldots, q(r_{|R|}|\mathbf{r_b})]$ where $q(r_k|\mathbf{r_b})$ are the score of rule $(r_k, \mathbf{r_b})$. Top $k$ rules with highest score will be selected as learned rules.

### 4.4.2 Connection to Existing Approaches

Note that logical rule is a schema-level concept, while only instance-level evidence can be directly observed from KGs. To bridge this gap, the frequency of rule instances is commonly used to determine the plausibility of logical rules. For example, traditional methods such as association rule mining [42, 74], define the score as the ratio of the total number of rule instances to the total number of body instances for the corresponding rule. Then the rule mining process is to search over the rule space and select the rules with the highest score. For example, given the KG in Fig. 4.3, the rule space with length 2 to predict head relation $r_2$ (hasGrandma) can be represented as a tree structure, and each path from the root to a leaf corresponds to a rule body. Two rules with high scores can be learned in Fig. 4.3 ($\delta_1$ has a score of 0.75 and $\delta_2$ has a score of 1), which are:

$$\begin{aligned} \delta_1 &:= \text{hasGrandma}(x, y) \leftarrow \text{hasMother}(x, z) \wedge \text{hasMother}(z, y) \\ \delta_2 &:= \text{hasUncle}(x, y) \leftarrow \text{hasMother}(x, z_1) \wedge \text{hasMother}(z_1, z_2) \wedge \text{hasSon}(z_2, y) \end{aligned} \tag{4.38}$$

In recent years, there have been proposals for neural symbolic approaches [136, 103, 137] to

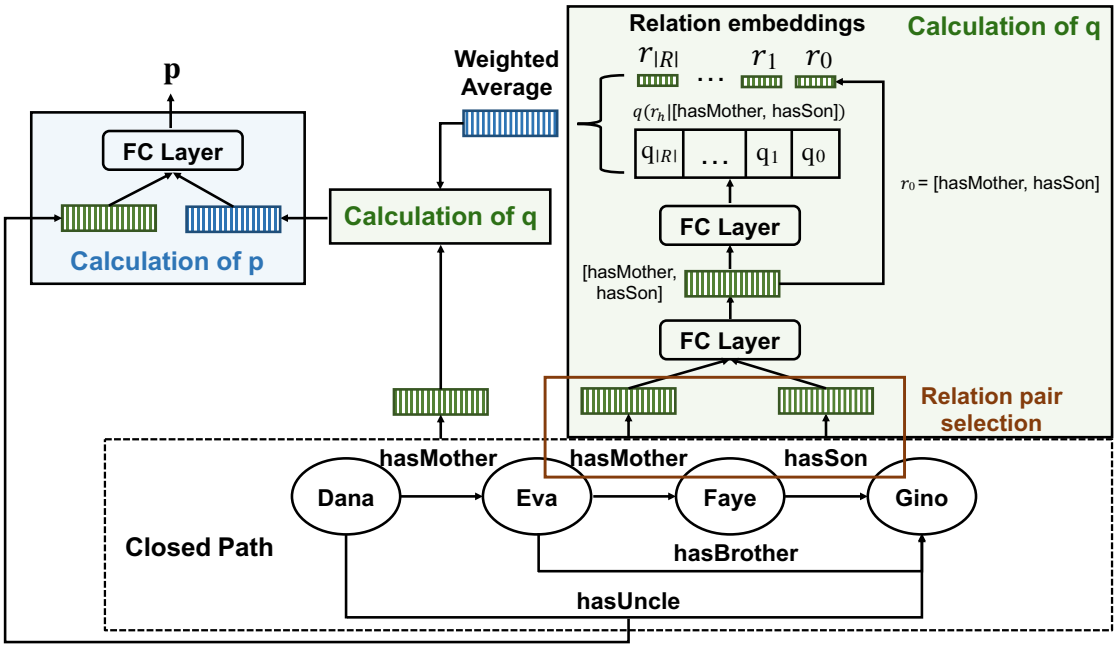



Figure 4.6: Illustration of the RLogic framework. It contains two main components: (1) a relation path encoder to model $q(r_h|\mathbf{r_b})$; and (2) a close ratio predictor to model $p(r_t|r_h)$. Given a path $\mathbf{r_b}$, the relation path encoder reduces $\mathbf{r_b}$ into a single head $r_h$ by recursively merge relation pairs in path $\mathbf{r_b}$ according to $q(r_h|r_i, r_j)$. Then, the close ratio predictor bridges the gap between "ideal prediction" following logical rules and "real observation" by predicting the ratio that the relation path $\mathbf{r_b}$ will close.

learn logical rules by integrating symbolic logical rule learning with neural network models. A notable example of such an approach is Neural-LP, which enables the soft counting of rule instances through differentiable tensor multiplication rather than discrete counting. To identify useful rules from the rule space, these methods also search for rule bodies (or proof paths) that maximize the observations (i.e., triples) of the rule head.

Although many efforts have been made to learn logical rules automatically, there are two limitations to the existing studies. First, most existing methods entirely rely on observed rule instances to define the score function for rule evaluation. They are unable to mine rules that have no support from rule instances. For example, due to the lack of supportive evidence, the following rule cannot be learned from Fig. 4.3:

$$\delta_3 := \text{hasUncle}(x, y) \leftarrow \text{hasGrandma}(x, z) \wedge \text{hasSon}(z, y) \tag{4.39}$$

Note that the number of rule instances is extremely large for a big KG, scalability is another central challenge. Instead of completely relying on rule instances for rule evaluation, we propose to learn logical rules directly at the schema level via a representation learning-based model. The score of a rule can be calculated based on the learned predicate representations. In this way, a rule can be evaluated without direct support from rule instances. Since a small amount of sampled rule instances is enough for training such a model, it greatly improves the efficiency.

Second, a majority of existing methods learn logical rules by assuming that logical rules are independent of each other, which seriously contradicts the *deductive nature* of logical rules. The deductive nature of logical rules describes the capability to combine existing rules for deriving new rules. For example, given two short rules $\delta_1$ and $\delta_3$, we can infer a long rule $\delta_2$. The inference of $\delta_2$ can be decomposed into recursive steps as shown in Fig. 4.7. Starting from the first two predicates, we can derive an intermediate conclusion $\mathit{hasGrandma}(x, z_2)$ following $\delta_1$. Then, by replacing the first two predicates with the derived relation, we rewrite the rule body as $\mathit{hasGrandma}(x, z_2) \wedge \mathit{hasMother}(z_2, y)$ and derive the final conclusion according to $\delta_3$. Because deductive nature describes the logical dependency among rules, it can be considered as "higher-order constraints" over rules, which is essential for us to validate rules without enough support from rule instances. For example, although we cannot rely on rule instances to evaluate $\delta_3$ due to the

lack of supportive evidence, as long as we can see evidence from Fig. 4.3 to support $\delta_1$ and $\delta_2$, by pushing deduction deeper into $\delta_2$, $\delta_3$ is forced to be true to make $\delta_2$ true. To incorporate deductive nature into rule learning, we propose to break a big sequential model to small atomic models in a recursive way following logical deduction. We summarize the advantage of RLogic compared to SOTA below.

- **Stronger generalization ability**: Rule is an abstract level concept, which consists of only predicates and variables. Since we cannot directly observe abstract rules in KGs, existing methods, such as AMIE, Neural-LP, rely on the rule instances to define the score function of rules. Such definitions lack generalizability. Instead, RLogic proposed to learn the representation of predicates (i.e., relations) to define the score of a rule as shown in Eq. 4.28 and thus show stronger generalization ability. In this way, a rule can be detected without the need to see a rule instance. Besides, RLogic can identify similar predicates like *Mother* and *Mum* even though they are lexically different.

- **Pushing deductive reasoning deeper into rule learning**: Most existing methods, such as Neural-Lp, NTPs and RNNLogic learn logical rules by assuming that logical rules are independent of each other, which seriously contradicts the deductive nature of logical rules. Instead, RLogic is able to follow logical deductions to validate a rule by pushing deductive reasoning deeper into rule learning, which is critical when a rule lacks supporting evidence. For example, although we cannot rely on rule instances to evaluate $\delta_3$ due to the lack of supportive evidence, as long as we can see evidence from Fig. 4.3 to support $\delta_1$ and $\delta_2$, by pushing deduction deeper into $\delta_2$, $\delta_3$ is forced to be true to make $\delta_2$ true.

### 4.4.3 Experiments

#### 4.4.3.1 Datasets.

Four widely used benchmark datasets are adopted to evaluate our proposed RLogic, which includes WN18RR [35], FB15K-237 [121], YAGO3-10 [116] and Family [53]. The detailed statistics of the datasets are summarized in Table 4.2.

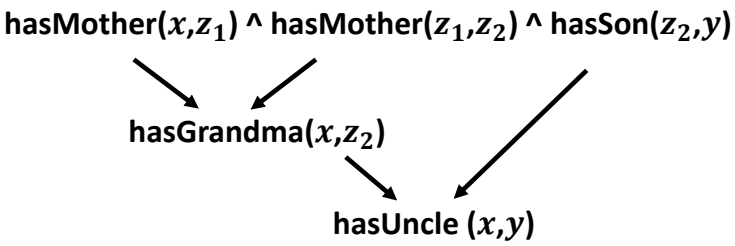



Figure 4.7: Deduction from $\delta_1$ and $\delta_3$ to $\delta_2$.

Table 4.2: Data statistics.

| Dataset | # Data | # Relation | # Entity |
|---|---|---|---|
| FB15K-237 | 310,116 | 237 | 14,541 |
| WN18RR | 93,003 | 11 | 40,943 |
| YAGO3-10 | 1,089,040 | 37 | 123,182 |
| Family | 28,356 | 12 | 3007 |

#### 4.4.3.2 Quality of Learned Rules in Terms of KG Completion Task

KG completion is a classic task widely used by logical rule learning methods such as Neural-LP [136], DRUM [103], and NLIL [137] to evaluate the quality of learned rules. An existing algorithm called *forward chaining* [104] can be used to derive missing facts from logical rules efficiently. The top 2400 rules with the highest score learned by RLogic are selected for the KG completion task.

**Evaluation Metrics.** RLogic mask the head or tail entity of each test triple and require each method to predict the masked entity. During the evaluation, the filtered setting [13] and three evaluation metrics, i.e., Hit@1, Hit@10, and MRR are used. To break ties for triples with the same score, RLogic follows *random protocol* [118] to rank the triples with the same score.

**Comparing with Other Methods.** RLogic is evaluated against SOTA algorithms, including (1) traditional KG embedding (KGE) methods (e.g., TransE [13], DistMult [135], ConvE [35], ComplEx [122] and RotatE [117]); (2) logical rule learning methods (e.g., Neural-LP [136],

Table 4.3: Transductive link prediction. The bold numbers represent the best performances among all methods while the underlined numbers represent the best performances among all rule learning methods.

| Category | Model | WN18RR | | | FB15K-237 | | | YAGO3-10 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | MRR | Hit@1 | Hit@10 | MRR | Hit@1 | Hit@10 | MRR | Hit@1 | Hit@10 |
| KGE | TransE | 0.23 | 2.2 | 52.4 | 0.29 | 18.9 | 46.5 | 0.36 | 25.1 | 58.0 |
| | DistMult | 0.42 | 38.2 | 50.7 | 0.22 | 13.6 | 38.8 | 0.34 | 24.3 | 53.3 |
| | ConvE | 0.43 | 40.1 | 52.5 | **0.32** | 21.6 | 50.1 | 0.36 | 26.5 | 55.6 |
| | ComplEx | 0.44 | 41.0 | 51.2 | 0.24 | 15.8 | 42.8 | 0.34 | 24.8 | 54.9 |
| | RotatE | **0.47** | 42.9 | **55.7** | **0.32** | **22.8** | **52.1** | **0.49** | **40.2** | **67.0** |
| Rule Learning | Neural-LP† | 0.38 | 36.8 | 40.8 | 0.24 | 17.3 | 36.2 | - | - | - |
| | NLIL† | 0.30 | 20.1 | 33.5 | 0.25 | 13.8 | 32.4 | - | - | - |
| | DRUM† | 0.38 | 36.9 | 41.0 | 0.23 | 17.4 | 36.4 | - | - | - |
| | AMIE | 0.36 | 39.1 | 48.5 | 0.23 | 14.8 | 41.9 | 0.25 | 20.6 | 34.3 |
| | RNNLogic (w/o emb)‡ | 0.46 | 41.4 | 53.1 | 0.29 | 20.8 | 44.5 | - | - | - |
| | RLogic | **0.47** | **44.3** | 53.7 | 0.31 | 20.3 | 50.1 | 0.36 | 25.2 | 50.4 |

† Neural-LP, NLIL and DRUM exceeds the capacity of our machines on YAGO3-10 dataset

‡ Results on RNNLogic are taken from the original papers.

Table 4.4: Inductive link prediction. The bold numbers represent the best performances among all methods.

| Model | WN18RR | | | FB15K-237 | | | YAGO3-10 | | |
|---|---|---|---|---|---|---|---|---|---|
| | MRR | Hit@1 | Hit@10 | MRR | Hit@1 | Hit@10 | MRR | Hit@1 | Hit@10 |
| KGE† | - | - | - | - | - | - | - | - | - |
| Neural-LP‡ | 0.23 | 20.3 | 33.1 | 0.14 | 9.3 | 27.6 | - | - | - |
| DRUM‡ | 0.23 | 20.5 | 34.4 | 0.16 | 10.8 | 29.3 | - | - | - |
| AMIE | 0.32 | 33.6 | 45.5 | 0.19 | 13.9 | 38.0 | 0.21 | 15.8 | 30.1 |
| RLogic | **0.43** | **42.1** | **50.8** | **0.29** | **18.4** | **48.7** | **0.32** | **22.8** | **47.2** |

† KGE methods are not applicable in an inductive setting.

‡ Neural-LP, NLIL, and DRUM exceed the capacity of our machines on YAGO3-10 dataset

* Results on RLogic are taken from the original papers.

NLIL [137], DRUM [103], AMIE [42] and RNNLogic [94]). The comparison results are presented in Table 4.9. We can observe that: (1) Although RLogic is not designed specifically for KG completion tasks, compared with traditional KGE models, it still achieves comparable results on all datasets; (2) RLogic outperforms most logical rule learning methods with significant performance gain in most cases; (3) RNNLogic shows great performance over KG completion tasks because it jointly trains a powerful reasoning predictor to predict missing links. To fairly compare with RNNLogic, RLogic+ is proposed to incorporate an improved reasoning predictor for prediction. The results of RLogic+ on KG completion task are shown in the following section.

**Inductive Link Prediction.** It is important to note that comparing logical rule learning methods with KGE methods solely on transductive KG completion tasks is not fair. Different from KGE methods, which are not capable of reasoning on unseen entities, logical rules are more powerful in an inductive setting. The results for the inductive link prediction experiment are shown in Table 4.4. We observe that all rule learning algorithms still achieve similar performances as they did in the transductive settings.

**RLogic+.** Unlike other baseline methods, RLogic directly learns rules without enhancing KG

Table 4.5: Combine learned rules with KG embedding for KG completion task.

| Model | WN18RR | | | FB15K-237 | | | YAGO3-10 | | |
|---|---|---|---|---|---|---|---|---|---|
| | MRR | Hit@1 | Hit@10 | MRR | Hit@1 | Hit@10 | MRR | Hit@1 | Hit@10 |
| RNNLogic+ (with emb.)† | 0.51 | **47.1** | 59.7 | 0.35 | 25.8 | 53.3 | - | - | - |
| RLogic+ | **0.52** | 46.6 | **60.4** | **0.55** | **51.1** | **64.3** | **0.53** | **42.6** | **70.3** |

† Results on RLogic are taken from the original papers.

completion tasks as a side product. Although RLogic is able to generate high-quality logic rules, its performance over KG completion task is severely limited by the coverage of rules and the incompleteness of KGs due to the lack of a good reasoning predictor. Following UniKER [24], we resolve the KG sparsity issue by adding additional triples with the high score using RotatE [117], and then conduct forward chaining to predict missing triples. The results are shown in Table 4.5. We can see that with the help of KG embedding, the performance of RLogic+ over KG completion task gets significantly improved on all datasets, especially on FB15k237.

#### 4.4.3.3 Quality of Learned Rules in Terms of Rule Head Prediction Task.

In addition to the promising performance in terms of the KG completion task, *we also directly evaluate the correctness of rules learned by each system on Family dataset*. In particular, we propose a novel task **rule head prediction** to predict the heads of a set of rule bodies. *Human annotations are used to label the rule bodies for ground truth preparation.* Mean Average Precision (MAP) is taken as an evaluation metric.

**Learn Equal-length Rules** We evaluate RLogic against a number of logical rule learning methods. Each method is given a set of closed paths with length 2 as training data and required to predict the rule head of *equal-length* rule bodies. The comparison results are presented in Fig. 4.8. We observed that RLogic outperforms all other logical rule learning methods with significant gain.

**Learn Longer Rules** RLogic incorporates the deductive nature into rule learning and thus can learn longer rules with only shorter closed paths observed in the training stage. In this experiment, we allow only closed paths with length 2 to be observed in the training stage and require each

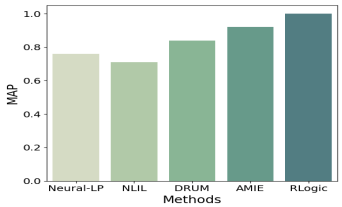
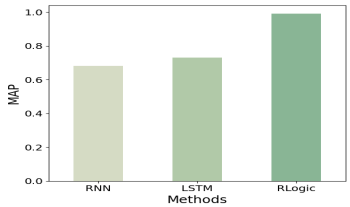



Figure 4.8: Equal-length rule detection comparison on Family dataset[2]



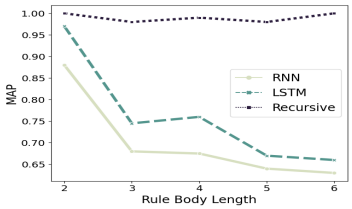




Figure 4.9: Longer-length rule detection comparison on Family dataset. Left: overall rule detection performance, where the lengths of rule bodies are varied among $\{2, 3, 4, 5, 6\}$. Right: rule detection performance w.r.t body length.

system to predict rule heads of *longer rule bodies* whose lengths range from 2 to 6.

**Comparing with Other Methods.** Considering that most of the existing logical rule learning methods rely on rule instances to define the score function for rule evaluation and thus cannot handle such a difficult setting, we construct two baselines by replacing the relation path encoder in RLogic with RNN and LSTM. Experiment results are shown in the left figures of Fig. 4.9. We observe that: (1) RLogic performs the best in rule head prediction tasks, giving almost completely correct predictions; (2) LSTM performs better than RNN, as it naturally can handle longer sequential data better compared with RNN.

**Effect of Rule Length.** To further investigate how the length of the rule body affects the performance, we vary the lengths of rule bodies among $\{2, 3, 4, 5, 6\}$ and report corresponding MAP. As depicted in the right figure of Fig. 4.6, the performances of RNN and LSTM drop severely when the rule bodies grow longer while RLogic is less affected by the length of rule body.

**Case Study.** In addition, we conduct a case study to show the power of the recursive mechanism which enables RLogic to give the right predictions when the rule body grows longer. As shown in the figure on the top of Fig. 4.10, three queries are manually designed with rule bodies of length 2, 5, and 6 respectively. The longer queries are formed by adding new predicates to the end of the rule body of the shorter ones. The top three rule heads inferred are given in the table at the bottom

[2]The code provided by RNNLogic didn't output the learned rules. Without the learned rules, we are unable to include RNNLogic in this experiment.

**Query 1** $?(x,y) \leftarrow \boldsymbol{hasBrother}(x,z_1) \wedge \boldsymbol{hasSister}(z_1,y)$

**Query 2** $?(x,y) \leftarrow \boldsymbol{hasBrother}(x,z_1) \wedge \boldsymbol{hasSister}(z_1,z_2) \wedge \boldsymbol{hasFather}(z_2,z_3) \wedge \boldsymbol{hasWife}(z_3,z_4) \wedge \boldsymbol{hasHusband}(z_4,y)$

**Query 3** $?(x,y) \leftarrow \boldsymbol{hasBrother}(x,z_1) \wedge \boldsymbol{hasSister}(z_1,z_2) \wedge \boldsymbol{hasFather}(z_2,z_3) \wedge \boldsymbol{hasWife}(z_3,z_4) \wedge \boldsymbol{hasHusband}(z_4,z_5) \wedge \boldsymbol{hasBrother}(z_5,y)$

| Query | RNN | LSTM | RLogic |
|---|---|---|---|
| Query 1 | **hasSister 0.7** | **hasSister 0.9** | **hasSister 0.9** |
| | hasBrother 0.1 | hasBrother 0.0 | hasDaughter 0.0 |
| | hasMother 0.1 | hasNiece 0.0 | hasBrother 0.0 |
| Query 2 | hasSister 0.8 | **hasFather 0.8** | **hasFather 1.0** |
| | **hasFather 0.1** | hasMother 0.1 | hasSon 0.0 |
| | hasUncle 0.0 | hasDaughter 0.1 | hasMother 0.0 |
| Query 3 | hasBrother 0.3 | hasMother 0.8 | **hasUncle 0.8** |
| | hasDaughter 0.3 | hasBrother 0.1 | hasBrother 0.1 |
| | hasSon 0.2 | hasFather 0.1 | hasAunt 0.0 |

Figure 4.10: Case study of RLogic in inferring rules with different lengths. The figure on the top gives rule bodies with different lengths (i.e., 2, 5, 6). The table at the bottom presents predicted rule heads ranked by probabilities. The bold predicates correspond to the ground truth answers.

of Fig. 4.10, associated with their probabilities (rounded). We observed that RLogic consistently performs well in all cases while it is more and more challenging for RNN and LSTM to provide correct prediction with the increase of body length.

#### 4.4.3.4 Training Efficiency

RLogic learns rules directly at the schema level while other existing methods learn rules based on instance-level ground rules. Therefore, RLogic is much more efficient than all existing methods. To demonstrate the scalability of RLogic, we give the training time of RLogic and other logical rule learning methods on three benchmark datasets in Table 4.6. Considering that it is challenging for baseline methods to learn long rules, to fairly compare with different methods, we limit the maximum length of learned rules to 2. We can observe that: (1) Neural-LP, NLIL, and DRUM do not perform well in terms of efficiency as they involve large matrix multiplication. They cannot handle YAGO3-10 dataset due to the memory issue; (2) It is also challenging for RNNLogic to scale to large-scale KGs. It can neither handle KG with hundreds of relations (e.g., FB15K-237) nor KG with million entities (e.g., YAGO3-10); (3) Although performances of RLogic and AMIE are on the same scale, AMIE is less efficient as it relies on all rule instances for rule evaluation.

Table 4.6: Training time (minutes) of logical rule learning methods on WN18-RR, FB15K-237, and YAGO3-10 datasets.

| Methods | WN18-RR | FB15K-237 | YAGO3-10 |
|---|---|---|---|
| Neural-LP† | 21.8 | 395.0 | - |
| NLIL† | 14.9 | 108.3 | - |
| DRUM† | 19.1 | 373.8 | - |
| AMIE | 0.5 | 13.9 | 41.3 |
| RNNLogic† | 17.4 | - | ¿ 4 days |
| RLogic | **0.2** | **5.2** | **17.3** |

† Neural-LP, NLIL, and DRUM exceed the capacity of our machines on YAGO3-10 dataset and RNNLogic exceeds the capacity of our machines on FB15K-237 dataset.

#### 4.4.3.5 Quality and Interpretability of the Rules

To demonstrate the quality and interpretability of rules mined by RLogic, we show some logic rules generated on the FB15k-237 dataset in Table 4.7. Two rules with different lengths are presented for each head predicate. We highlight the predicates which convey the same semantic meaning with boldface. We can observe that the boldfaced predicates in longer rules can be used to infer the boldfaced predicate in shorter rules. This observation again validates that RLogic is able to capture the deductive nature of logical rules.

## 4.5 NCRL: An Extension of RLogic

Logical rules naturally have an interesting property - called *compositionality*: where the meaning of a whole logical expression is a function of the meanings of its parts and of the way they are combined hupkes2020compositionality. To concretely explain compositionality, let us consider the family relationships shown in Fig. 4.11. In Fig. 4.11(a), we show that the rule ($hasUncle \leftarrow hasMother \wedge hasMother \wedge hasSon$) forms a composition of smaller logical expressions, which can

Table 4.7: Top rules learned by RLogic on FB15K-237. We highlight the predicates which convey the same semantic meaning with boldface.

| |
|---|
| speak_language$(x, y) \leftarrow$ geographic_distribution$(x, z) \wedge$ **phone_service_language**$(z, y)$ |
| speak_language$(x, y) \leftarrow$ geographic_distribution$(x, z_1) \wedge$ **tv_network_programs**$(z_1, z_2) \wedge$ **program_language**$(z_2, y)$ |
| location_at_time_zones$(x, y) \leftarrow$ county_at_location$(x, z) \wedge$ **location_at_time_zones**$(z, y)$ |
| location_at_time_zones$(x, y) \leftarrow$ county_at_location$(x, z_1) \wedge$ **location_partially_contains**$(z_1, z_2) \wedge$ **location_at_time_zones**$(z_2, y)$ |
| has_nationality$(x, y) \leftarrow$ write_tv_programs$(x, z) \wedge$ **tv_programs_in_country**$(z, y)$ |
| has_nationality$(x, y) \leftarrow$ write_tv_programs$(x, z_1) \wedge$ **has_regular_tv_appearance**$(z_1, z_2) \wedge$ **headquarters_in_country**$(z_2, y)$ |

be expressed as a hierarchy, where predicates (i.e., relations) can be combined and replaced by another single predicate. For example, predicates *hasMother* and *hasMother* can be combined and replaced by predicate *hasGrandma* as shown in Fig. 4.11(a). As such, by recursively combining predicates into a composition and reducing the composition into a single predicate, we can finally infer the rule head (i.e., *hasUncle*) from the rule body. While there are various possible hierarchical trees to represent such rules, not all of them are valid given the observed relations in the KG. For example, in Fig. 4.11(b), given a KG which only contains relations $\{$*hasMother*, *hasSon*, *hasGrandma*, *hasUncle*$\}$, it is possible to combine *hasMother* and *hasSon* first. However, there is no proper predicate to represent it in the KG. Therefore, learning a high-quality compositional structure for a given logical expression is critical for rule discovery.

Although RLogic attempts to learn the compositional structure, it adopts a heuristic method to find the order to deduct a relation path. Due to the limited modeling capability of the heuristic method, it is likely to lead to errors or biases because it is based on incomplete or inaccurate information. NCRL extends RLogic to *explicitly learns a hierarchical tree* to express the rule composition in an *end-to-end* way.

### 4.5.1 Framework of NCRL

An overview of NCRL is shown in Fig. 4.12. NCRL starts by sampling a set of paths from a given KG, and further splitting each path into short compositions using a sliding window. Then, NCRL uses a reasoning agent to reason over all the compositions to select one composition. NCRL uses

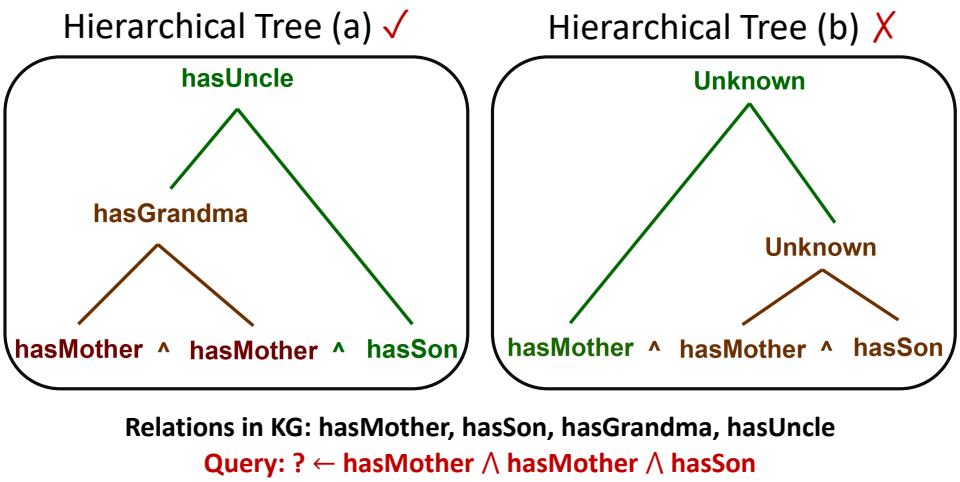



Figure 4.11: Learning an accurate hierarchical structure is significant for rule discovery: (a) a good compositional structure; (b) an improper compositional structure.

a recurrent attention unit to transform the selected composition into a single relation represented as a weighted combination of existing relations. By recurrently merging compositions in the path, NCRL finally predicts the rule head. Algorithm 4 outlines the learning procedure of NCRL.

#### 4.5.1.1 Logic Rule Learning With Recurrent Attention Unit

While the rule body can be viewed as a sequence, it naturally exhibits a rich hierarchical structure. The semantics of the rule body is highly dependent on its hierarchical structure, which cannot be exploited by most of the existing rule learning methods. To explicitly allow our model to capture the hierarchical nature of the rule body, we need to learn how the relations in the rule body are combined as well as the principle to reduce each composition in the hierarchical tree into a single predicate.

**Hierarchical Structure Learning** The hierarchical structure of logical rules is learned in an iterative way. At each step, NCRL selects only one composition from the rule body and replaces the selected composition with another single predicate based on the recurrent attention unit to reduce the rule body. Although rule body is hierarchical, when operations are very local (i,e., leaf-level composition), a composition is strictly sequential. To identify a composition from a

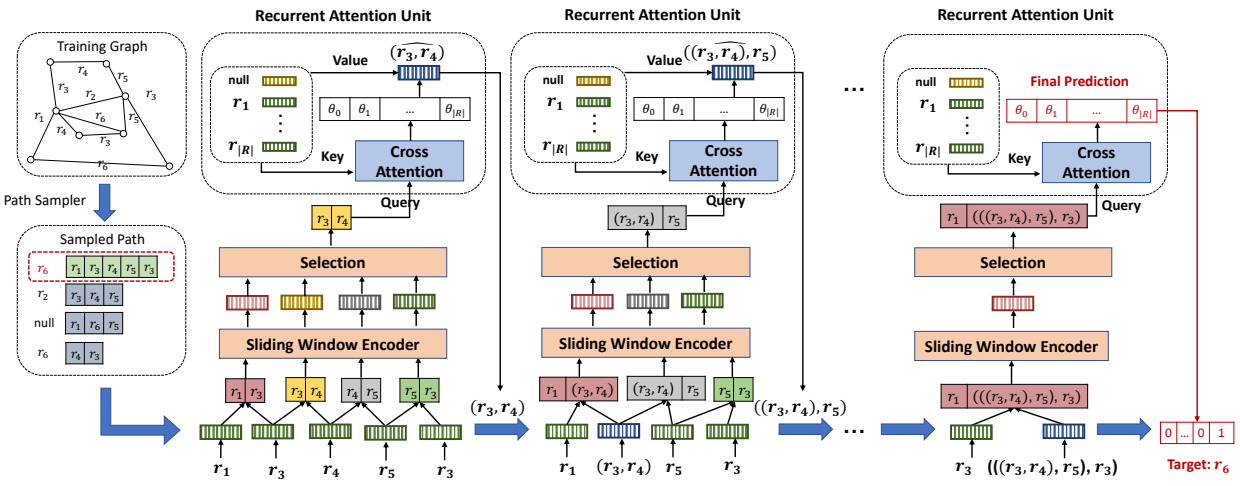



Figure 4.12: An overview of NCRL. It samples paths from KG (e.g.,$[r_1, r_3, r_4, r_5, r_3]$), and predicts the relations that directly connect the sampled paths (e.g.,$r_6$) based on the learned rules. NCRL takes the embeddings of predicates in the sampled paths as the input and outputs $\boldsymbol{\theta}$ as the probability of each relation to be the rule head.

sampled path, we use a sliding window with different lengths to decompose the sampled paths into compositions of different sizes. In our implementation, we vary the size of the sliding window among $\{2, 3\}$. Given a fixed window size $s$, sliding windows are generated by a size $s$ window which slides through the rule body $\mathbf{r_b} = [r_{b_1}, \ldots, r_{b_n}]$.

**Sliding Window Encoder.** When operations are over a local sliding window (i.e., composition), the relations within a sliding window should strictly follow a chain structure. Sequence models can be utilized to encode a sliding window. Considering the tradeoff between model complexity and performance, we chose RNN schuster1997bidirectional over other sequence models to encode the sequence. For example, taking $i$-th sliding window whose size is 2 (i.e., $w_i = [r_{b_i}, r_{b_{i+1}}]$) as the input, RNN outputs:

$$[\mathbf{h}_i, \mathbf{h}_{i+1}] = \text{RNN}(w_i) \tag{4.40}$$

where $\mathbf{h}_i \in \mathbb{R}^d$ is a hidden-state of predicate $r_{b_i}$ in $w_i$. Since the final hidden-state $\mathbf{h}_{i+1}$ is usually used to represent the whole sequence, we represent the sliding window as $\mathbf{w}_i = \mathbf{h}_{i+1}$.

**Composition Selection.** $\mathbf{w}_i$ is useful to estimate how likely the relations in $i$-th window appear together. If these relations always appear together, they have a higher probability to form

a meaningful composition. To incorporate this observation into our model, we select the sliding window by computing:

$$\boldsymbol{\mu} = \text{softmax}([f(\mathbf{w}_1), f(\mathbf{w}_2), \ldots, f(\mathbf{w}_{n+1-s})]) \tag{4.41}$$

where $f$ is a fully connected neural network. It learns the probability of $i$-th window to be a meaningful composition from its representation $\mathbf{w}_i$. $w_i$ with the highest $\boldsymbol{\mu}_i$ will be selected as the input to the recurrent attention unit.

**Recurrent Attention Unit** Note that rule induction following its underlying hierarchical structure is a recurrent process. Therefore, we propose a novel recurrent attention unit to recurrently reduce the selected composition into a single predicate until it outputs a final relation.

**Attention-based Induction.** The goal of a recurrent attention unit is to reduce the selected composition into a single predicate, which can be modeled as matching the composition with another single predicate based on its semantic consistency. Since attention mechanisms yield impressive results in Transformer models by capturing the semantic correlation between every pair of tokens in natural language sentence vaswani2017attention, we propose to utilize attention to reduce the selected composition $\mathbf{w}_i$. Note that we may not always find an existing relation to replace the selected composition. For example, given the composition [hasBrother, hasWife], none of the existing relations can be used to represent it. As such, in order to accommodate unseen relations, we incorporate a "null" predicate into potential rule heads and denote it as $r_0$. The embedding corresponding to $\mathbf{r_0}$ is set as the representation of the selected composition $\mathbf{w}_i$. In this way, when there is no direct link closing a sampled path (which means we do not have the ground truth about the rule head), we use the representation of the selected composition to represent itself rather than replace it with an existing relation. Let $H \in \mathbb{R}^{(|R|+1)\times d}$ be the matrix of the concatenation of all head relations, where $H_0 = \mathbf{w}_i \in \mathbb{R}^d$ is set as the selected composition. By taking $\mathbf{w}_i$ as a query and $H$ as the content, the scaled dot-product attention $\boldsymbol{\theta}$ [3] can be computed to estimate the semantic

[3] $\boldsymbol{\theta}$ is specific to the query composition $\mathbf{w}_i$.

consistency between the selected composition and its potential heads:

$$\boldsymbol{\theta} = \text{softmax}(\frac{\mathbf{w}_i W_Q (HW_K)^T}{\sqrt{d}}) \tag{4.42}$$

where $W_Q, W_K \in \mathbb{R}^{d\times d}$ are learnable parameters that project the inputs into the space of query and key. $\boldsymbol{\theta} \in \mathbb{R}^{|R|+1}$ is the learned attention, in which $\boldsymbol{\theta}_j$ measures $p(r_j|w_i)$ - the probability that the selected composition can be replaced by the predicate $r_j$ based on their semantic consistency. Given $\boldsymbol{\theta}$, we are able to compute a new representation for the selected composition as a weighted combination of all head relations (i.e., values) each weighted by its attention weight:

$$\widehat{\mathbf{w}}_i = \boldsymbol{\theta} H W_V \tag{4.43}$$

where $\widehat{\mathbf{w}}_i \in \mathbb{R}^d$ is the new representation of the selected composition. We project the key and value to the same space by requiring $W_V = W_K$ because the keys and the values are both embeddings of relations in KG. As shown in Fig. 4.12, we can reduce the long rule body $[r_{b_1}, r_{b_2} \ldots, r_{b_n}]$ by recursively applying the attention unit to replace its composition $(r_{b_i}, r_{b_{i+1}})$ with a single predicate. In the final step of the prediction, the attention $\boldsymbol{\theta}$ computed following Eq. 4.42 collects the probability that the rule body can be replaced by each of the head relations.

#### 4.5.1.2 Training and Rule Extraction

NCRL is trained in an end-to-end fashion. It starts by sampling paths from an input KG and predicts the relation which directly closes the sampled paths based on learned rules.

**Path Sampling.** We utilize a random walk spitzer2013principles sampler to sample paths that connect two entities from the KG. Formally, given a source entity $x_0$, we simulate a random walk of max length $n$. Let $x_i$ denote the $i$-th node in the walk, which is generated by the following distribution:

$$p(x_i = e_i | x_{i-1} = e_j) = \begin{cases} \frac{1}{|\mathcal{N}(e_j)|}, & \text{if } (e_i, e_j) \in E \\ 0, & \text{otherwise} \end{cases} \tag{4.44}$$

where $|\mathcal{N}(e_j)|$ is the neighborhood size of entity $e_j$. Different from a random walk, each time after we sample the next entity $x_i$, we add all the edges which can directly connect $x_0$ and $x_i$ in KG. We denote the path connecting two nodes $x_0$ and $x_n$ as $p$, where $p = [r_{b_1}, \ldots, r_{b_n}]$, indicating

$x_0 \xrightarrow{r_1} \ldots \xrightarrow{r_n} x_n$. We also denote the relation that directly connects $x_0$ and $x_n$ as $r_h$. If none of the relations directly connects the $x_0$ and $x_n$, we set $r_h$ as "null". We control the ratio of non-closed paths during path sampling to ensure a majority of sampled paths are associated with an observed head relation.

**Algorithm 4:** Learning Algorithm

**Input:** Observed triples in KG $O$

**Output:** Relation embeddings

$\mathcal{P}$ = SamplePaths($O$)
**for** $(p, r_h) \in \mathcal{P}$ **do**
  **while** $len(p) > s$ **do**
    // Decompose $p$ with a sliding window, whose size is $s$
    $[w_1, \ldots, w_{n+1-s}]$ = Decompose($p$)
    // Select a composition
    $[\mathbf{w}_1, \ldots, \mathbf{w}_{n+1-s}]$ = RNN($[w_1, \ldots, w_{n+1-s}]$)
    $\mathbf{w}_i$ = Select($[\mathbf{w}_1, \ldots, \mathbf{w}_{n+1-s}]$)
    // Apply recurrent attention unit
    $\widehat{\mathbf{w}_i}$ = Attn($\mathbf{w}_i$)
    // Reduce the sampled path $p$
    $p = [\mathbf{r_{b_1}}, \ldots, \mathbf{w}_i, \ldots, \mathbf{r_{b_n}}]$
  **end**
  // Final prediction
  $\mathbf{w}$ = RNN($p$)
  Take $\mathbf{w}$ as the query and compute $\boldsymbol{\theta}$ based on Eq. 4.42
  Minimize the loss in Eq. 4.45
**end**

**Objective Function.** Our goal is to maximize the likelihood of the observed relation $r_h$, which directly closes the sampled path $p$. The attention $\boldsymbol{\theta}$ collects the predicted probability for $p$ being

closed by each of the head relations. We formulate the objective using the cross-entropy loss as:

$$-\sum_{(p,r_h)\in\mathcal{P}}\sum_{k=0}^{|R|}\mathbf{y}_k^{r_h}\log\boldsymbol{\theta}_k^p \tag{4.45}$$

where $\mathcal{P}$ denotes a set of sampled paths from a given KG, $\mathbf{y}^{r_h}\in\{0,1\}^{|R|+1}$ is the one-hot encoded vector such that only the $r_h$-th entry is 1, and $\boldsymbol{\theta}^p\in\mathbb{R}^{|R|+1}$ is the learned attention for the sampled path $p$. In particular, $\boldsymbol{\theta}_0^p$ represents the probability that the sampled path cannot be closed by any existing relations in KG.

**Rule Extraction.** Similar to RLogic, to recover logical rules, we calculate the score $s(r_h,\mathbf{r_b})$ for each rule $(r_h,\mathbf{r_b})$ in rule space based on the learned model. Given a candidate rule $(r_h,\mathbf{r_b})$, we reduce the rule body $\mathbf{r_b}$ into a single head $r_h$ by recursively merge compositions in path $\mathbf{r_b}$. At the final step of the prediction, we learn the attention $\boldsymbol{\theta}=[\boldsymbol{\theta}_0,\ldots,\boldsymbol{\theta}_{|R|}]$, where $\boldsymbol{\theta}_k$ is the score of rule $(r_k,\mathbf{r_b})$. The top $k$ rules with the highest score will be selected as learned rules.

### 4.5.2 Connection to Existing Approaches

The main objective of NCRL is to learn rules that *generalize to large-scale tasks and unseen graphs*. Let us consider the example in Fig. 4.13. From the training KG, we can extract two rules – rule ①: $\text{hasGrandma}(x,y)\leftarrow\text{hasMother}(x,z)\wedge\text{hasMother}(z,y)$ and rule ②: $\text{hasUncle}(x,y)\leftarrow\text{hasGrandma}(x,z)\wedge\text{hasSon}(z,y)$. We also observe that the necessary rule to infer the relation between *Alice* and *Bob* in the test KG is rule ③: $\text{hasUncle}(x,y)\leftarrow\text{hasMother}(x,z_1)\wedge\text{hasMother}(z_1,z_2)\wedge\text{hasSon}(z_2,y)$, which is not observed in the training KG. However, using *compositionality* to combine rules ① and ②, we can successfully learn rule ③ which is necessary for inferring the relation between *Alice* and *Bob* in the test KG. The successful prediction in the test KG shows the model's ability for systematic generalization, i.e., learning to reason on smaller graphs and making predictions on unseen graphs sinha2019clutrr.

Although compositionality is crucial for learning logical rules, most existing logical rule learning methods fail to exploit it. In traditional AI, inductive Logic Programming (ILP) muggleton1994inductive,muggleton1990efficient is the most representative symbolic method. Given a collection of positive examples and negative examples, an ILP system aims to learn logical rules

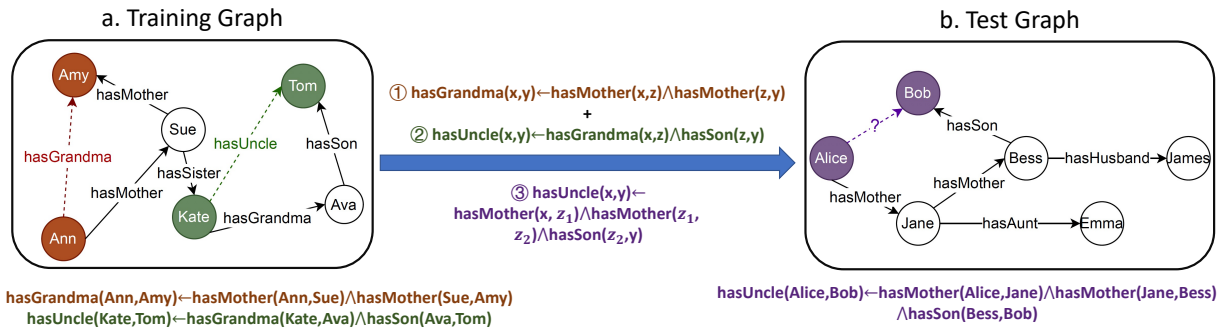



Figure 4.13: Illustration of how the compositionality of logical rules helps improve systematic generalization. (a) logical rule extraction from the observed graph (i.e., training stage) and (b) Inference on an unseen graph (i.e., test stage). The train and the test graphs have disjoint sets of entities. By combining logical rules ① and ② we can successfully learn rule ③ for prediction on unseen graphs.

which are able to entail all the positive examples while excluding any of the negative examples. However, it is difficult for ILP to scale beyond small rule sets due to their restricted computational complexity to handle the large search space of compositional rules. There are also some recent neural-symbolic methods, e.g., neural logic programming methods yang2017differentiable, sadeghian2019drum and principled probabilistic methods qu2020rnnlogic. Neural logic programming simultaneously learns logical rules and their weights in a differentiable way. Alternatively, principled probabilistic methods separate rule generation and rule weight learning by introducing a rule generator and a reasoning predictor. However, most of these approaches are particularly designed for the KG completion task. Moreover, since they require an enumeration of rules given a maximum rule length $T$, the complexity of these methods grows exponentially as max rule length increases, which severely limits their systematic generalization capability.

To overcome these issues, several works such as conditional theorem provers (CTPs) minervini2020learning and recurrent relational reasoning (R5) lu2022r focused on the model's systematicity instead. CTPs is proposed to learn an optimal rule selection strategy via gradient-based optimization. For each sub-goal, a select module produces a smaller set of rules, which is then used during the proving mechanism. However, since the length of the learned rules influences

the number of parameters of the model, it limits the capability of CTPs to handle the complicated rules whose depth is large. In addition, due to its high computational complexity, CTPs cannot handle KG completion tasks for large-scale KGs. Another reinforcement learning-based method - R5 is proposed to provide a recurrent relational reasoning solution to learn compositional rules. However, R5 cannot generalize to the KG completion task due to the lack of scalability. It requires pre-sampling for the paths that entail the query. Considering that all the triples in a KG share the same training graph, even a relatively small-scale KG contains a huge number of paths. Thus, it is impractical to apply R5 to even small-scale KG for rule learning. In addition, R5 employs a hard decision mechanism for merging a relation pair into a single relation, which makes it challenging to handle the widely existing uncertainty in KGs. For example, given the rule body $hasAunt(x, z) \wedge hasSister(z, y)$, both $hasMother(x, y)$ and $hasAunt(x, y)$ can be derived as the rule head. The inaccurate merging of a relation pair may result in error propagation when generalizing to longer paths.

### 4.5.3 Experiments

Logical rules are valuable for various downstream tasks, such as (1) *KG completion task*, which aims to infer the missing entity given the query $(h, r, ?)$ or $(?, r, t)$; (2) A more challenging *inductive relational reasoning task*, which tests the systematic generalization capability of the model by inferring the missing relation between two entities (i.e., $(h, ?, t)$) with more hops than the training data. A majority of existing methods can handle only one of these two tasks (e.g., RNNLogic is designed for the KG completion task while R5 is designed for the inductive relational reasoning task). In this section, we show that our method is superior to existing SOTA algorithms on both tasks. In addition, we also empirically assess the interpretability of the learned rules.

**Knowledge Graph Completion** KG completion is a classic task widely used by logical rule learning methods such as Neural-LP yang2017differentiable, DRUM sadeghian2019drum and RNNLogic qu2020rnnlogic to evaluate the quality of learned rules. An existing algorithm called forward chaining salvat1996sound can be used to predict missing facts from logical rules.

**Datasets.** We use six widely used benchmark datasets to evaluate our NCRL in comparison to SOTA methods from knowledge graph embedding and rule learning methods. Specifically, we use the Family hinton1986learning, UMLS kok2007statistical, Kinship kok2007statistical, WN18RR dettmers2018convolutional, FB15K-237 toutanova2015observed, YAGO3-10 suchanek2007yago datasets. The statistics of the datasets are given in Table 4.8.

Table 4.8: Data statistics of widely used benchmark knowledge graphs.

| Dataset | # Data | # Relation | # Entity |
|---|---|---|---|
| Family | 28,356 | 12 | 3007 |
| UMLS | 5,960 | 46 | 135 |
| Kinship | 9,587 | 25 | 104 |
| FB15K-237 | 310,116 | 237 | 14,541 |
| WN18RR | 93,003 | 11 | 40,943 |
| YAGO3-10 | 1,089,040 | 37 | 123,182 |

**Evaluation Metrics.** We mask the head or tail entity of each test triple and require each method to predict the masked entity. During the evaluation, we use the filtered setting bordes2013translating and three evaluation metrics, i.e., Hit@1, Hit@10, and MRR.

**Comparing with Other Methods.** We evaluate our method against SOTA methods, including (1) traditional KG embedding (KGE) methods (e.g., TransE bordes2013translating, DistMult yang2014embedding, ConvE dettmers2018convolutional, ComplEx trouillon2016complex and RotatE sun2019rotate); (2) logical rule learning methods (e.g., Neural-LP yang2017differentiable, DRUM sadeghian2019drum, RNNLogic qu2020rnnlogic and RLogic cheng2022rlogic). The systematic generalizable methods (e.g., CTPs and R5) cannot handle KG completion tasks due to their high complexity.

**Results.** The comparison results are presented in Table 4.9. We observe that: (1) Although NCRL is not designed for KG completion task, compared with KGE models, it achieves comparable results on all datasets, especially on Family, UMLS, and WN18RR datasets; (2) NCRL consistently outperforms all other rule learning methods with significant performance gain in most cases.

Table 4.9: KG completion. The red numbers represent the best performances among all methods, while the blue numbers represent the best performances among all *rule learning* methods.

| Methods | Models | Family | | | Kinship | | | UMLS | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | MRR | Hit@1 | Hit@10 | MRR | Hit@1 | Hit@10 | MRR | Hit@1 | Hit@10 |
| KGE | TransE | 0.45 | 22.1 | 87.4 | 0.31 | 0.9 | 84.1 | 0.69 | 52.3 | 89.7 |
| | DistMult | 0.54 | 36.0 | 88.5 | 0.35 | 18.9 | 75.5 | 0.391 | 25.6 | 66.9 |
| | ComplEx | 0.81 | 72.7 | 94.6 | 0.42 | 24.2 | 81.2 | 0.41 | 27.3 | 70.0 |
| | RotatE | 0.86 | 78.7 | 93.3 | **0.65** | **50.4** | **93.2** | 0.74 | 63.6 | 93.9 |
| Rule Learning | Neural-LP | 0.88 | 80.1 | 98.5 | 0.30 | 16.7 | 59.6 | 0.48 | 33.2 | 77.5 |
| | DRUM | 0.89 | 82.6 | 99.2 | 0.33 | 18.2 | 67.5 | 0.55 | 35.8 | 85.4 |
| | RNNLogic | 0.86 | 79.2 | 95.7 | **0.64** | **49.5** | 92.4 | 0.75 | 63.0 | 92.4 |
| | RLogic | 0.88 | 81.3 | 97.2 | 0.58 | 43.4 | 87.2 | 0.71 | 56.6 | 93.2 |
| | NCRL | **0.91** | **85.2** | **99.3** | **0.64** | 49.0 | **92.9** | **0.78** | **65.9** | **95.1** |

| Methods | Models | WN18RR | | | FB15K-237 | | | YAGO3-10 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | MRR | Hit@1 | Hit@10 | MRR | Hit@1 | Hit@10 | MRR | Hit@1 | Hit@10 |
| KGE | TransE | 0.23 | 2.2 | 52.4 | 0.29 | 18.9 | 46.5 | 0.36 | 25.1 | 58.0 |
| | DistMult | 0.42 | 38.2 | 50.7 | 0.22 | 13.6 | 38.8 | 0.34 | 24.3 | 53.3 |
| | ConvE | 0.43 | 40.1 | 52.5 | **0.32** | 21.6 | 50.1 | 0.44 | 35.5 | 61.6 |
| | ComplEx | 0.44 | 41.0 | 51.2 | 0.24 | 15.8 | 42.8 | 0.34 | 24.8 | 54.9 |
| | RotatE | 0.47 | 42.9 | 55.7 | **0.32** | **22.8** | **52.1** | **0.49** | **40.2** | **67.0** |
| Rule Learning | Neural-LP | 0.38 | 36.8 | 40.8 | 0.24 | 17.3 | 36.2 | - | - | - |
| | DRUM | 0.38 | 36.9 | 41.0 | 0.23 | 17.4 | 36.4 | - | - | - |
| | RNNLogic | 0.46 | 41.4 | 53.1 | 0.29 | 20.8 | 44.5 | - | - | - |
| | RLogic | 0.47 | 44.3 | 53.7 | **0.31** | 20.3 | **50.1** | 0.36 | 25.2 | 50.4 |
| | NCRL | **0.67** | **56.3** | **85.0** | 0.30 | **20.9** | 47.3 | **0.38** | **27.4** | **53.6** |

† Neural-LP, DRUM, and RNNLogic exceed the memory capacity of our machines on YAGO3-10 dataset

**Ablation Study** **Performance w.r.t. Data Sparsity.** We construct sparse KG by randomly removing $a$ triples from the original dataset. Following this approach, we vary the sparsity ratio $a$ among $\{0.33, 0.66, 1\}$ and report performance on different methods over the KG completion task on Kinship dataset. As presented in Fig. 4.14, the performance of NCRL does not vary a lot with different sparsity ratios $a$, which is appealing in practice.

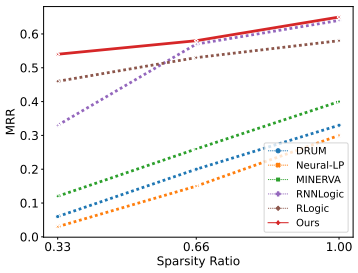



Figure 4.14: Performance of KG completion vs sparsity ratio on Kinship.

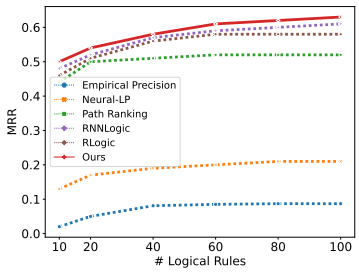



Figure 4.15: Performance of KG completion vs # logical rules on Kinship.

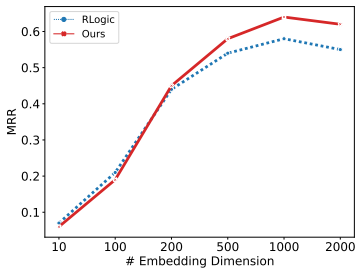



Figure 4.16: Performance of KG completion vs embedding dimension on Kinship.

**KG completion performance w.r.t. the Number of Learned Rules.** We generate $k$ rules with the highest qualities per query relation and use them to predict missing links. We vary $k$ among $\{10, 20, 40, 60, 80, 100\}$. The results on Kinship are given in Fig. 4.15. We observed that even with only 10 rules per relation, NCRL still gives competitive results.

**Performance w.r.t. Embedding Dimension.** We vary the dimension of relation embeddings among $\{10, 100, 200, 500, 1000, 2000\}$ and present the results on Kinship in Fig. 4.16, comparing against RLogic cheng2022rlogic. We see that the embedding dimension has a significant impact on KG completion performance. The best performance is achieved at $d = 1000$.

**Training Efficiency** To demonstrate the scalability of NCRL, we give the training time of NCRL against other logical rule learning methods on three benchmark datasets in Table 4.10. We observe that: (1) Neural-LP and DRUM do not perform well in terms of efficiency as they apply a sequence of large matrix multiplications for logic reasoning. They cannot handle YAGO3-10 dataset due to the memory issue; (2) It is also challenging for RNNLogic to scale to large-scale KGs as it relies

Table 4.10: Training time (s) of rule learning methods

| | NeuralLP | DRUM | RNNLogic | NCRL |
|---|---|---|---|---|
| WN18RR | 1,308 | 1,146 | 1,044 | **77** |
| FB15k-237 | 23,708 | 22,428 | - | **410** |
| YAGO3-10 | - | - | - | **190** |

on all ground rules to evaluate the generated rules in each iteration. It is difficult for it to handle KG with hundreds of relations (e.g., FB15K-237) nor KG with million entities (e.g., YAGO3-10); (3) our NCRL is on average 100x faster than state-of-the-art baseline methods.

**Systematic Generalization** We test NCRL for systematic generalization to demonstrate the ability of NCRL to perform reasoning over graphs with more hops than the training data, where the model is trained on smaller graphs and tested on larger unseen ones. The goal of this experiment is to infer the relation between node pair queries. We use two benchmark datasets:

- **Clean Data** CLUTRR [110] is a dataset for inductive reasoning over family relations. The goal is to infer the missing relation between two family members. The train set contains graphs with up to 4 hops paths, and the test set contains graphs with up to 10 hops paths. The train and test splits share disjoint set of entities. The detailed statistics of CLUTRR is provided in Table 4.11.

- **Noisy Data** Graphlog [111] is a benchmark dataset designed to evaluate systematicity. It contains logical worlds where each world contains graphs that are created using a different set of ground rules. The goal is to infer the relation between a queried node pair. GraphLog contains more bad examples than CLUTRR does. The detailed statistics of Graphlog are provided in Table 4.12. When the ARL is larger, the dataset becomes noisier and contains more bad cases.

Note that most existing rule learning methods lack systematic generalization. CTPs minervini2020learning, R5 lu2022r, and RLogic cheng2022rlogic are the only comparable rule learning methods for this

Table 4.11: Data statistics of CLUTRR datasets.

| Dataset | # Relation | # Train | # Test |
|---|---|---|---|
| CLUTRR | 22 | 15,083 | 823 |

Table 4.12: Data statistics of GraphLog datasets. NC: number of classes. ND: number of distinct resolution edge sequences (distinct descriptors). ARL: average resolution length. AN: average number of nodes. AE: average number of edges.

| Dataset | NC | ND | ARL | AN | AE | #Train | #Test |
|---|---|---|---|---|---|---|---|
| World 2 | 17 | 157 | 3.21 | 9.8 | 11.2 | 5000 | 1000 |
| World 3 | 16 | 189 | 3.63 | 11.1 | 13.3 | 5000 | 1000 |
| World 6 | 16 | 249 | 5.06 | 16.3 | 20.2 | 5000 | 1000 |
| World 8 | 15 | 404 | 5.43 | 16.0 | 19.1 | 5000 | 1000 |

task.

**Systematic Generalization on CLUTRR.** Table 4.13 shows the results of NCRL against SOTA algorithms. We observe that the performances of sequential models and embedding-based models drop severely when the path length grows longer while NCRL still predicts successfully on longer paths without significant performance degradation. Compared with systematic generalizable rule learning methods, NCRL has better generalization capability than CTPs especially when the paths grow longer. Even though R5 gives invincible results over CLUTRR dataset, NCRL shows comparable performance.

**Systematic Generalization on GraphLog.** Table 4.14 shows the results on 4 selected worlds. We observed that NCRL consistently outperforms other rule-learning baselines over all 4 worlds.

**Case Study of Generated logical rules** Finally, we show a case study of logical rules that are generated by NCRL on the YAGO3-10 dataset in Table 4.15. We can see that these logical rules are meaningful and diverse. Two rules with different lengths are presented for each head predicate.

Table 4.13: Results of inductive relational reasoning on CLUTRR dataset. Trained on path samples with hops $\{2, 3, 4\}$ and evaluated on path samples with hops $\{5, \ldots, 10\}$. The red numbers represent the best performances while the brown numbers represent the second best performances.

| Model \ # Hops | **5 Hops** | **6 Hops** | **7 Hops** | **8 Hops** | **9 Hops** | **10 Hops** |
|---|---|---|---|---|---|---|
| RNN | 0.93±0.06 | 0.87±0.07 | 0.79±0.11 | 0.73±0.12 | 0.65±0.16 | 0.64±0.16 |
| LSTM | 0.98±0.03 | 0.95±0.04 | 0.89±0.10 | 0.84±0.07 | 0.77±0.11 | 0.78±0.11 |
| GRU | 0.95±0.04 | 0.94±0.03 | 0.87±0.8 | 0.81±0.13 | 0.74±0.15 | 0.75±0.15 |
| Transformer | 0.88±0.03 | 0.83±0.05 | 0.76±0.04 | 0.72±0.04 | 0.74±0.05 | 0.70±0.03 |
| GNTP | 0.68±0.28 | 0.63±0.34 | 0.62±0.31 | 0.59±0.32 | 0.57±0.34 | 0.52±0.32 |
| GAT | 0.99±0.00 | 0.85±0.04 | 0.80±0.03 | 0.71±0.03 | 0.70±0.03 | 0.68±0.02 |
| GCN | 0.94±0.03 | 0.79±0.02 | 0.61±0.03 | 0.53±0.04 | 0.53±0.04 | 0.41±0.04 |
| $\text{CTP}_L$ | **0.99±0.02** | 0.98±0.04 | 0.97±0.04 | **0.98±0.03** | 0.97±0.04 | 0.95±0.04 |
| $\text{CTP}_A$ | 0.99±0.04 | **0.99±0.03** | 0.97±0.03 | 0.95±0.06 | 0.93±0.07 | 0.91±0.05 |
| $\text{CTP}_M$ | 0.98±0.04 | 0.97±0.06 | 0.95±0.06 | 0.94±0.08 | 0.93±0.08 | 0.90±0.09 |
| RLogic | **0.99±0.02** | 0.98±0.02 | 0.97±0.04 | 0.97±0.03 | 0.94±0.06 | 0.94±0.07 |
| R5 | **0.99±0.02** | 0.99±0.04 | **0.99±0.03** | **1.0±0.02** | **0.99±0.02** | **0.98±0.03** |
| NCRL | **1.0±0.01** | **0.99±0.01** | **0.98±0.02** | **0.98±0.03** | **0.98±0.03** | **0.97±0.02** |

Table 4.14: Results of inductive relational reasoning on GraphLog datasets for robustness analysis.

| | CTP | | RLogic | | R5 | | NCRL | |
|---|---|---|---|---|---|---|---|---|
| | ACC | Recall | ACC | Recall | ACC | Recall | ACC | Recall |
| **World 2** | 0.685±0.03 | 0.80±0.05 | 0.726±0.02 | 0.95±0.00 | 0.755 ±0.02 | 1.0±0.00 | **0.774±0.01** | 1.0±0.00 |
| **World 3** | 0.624±0.02 | 0.85±0.00 | 0.737±0.02 | 1.0±0.00 | 0.791±0.03 | 1.0±0.00 | **0.797±0.02** | 1.0±0.00 |
| **World 6** | 0.533 ±0.03 | 0.85±0.00 | 0.638 ±0.03 | 0.90±0.00 | 0.687±0.05 | 0.9±0.00 | **0.702±0.02** | 0.95±0.00 |
| **World 8** | 0.545±0.02 | 0.70±0.00 | 0.605±0.02 | 0.90±0.00 | 0.671±0.03 | 0.95±0.00 | **0.687±0.02** | 0.95±0.00 |

Table 4.15: Top rules learned on YAGO3-10. We highlight the composition and predicate which share the same semantic meaning with boldface.

| |
|---|
| $\text{isLocatedIn}(x, y) \leftarrow \textbf{isLocatedIn}(x, z) \wedge \text{isLocatedIn}(z, y)$ |
| $\text{isLocatedIn}(x, y) \leftarrow \textbf{hasAcademicAdvisor}(x, z_1) \wedge \textbf{isLocatedIn}(z_1, z_2) \wedge \text{isLocatedIn}(z_2, y)$ |
| $\text{isAffiliatedTo}(x, y) \leftarrow \text{isKnownFor}(x, z) \wedge \textbf{isAffiliatedTo}(z, y)$ |
| $\text{isAffiliatedTo}(x, y) \leftarrow \text{isKnownFor}(x, z_1) \wedge \textbf{isAffiliatedTo}(z_1, z_2) \wedge \textbf{isLeaderOf}(z_2, y)$ |
| $\text{playsFor}(x, y) \leftarrow \text{isKnownFor}(x, z) \wedge \textbf{isAffiliatedTo}(z, y)$ |
| $\text{playsFor}(x, y) \leftarrow \text{isKnownFor}(x, z_1) \wedge \textbf{playsFor}(z_1, z_2) \wedge \textbf{owns}(z_2, y)$ |
| $\text{influences}(x, y) \leftarrow \text{isPoliticianOf}(x, z) \wedge \textbf{influences}(z, y)$ |
| $\text{influences}(x, y) \leftarrow \text{isPoliticianOf}(x, z_1) \wedge \textbf{influences}(z_1, z_2) \wedge \textbf{influences}(z_2, y)$ |

We highlight the composition and predicate which share the same semantic meaning with boldface.

## 4.6 Summary and Discussions

The goal of logical rule learning is to discover a set of logical rules that can be used to represent the underlying structure of a given domain or to make accurate predictions about new, unseen data. By representing knowledge in a logical format, logical rule learning approaches allow human experts to understand and reason about the learned models and integrate them with existing KGs. Two broad categories of methods used for logical rule learning are traditional search-based methods and modern neural-symbolic integration.

Traditional search-based methods involve exploring the space of possible rules to identify the best ones based on specific criteria, such as accuracy. These methods offer various benefits, such as expressive knowledge representation and generalization capability. However, they also face challenges such as computational complexity, sensitivity to noise, and difficulties in integrating background knowledge or managing complex relationships.

Modern neural-symbolic integration offers an alternative approach to learning logical rules by

combining symbolic logical rule learning with neural network models. The incorporation of neural network models allows neural-symbolic integration methods to handle the challenges of learning from real-world KGs. Existing neural-symbolic integration approaches can be divided into two categories: differentiable searching-based methods and learning-based methods. Differentiable searching-based methods extend traditional searching-based methods by using a differentiable loss function to optimize the search in a continuous search space, while learning-based methods combine various learning paradigms to learn rules from data, where the rules are represented in a symbolic form. By combining the benefits of symbolic reasoning and neural networks, these methods provide increased adaptability in representing and learning logical rules from data, as well as harnessing the scalability of neural networks. However, they may require more data for learning precise rules and may be less interpretable compared to traditional search-based strategies.

Neural-symbolic integration is heavily reliant on observed data to learn rules, which makes it difficult for the method to identify rules that lack enough supported instances. To address this limitation, RLogic proposes a different approach. Rather than relying solely on rule instances for evaluation, RLogic suggests learning logical rules directly at the schema level using a representation learning-based model. The efficiency of this approach is greatly improved because only a small number of closed paths are required for model training. Additionally, RLogic recursively decomposes a large sequential model into smaller atomic models, enabling the detection of rules without the need for rule instances. This step is crucial in embedding deductive reasoning more deeply into the rule-learning process.

Due to space limitations, we can only cover the primary approaches to rule learning. However, it is essential to acknowledge that there are other promising directions in this field, which we will discuss below.

**Compositional Rule Learning** Logical rules naturally have an interesting property - called *compositionality*: where the meaning of a whole logical expression is a function of the meanings of its parts and of the way they are combined hupkes2020compositionality.

Although RLogic attempts to learn the compositional structure, it adopts a heuristic method to

find the order to deduct a relation path. Due to the limited modeling capability of the heuristic method, it is likely to lead to errors or biases because it is based on incomplete or inaccurate information. NCRL [21] extends RLogic to *explicitly learns a hierarchical tree* to express the rule composition in an *end-to-end* way. NCRL starts by sampling a set of paths from a given KG, and further splitting each path into short compositions using a sliding window. Then, NCRL uses a reasoning agent to reason over all the compositions to select one composition. NCRL uses a recurrent attention unit to transform the selected composition into a single relation represented as a weighted combination of existing relations. By recurrently merging compositions in the path, NCRL finally predicts the rule head.

# CHAPTER 5

# Combining Large Language Models and Knowledge Graphs

## 5.1 Overview

While LLMs can effectively handle straightforward reasoning problems, they often encounter challenges when faced with more complex reasoning, such as scenarios demanding multi-step reasoning [89]. Multi-step reasoning typically involves making inferences or answering questions that require multiple steps of logical reasoning. Here's an illustration of multi-step reasoning: "Marian went shoe shopping with her sister Michelle. Darnell's grandfather, Stanley, taught her how to make a paper airplane while her mother, Marian, prepared dinner. What is the family relationship between Michelle and Stanley?" Various methods, such as chain-of-thought (CoT) [129, 105] and Zero-Shot-CoT [63], have been proposed to improve multi-step reasoning in LLMs. These approaches involve step-by-step reasoning, either by providing examples with detailed intermediate steps leading to a conclusion or by prompting the model with "Let's think step by step" in a zero-shot setting. Despite their effectiveness, LLMs still face challenges in effectively addressing complex multi-step reasoning questions. The first challenge involves accurately comprehending relationships conveyed through natural language, as evident in the given example where Marian has a sister named Michelle, and Darnell has a grandfather named Stanley. Identifying these relationships accurately is crucial, but the inherent ambiguity in natural language makes this difficult. For instance, consider the sentence "Darnell's grandfather, Stanley, taught her how to make a paper airplane while her mother, Marian, prepared dinner," correctly inferring that Marian, not Stanley, is Darnell's mother requires understanding the gender implications and the contextual relational information. Second, LLMs must identify relevant information while ignoring the irrelevant. In the example "Marian went shoe shopping with her sister Michelle," recognizing that Michelle is Marian's sister is crucial,

while the detail about shoe shopping is not. This requires discernment in filtering out unnecessary details that could mislead. Third, accurate multi-step reasoning requires LLMs to logically connect information. In the given scenario, two steps of inference are required. Initially, recognizing that Marian is Stanley's daughter, followed by combining this with the fact that Michelle is Marian's sister, leads to the final deduction that Michelle is Stanley's daughter. This process, typically more straightforward in formal logic due to clear logical indicators, becomes more complex in natural language due to the lack of explicit logical connectors, posing challenges for LLMs in constructing accurate reasoning paths.

Considering all the previously mentioned challenges, performing multi-step reasoning directly based on unstructured text is a challenging task. To reduce the complexity, can LLMs be guided to adopt a more systematic and structured method for identifying reasoning paths for multi-step reasoning? Multi-step reasoning is essential to human intelligence, inspiring how we guide LLMs. Humans usually rely on structured knowledge representations, like Knowledge Graphs (KGs), to link different pieces of information clearly and systematically. Consider the question involving Marian, Michelle, Stanley, and Darnell. Answering it can be challenging due to the multiple individuals involved and the need to remember and correctly sequence their relationships. To address this, humans often create a graph to visually represent the relationships, as depicted in Fig. 5.1. They then deduce the relationships step by step, based on this graph. Although this method might seem simple, it is particularly effective, especially with longer inference chains.

Due to the inherent capability of Knowledge Graphs (KGs) to facilitate multihop reasoning, there have been various approaches to implement multihop reasoning using KGs. We explore existing techniques in two main categories: (1) *multi-step reasoning with LLMs*, and (2) *multi-step Reasoning with KGs*.

- **Multi-step Reasoning with LLMs**: This approach utilizes prompt engineering to improve multi-step reasoning capabilities within LLMs.
- **Multi-step Reasoning with KGs**: This strategy focuses on executing multihop reasoning processes within KGs.

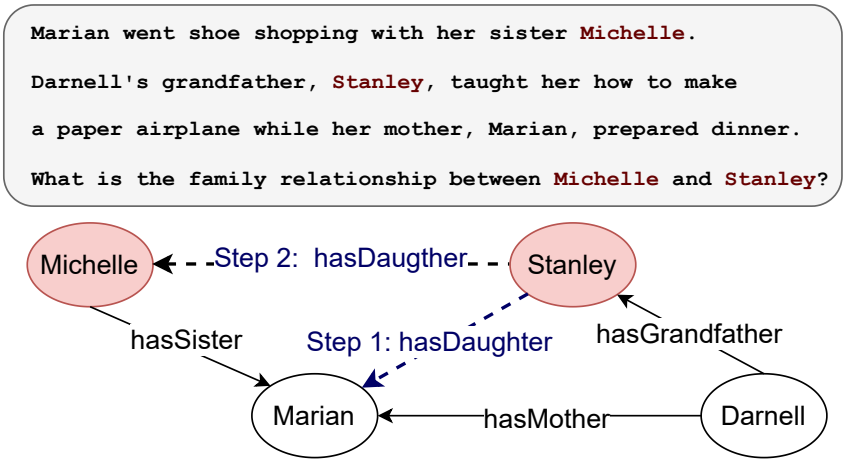



Figure 5.1: An example illustrating how humans manage multi-step questions. Our objective is to deduce the relationship between two individuals, Michelle and Stanley, highlighted in red, from a given story. Given that the story involves various individuals, humans typically first create a graph to clearly visualize the relationships among them. Then, they infer the relationship step by step, based on the graph.

Following this, we will introduce our innovative approach that integrates KGs to improve the reasoning abilities of LLMs.

## 5.2 Multi-step Reasoning with LLMs

### 5.2.1 In-context Learning

Multi-step reasoning is a challenging NLP task that requires a system to make multiple inference steps to answer a question. While LLMs exhibit strong capabilities in one-hop inference, they struggle to perform effectively in multi-step reasoning. Numerous in-context Learning strategies have been suggested to enhance the multi-step reasoning capabilities of LLMs, such as implementing step-by-step reasoning using few-shot examples. Unlike "naive" prompting, which expects that the input should be immediately followed by the output or answer, eliciting prompts direct LLMs to tackle tasks by guiding them through intermediate steps before making predictions for the final

output or answer. This method, known as chain-of-thought (CoT) [129, 105], has demonstrated that elicitive prompting equips LMs with superior reasoning abilities in a few-shot setting. Later, Zero-Shot-CoT [63] presented similar capabilities in a zero-shot setting. They simply prepended the input question with the phrase "Let's think step by step" before querying the model, and showed that large LMs performed well in zero-shot-CoT on reasoning tasks like GSM8K, though not as proficiently as in few-shot-CoT. Least to Most prompting (LtM) [145] takes CoT prompting a step further by first breaking a problem into sub problems and then proceeds to solve each one independently. These sub-question answers are then synthesized to obtain the final response. Additionally, Tree of Thoughts (ToT) [139] and Graph of Thoughts (GoT) [8] use complex structures like trees and graphs to organize thoughts. These systems combine the way LLMs generate thoughts with search algorithms for systematic exploration, further enhancing their multi-step reasoning capabilities. In contrast to all these approaches, our work introduces a distinct three-step prompting framework. This framework emulates the problem-solving approach employed by humans when dealing with data rich in relationships. It enables users to transform a natural language paragraph into a graph, and subsequently, based on the query type, navigate this graph for the purpose of answering questions.

### 5.2.2 Integrate LLMs with Logical Inference

The most studied approach to reasoning since the earliest days of AI is logical inference [18]. Logical systems are fundamentally rule-based [96, 112], enabling the precise tracing of the specific path or rule that leads to a particular conclusion. This characteristic facilitates the establishment of proofs and verification processes, thereby ensuring that derived statements are sound based on the given axioms [112]. In contrast, LLMs, as neural-based models, often act as "black boxes," introducing a level of unconstrained behavior that poses challenges in following strict logical reasoning [76]. To enhance systematic reasoning, various strategies were proposed to integrate LLMs with classical logical inference algorithms such as forward chaining [27] and backward chaining [59]. Yet, applying these techniques in open domains presents significant challenges as they frequently necessitate supplementary context or logical rules to provide constraints. Creating

such logical rules can be demanding, especially with limited resources. Our proposed approach provides a systematic solution to address gaps in cases where explicit rules are absent. It achieves this by exploring the underlying graph structure of unstructured text, potentially improving the capability of Language Models (LLMs) to effectively traverse reasoning paths.

## 5.3 Multi-step Reasoning with KGs

KGs provide an effective way to explicitly organize information in the form of a structured graph. Multi-step reasoning naturally aligns with graph-based techniques, utilizing explicit pathways in the graph to represent the reasoning process [142, 19]. For example, multi-step reasoning has been formulated in a reinforcement learning setup, where a policy-based agent sequentially extends its inference path until it reaches a target [29, 108, 134, 68]. Moreover, to address the challenge of the more complex logical query answering in KGs, the query embedding method is proposed to conduct complex logical reasoning in the embedding space [51, 97, 98]. This method involves transforming a First-Order Logic (FOL) query into a vector within the embedding space and subsequently searching for entities in the KG that share similar embeddings. Despite significant efforts to use KGs for direct reasoning, these graphs are often domain-specific and suffer from data sparsity. This means they might not have enough information for accurate multi-step reasoning across various topics. LLMs, on the other hand, can access a vast range of unstructured text, offering broader knowledge and topic coverage. To combine the strengths of both KGs and LLMs, there have been attempts to use KGs as external tools to incorporate additional facts into the reasoning process [87]. For instance, MindMap [130] uses KGs to provide LLMs with up-to-date information and help them find reasoning paths. However, constructing and maintaining these KGs can be expensive and might even overwhelm LLMs with too much irrelevant information when addressing specific queries. In contrast, our approach takes a distinctive route. We hold the belief that natural language paragraphs inherently contain sufficient information for answering questions effectively. Instead of depending on external KGs, our proposition involves refining the organization of information within these paragraphs to enhance information retrieval and reasoning.

## 5.4 Structure Guided Prompt: Instructing Large Language Model in Multi-Step Reasoning by Exploring Graph Structure of the Text

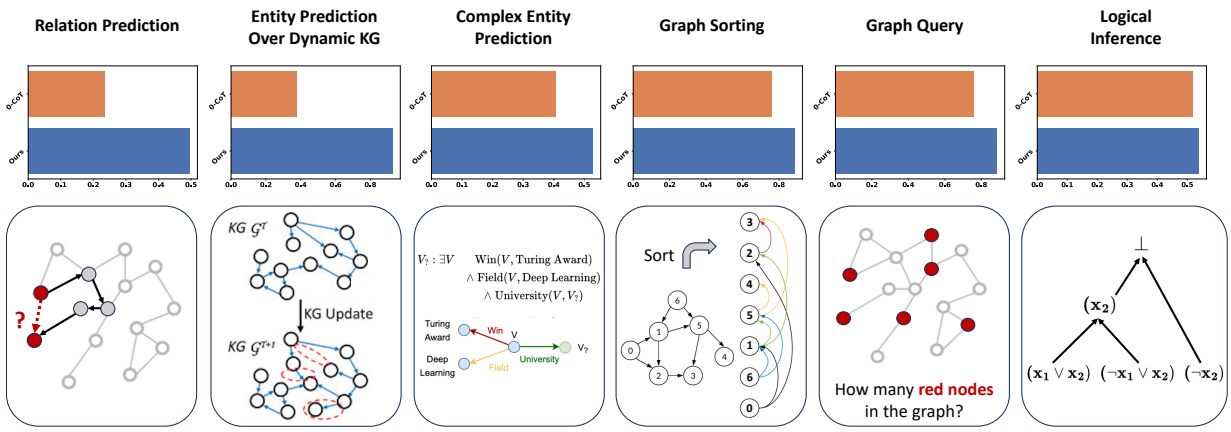



Figure 5.2: GPT-4's performance using 0-shot chain-of-thought (0-CoT) (represented as the orange bars) compared to the results of *Structure Guided Prompt* (represented as the blue bars) across a variety of tasks. It is evident that *Structure Guided Prompt* consistently and significantly outperforms the approach with 0-CoT.

Constructing and maintaining KGs to enhance LLMs can be expensive, and using an external KG may overwhelm LLMs with too much irrelevant information when addressing specific queries. In this section, we proposed a novel approach that takes a unique path. We firmly believe that natural language paragraphs inherently contain sufficient information for effectively answering questions. Rather than relying on external KGs, our approach centers on refining the organization of information within these paragraphs to enhance information comprehension and reasoning. Consequently, we introduce *Structure Guided Prompt*, a novel prompting framework designed to guide LLMs in multi-step reasoning. It explicitly converts unstructured text into a graph and instructs LLMs to navigate this graph to formulate responses in a *zero-shot setting*. Acknowledging the diversity of queries and their corresponding graph structures, we have categorized reasoning tasks into various categories as shown in Fig. 5.2. Each category is aligned with a unique graph structure. These categories present distinctive challenges for LLMs. In the evaluation, we compared the performance of both GPT-3.5 and the advanced GPT-4 model [86] when equipped with our proposed prompt.

Remarkably, our framework emerges as a catalyst, significantly enhancing the reasoning capabilities of LLMs across broader natural language scenarios. The results unequivocally demonstrate that *Structure Guided Prompt* empowers general-purpose LLMs to achieve competitive performance, underscoring its pivotal role in exploring the graph structure of text for instructing LLMs in multi-step reasoning.

### 5.4.1 Framework of Structure Guided Prompt

We propose *Structure Guided Prompt* [22], *a zero-shot prompting framework to guide LLMs in multi-step reasoning by explicitly converting unstructured text into a graph and instructs LLMs to navigate this graph using task-specific strategies to formulate responses.* The proposed framework is general, inherently task-agnostic, and capable of eliciting multi-step reasoning across broader natural language scenarios with a unified template.

#### 5.4.1.1 Three-stage prompting

Our three-stage prompting method, inspired by human problem-solving with graphs, involves: (1) Generating a graph from the given context; (2) Planning how to navigate the graph considering the tasks; (3) Executing the plan by traversing the graph to find the answer. This approach mirrors how humans tackle graph-based problems. To facilitate recognition, each stage of the prompt is color-coded: olive for the first stage, teal for the second, and violet for the third.

**Example** Let's illustrate the three-stage prompting using an example. Consider the following paragraph: *"Christian got his son, **Seth**, a car for his birthday. Christian and his brother Jonathan went to a basketball game. Jonathan's sister Ruth decided to tag along with them. Ruth invited her daughter Stephanie to lunch. Stephanie's brother **Jeremy** couldn't leave work to join them."* The question is to determine the family relationship between *Seth* and *Jeremy*.

**1st stage prompt: Concept Map Construction** In the first step, our goal is to convert an unstructured paragraph into a structured graph. Within this graph, each node corresponds to an entity, and the interconnecting edges depict the relationships linking these entities. Consider the

given example, its graph representation is given in the Fig. 5.3.

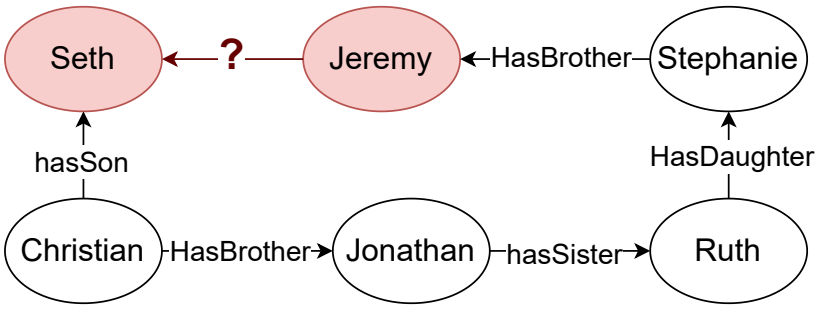



Figure 5.3: The graph representation of a story from CLUTRR dataset. our objective is to determine the family relationship between two nodes, *Seth* and *Jeremy*, which are highlighted in red.

**2nd stage prompt: Task-specific Planning** Fig. 5.3 demonstrates that, while the text and question seem straightforward, the graph reveals a complex path between *Seth* and *Jeremy*. Correctly navigating this path requires specific planning strategies that direct the reasoning process. It is important to note that these planning strategies are generally independent of the underlying data. The choice of strategy is task-specific. Consider the given example, to identify missing relationships between two entities (i.e., (*Seth*, ? ,*Jeremy*)), our method involves deducing this link by tracing a path between them. We start with the subject entity and iteratively explore the most relevant information, progressing step-by-step until we reach the object entity. From there, we deduce the missing relation by analyzing the path between the two entities. To enhance the versatility of our framework, the following section will discuss different planning strategies applicable for a range of tasks.

**3rd stage prompt: Execution with the Concept Map** Upon defining the task-specific planning strategy, we proceed to the execution phase, leveraging the concept map developed in 1st stage. This phase of instantiation enables us to address the specific problem in the given context. As illustrated in Fig 5.3, to derive the answer, we traverse the graph following the plan and carry out inference step by step: Step 1: Given Seth $\xleftarrow{\text{hasSon}}$ Christian $\xrightarrow{\text{hasBrother}}$ Jonathan, we have Seth $\xrightarrow{\text{hasUncle}}$ Jonathan; Step 2: Given Seth $\xrightarrow{\text{hasUncle}}$ Jonathan $\xrightarrow{\text{hasSister}}$ Ruth, we have Seth $\xrightarrow{\text{hasAunt}}$ Ruth; Step 3: Given Seth $\xrightarrow{\text{hasAunt}}$ Ruth $\xrightarrow{\text{hasDaughter}}$ Stephanie, we have Seth $\xrightarrow{\text{hasCousin}}$ Stephanie; Step 4: Given

Seth $\xrightarrow{\text{hasCousin}}$ Stephanie $\xrightarrow{\text{hasBrother}}$ Jeremy, we have Seth $\xrightarrow{\text{hasCousin}}$ Jeremy.

**A Complete Prompt.** By combining all three stages, we present the complete prompt for the given example: First, create a knowledge graph by extracting facts from each sentence in the given input story. Once this is done, I will pose a question. This question can be transformed into a triple (s, ?, o), where your primary task is to determine the missing relation ('?') that links the subject entity ('s') to the object entity ('o'). To begin, focus on the subject entity in this triple and choose the most relevant facts to expand from it. Step by step, progress towards the object entity, ensuring that each selected fact contributes to creating a link between the subject and object entities. Finally, utilize the established connection between the subject and object entities to answer the question.

### 5.4.2 Exploring Representative KG Reasoning Tasks

Our framework is inherently task-agnostic, designed to accommodate a wide range of tasks with versatility. To cater to this diversity, we establish task-specific planning in 2nd stage prompt, tailored to each unique task. This section outlines various planning approaches for different tasks, demonstrating the framework's adaptability.

#### 5.4.2.1 Relation Prediction

Relation prediction is a task focused on predicting the missing relations between two given entities, represented as $(h, ?, t)$. This task typically involves inferring the missing relations by tracing the path that links the target entities (i.e., $h$ and $t$) within the graph. We have discussed the planning strategies applicable to relation prediction task in Sec. 5.4.1.

#### 5.4.2.2 Entity Prediction

Entity prediction is a fundamental task in KGs that aims to infer the missing entity in a given query, such as $(h, r, ?)$ or $(?, r, t)$. For example, the question "Who currently holds the position of President in the USA?" can be structured as a link prediction task within a KG, seeking to resolve the query $(?, \text{isPresidentOf}, \text{USA})$. This query could be straightforward, in this paper, we

focus on more complex queries which require multi-step inference across various natural language scenarios.

**Entity Prediction over Dynamic KG** Given that the information in Knowledge Graphs (KGs) can change dynamically, each time step introduces new information for inference. Therefore, predicting entities within dynamic KGs necessitates a step-by-step understanding of the status at each time interval to effectively manage entity prediction. For instance, consider the scenario: *"Alice, Bob, and Claire are holding a white elephant gift exchange. At the start of the event, they are each holding a present of a different color: Alice has a yellow present, Bob has a brown present, and Claire has a blue present. As the event progresses, pairs of people swap gifts.* ***First****, Bob and Alice swap their gifts.* ***Then****, Claire and Alice swap their gifts.* ***Finally****, Bob and Alice swap their gifts. At the end of the event, what color gift does Bob have?"* This situation exemplifies entity prediction over dynamic KG, where the query can be structured as $(\text{Bob}, \text{hasGift}, ?)$. To accurately reflect the status at the event's conclusion, it is essential to capture changes at every time step, considering that each change depends on the previous time step. The primary planning strategy involves systematically tracking and recording the sequence of changes, with the KG at time step $t$ being modified based on the KG at the previous time step $t-1$.

**Complex Entity Prediction** While the previous method targets simpler one-hop queries in the form of $(h, r, ?)$, complex entity prediction aims to predict answers for queries with a more complex structure. For example, "Riom Trial was headed by the French general who reached what distinction?" is a complex query. This question's complexity goes beyond a straightforward relation, resembling a formal logic expression: $V_? := (\text{Riom Trial}, \text{wasHeadedBy}, V) \wedge (V, \text{Reached}, V_?)$. The bridging questions in HotpotQA provide typical examples of complex entity prediction, as illustrated in Fig. 5.4. Answering these requires aggregating and linking data from disparate sections of a text, following a specific sequence to construct the final answer. The primary planning strategy involves decomposing the question into simpler sub-questions and tackling these sub-questions sequentially, referencing the knowledge graph for information.

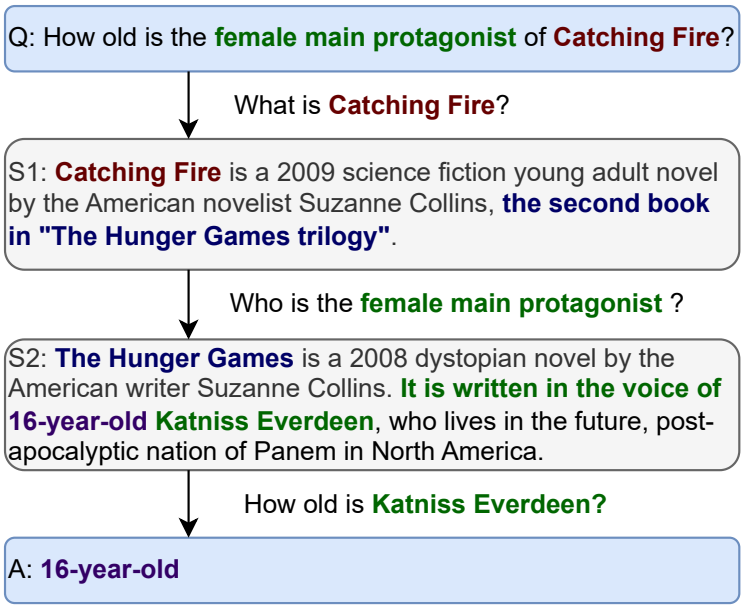



Figure 5.4: The bridging question in HotpotQA. It relies on multi-hop sequential reasoning to answer the question.

#### 5.4.2.3 Graph Sorting

Graph sorting task involves organizing entities within a graph according to a specified sequence. For instance, consider the scenario: *"The following paragraphs each describe three objects arranged in a fixed order. The statements are logically consistent within each paragraph. On a branch, there are three birds: a blue jay, a quail, and a falcon. The falcon is to the right of the blue jay. The blue jay is to the right of the quail. Which bird is the second one counting from the left?"* To solve this, the main approach is to arrange the birds in the correct order based on the given information and then determine the answer.

#### 5.4.2.4 Graph Query

Graph query task involves specifying a condition to retrieve specific data from a graph. For instance, consider the scenario: *"Here is a table where the first line is a header and each subsequent line is a penguin: name, age, height (cm), weight (kg) Louis, 7, 50, 11 Bernard, 5, 80, 13 Vincent, 9,*

*60, 11 Gwen, 8, 70, 15 For example: the age of Louis is 7, the weight of Gwen is 15 kg, the height of Bernard is 80 cm. How many penguins are more than 5 years old?"* This type of query can be expressed in SPARQL, a query language for databases, as follows:

```
SELECT (COUNT(?penguin) AS ?count)
WHERE {
    ?penguin ex:age ?age .
    FILTER (?age > 5)
}
```

The primary planning strategy involves identifying the condition and selecting the entities that meet the condition from the graph.

#### 5.4.2.5 Logical Inference

Logical inference and entailment are fundamental concepts in logic and reasoning, used in various fields to determine the logical relationships between statements. For instance, consider the scenario: *"Sentence 1: as the mass of a celestial object decreases, the surface gravity of that celestial object weakens. Sentence 2: less is the opposite of more. Sentence 3: as the force of gravity decreases, the weight of the object will decrease. Sentence 4: an astronaut is a kind of object. Sentence 5: The Earth has more mass than the Moon. Sentence 6: surface gravity is a kind of force of gravity. Why do astronauts weigh more on Earth than they do on the Moon?"* To answer this, we construct a logical sequence: Astronauts experience greater weight on Earth than on the Moon due to Earth's stronger gravitational force. This is inferred from the fact that Earth, having more mass than the Moon, exerts a stronger surface gravity. The primary planning strategy involves beginning with the subject entities referenced in the question and establishing a logical chain based on the provided context.

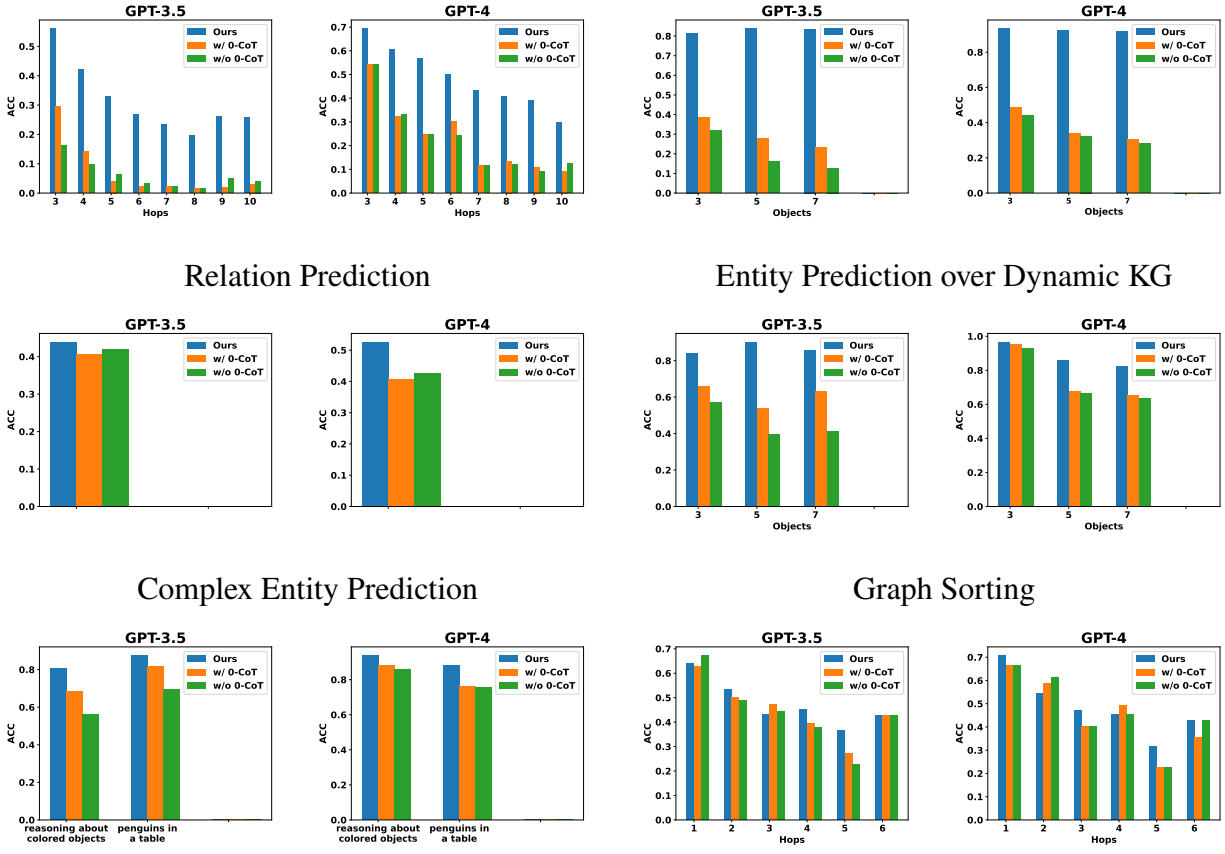



Figure 5.5: Main results. Different methods are illustrated through color-coded bars: blue bars indicate the results achieved using our *Structure Guided Prompt*, while orange bars show the performance with 0-shot chain-of-thought(0-CoT). Additionally, green bars depict the performance without 0-CoT. These results demonstrate that the *Structure Guided Prompt* consistently and significantly outperforms the other methods, both with and without 0-CoT, across GPT-3.5 and GPT-4 models.

### 5.4.3 Results

For each task, we evaluate the performance of two LLM models, GPT-4 (*gpt-4*) and GPT-3.5 (*gpt-3.5-turbo*) [86]. Since both methods are closed-source, we do not have specific information about their size, architecture, and pretraining particulars. For every task, we conduct a comparative analysis of our prompting framework against both with and without zero-shot chain-of-thought prompt (0-CoT), where 0-CoT encourages the model to engage in step-by-step reasoning by incorporating the phrase "Let's think step by step" in the prompts.

#### 5.4.3.1 Dataset

We have incorporated four datasets: *CLUTRR* [110], *BIG-bench-hard (BBH)* [119], *HotpotQA* [138] and *Entailment Bank* [28] in experiments. These datasets cover all six tasks discussed in Sec. 5.4.2.

#### 5.4.3.2 Analysis

**Relation Prediction** The most representative dataset for relation prediction task is *CLUTRR* [110]. It is a benchmark designed to infer the missing relationship between two individuals within a family network. To assess the complexity of the questions within the *CLUTRR*, we have organized them based on the length of the relational paths connecting the target family members, typically spanning 3 to 10 hops. As shown in Fig. 5.5, this task poses a formidable challenge for LLM, even when the path length is relatively short. Even with the utilization of GPT-4 with 0-CoT, satisfactory performance remains elusive. This observation underscores the inherent limitations of LLM in handling datasets with significant relational complexity. Furthermore, as the length of the relational paths extends, the challenge intensifies. In contrast, with our proposed *Structure Guided Prompt*, we can observe that it drastically increases the performance and suffers from less performance degradation when the path length increases.

**Entity Prediction over Dynamic KG**: We have included the *tracking shuffled objects* datasets from BBH [119] to assess the entity prediction performance within dynamic KG. These datasets are designed to infer the relative positions of various shuffled objects at the conclusion of a narrative.

The questions are organized according to the number of objects involved. As shown in Fig. 5.5, this task presents a significant challenge for LLM due to the requirement of maintaining an evolving graph representation at each time step when tracking shuffled objects. Given our proposed *Structure Guided Prompt*, which explicitly constructs and tracks changes within the KG, we can observe a remarkable performance enhancement (e.g., improve by 146% over GPT-4 w/ 0-CoT).

**Complex Entity Prediction** The bridging questions in *HotpotQA* [138] provide typical examples for complex entity prediction task. As shown in Fig. 5.5, while our proposed *Structure Guided Prompt* enhances performance, the improvement is not as significant as in other tasks. This is because the paragraphs in *HotpotQA* are exceptionally long, making it challenging for LLM to construct a KG that encompasses every piece of information within the context. Consequently, our proposed *Structure Guided Prompt* faces difficulty in further enhancing performance, especially with missing triples in the KG.

**Graph Sorting** In the graph sorting task, we have included the *logical deduction* datasets from BBH [119]. These datasets require to sort objects arranged in a line. The questions are organized according to the number of objects involved. As shown in Fig. 5.5, even though LLM already delivers impressive performance on this task, our proposed *Structure Guided Prompt* brings about further improvements, particularly as the number of involved objects increases.

**Graph Query** Within the graph query task, we have included *reasoning about colored objects* and *penguins in a table* datasets from BBH [119]. These datasets involve the selection and counting of objects that meet specific criteria from a given set of objects. As shown in Fig. 5.5, LLMs already deliver impressive performance on this task, but our proposed *Structure Guided Prompt* enhances performance even further.

**Logical Inference**: *Entailment Bank* [28] is a widely used dataset for multi-step entailment tasks involving logical reasoning. To assess the complexity of the questions within the *Entailment Bank*, we have categorized them based on the number of entailment steps required to arrive at an answer. As shown in Fig. 5.5, our proposed *Structure Guided Prompt* doesn't consistently improve performance for this task. The challenge lies in the fact that logical reasoning often demands a precise order when constructing the logical graph, with rules typically dictating a direction from the

premise to the conclusion. While following a forward chaining algorithm, one can readily employ the rules for logical inference sequentially, starting from known facts. However, in our scenario, we mix rules with facts and don't clearly distinguish between premises and conclusions within the rules. Consequently, even though we require LLMs to construct the logical graph, it remains challenging for LLMs to identify the correct logical order.

#### 5.4.3.3 Case Study

In this subsection, we aim to highlight the advanced capabilities and improvements our model, *Structure Guided Prompt*, offers over existing methodologies. We select the task of entity prediction over dynamic KG as a case for demonstration.

#### 5.4.3.4 Case Study of Success

We present a particular example where our *Structure Guided Prompt* delivers an accurate prediction, underlining its precision and reliability, especially when compared to the erroneous prediction made by 0-CoT in the same context.

**Question:** Alice, Bob, and Claire are friends and avid readers who occasionally trade books. At the start of the semester, they each buy one new book: Alice gets Hound of the Baskervilles, Bob gets The Pearl, and Claire gets The Odyssey. As the semester proceeds, they start trading around the new books. First, Bob and Claire swap books. Then, Claire and Alice swap books. Finally, Bob and Alice swap books. At the end of the semester, Alice has Options:

(A) Hound of the Baskervilles

(B) The Pearl

(C) The Odyssey

**0-CoT Answer:** (B) The Pearl ✗

***Structure Guided Prompt* Answer:** (C) The Odyssey

Explanation: Let's create the knowledge graph by extracting facts from each sentence in the

story.

1. Alice, Bob, and Claire are friends and avid readers who occasionally trade books.

- (Alice, is_friend_with, Bob)

- (Alice, is_friend_with, Claire)

- (Bob, is_friend_with, Claire)

- (Alice, is_avid_reader, True)

- (Bob, is_avid_reader, True)

- (Claire, is_avid_reader, True)

- (Alice, trades_books, True)

- (Bob, trades_books, True)

- (Claire, trades_books, True)

2. At the start of the semester, they each buy one new book: Alice gets Hound of the Baskervilles, Bob gets The Pearl, and Claire gets The Odyssey.

- (Alice, has_book, Hound of the Baskervilles)

- (Bob, has_book, The Pearl)

- (Claire, has_book, The Odyssey)

3. As the semester proceeds, they start trading around the new books. First, Bob and Claire swap books.

- (Bob, has_book, The Odyssey)

- (Claire, has_book, The Pearl)

4. Then, Claire and Alice swap books.

- (Claire, has_book, Hound of the Baskervilles)

- (Alice, has_book, The Pearl)

5. Finally, Bob and Alice swap books.

- (Bob, has_book, The Pearl)

- (Alice, has_book, The Odyssey)

So, at the end of the semester, Alice has The Odyssey. Therefore, option (C) is correct. ✔

**Analysis:** By explicitly converting unstructured text into a graph and directing LLMs to monitor the evolution of this dynamic graph, we can observe that LLMs accurately discern the altered object even after numerous iterations.

#### 5.4.3.5 Case Study of Failure

To comprehensively understand the limitations of our proposed methods, we provide a specific example demonstrating a scenario where our *Structure Guided Prompt* yields an incorrect prediction, in contrast to the accurate prediction made by 0-CoT under the same circumstances.

**Question:** Alice, Bob, and Claire are dancers at a square dance. At the start of a song, they each have a partner: Alice is dancing with Rodrigo, Bob is dancing with Jamie, and Claire is dancing with Lola. Throughout the song, the dancers often trade partners. First, Claire and Alice switch partners. Then, Bob and Claire switch partners. Finally, Claire and Alice switch partners. At the end of the dance, Alice is dancing with Options:

(A) Rodrigo

(B) Jamie

(C) Lola

**0-CoT Answer:** (B) Jamie ✔

***Structure Guided Prompt* Answer:** (C) Lola

Explanation:

Let's create the knowledge graph:

1. Alice, Bob, and Claire are dancers at a square dance.

- (Alice, is a dancer at, square dance)

- (Bob, is a dancer at, square dance)

- (Claire, is a dancer at, square dance)

2. At the start of a song, they each have a partner: Alice is dancing with Rodrigo, Bob is dancing with Jamie, and Claire is dancing with Lola.

- (Alice, is dancing with, Rodrigo)

- (Bob, is dancing with, Jamie)

- (Claire, is dancing with, Lola)

3. First, Claire and Alice switch partners.

- (Alice, is dancing with, Lola)

- (Claire, is dancing with, Rodrigo)

4. Then, Bob and Claire switch partners.

- (Bob, is dancing with, Rodrigo)

- (Claire, is dancing with, Jamie)

5. Finally, Claire and Alice switch partners.

- (Alice, is dancing with, Jamie) ✔

- (Claire, is dancing with, Lola)

So, at the end of the dance, Alice is dancing with Lola. Therefore, option (C) is correct. ✗

**Analysis:** As we highlight, while the inference process of our proposed model is accurate, it makes mistakes during the conclusion-drawing phase. This pattern has been widely observed in our experiments.

#### 5.4.3.6 Conclusion

A notable finding is that while the LLMs successfully adheres to the prompts to construct accurate KGs and navigates these KGs correctly according to task-specific strategic guidance, it often makes mistakes during the conclusion-drawing phase, even with correct inference results immediately preceding this stage. This issue could potentially be addressed by employing an additional LLM to verify the consistency of the generated content. We plan to explore this approach to further improve our framework in the future.

## 5.5 Summary and Discussions

LLMs often excel in simple reasoning tasks but struggle with multi-step reasoning. Graphs offer an effective way to model relational data and capture long-term dependencies among entities. In this section, we proposed to bridge this gap by introducing an innovative task-agnostic prompting framework, *Structure Guided Prompt*. This framework enhances the multi-step reasoning capabilities of LLMs within a zero-shot setting by systematically converting unstructured text into a graphical format and guiding LLMs in traversing this graph using task-specific strategies to construct responses. Our experiments show that our proposed framework significantly enhances the reasoning capabilities of LLMs, empowering them to excel in a broader spectrum of natural language scenarios.

# CHAPTER 6

# Future Works

In this dissertation, we have conducted an extensive investigation into the fascinating domain of integrating neural networks with symbolic reasoning for Knowledge Graph (KG) reasoning. Our focus has been directed toward three primary categories of reasoning tasks: knowledge graph completion and logical rule learning. For each task, we have provided a comprehensive overview of both symbolic and neural methodologies, delving into the synergistic potential that arises from their combined utilization. By leveraging the pattern and relationship learning abilities of neural networks alongside the logical deduction capabilities of symbolic reasoning, we can develop neural-symbolic reasoning systems that exhibit enhanced accuracy and interpretability. A succinct summary of each task is presented below.

- **Knowledge Graph Completion**: The objective of knowledge graph completion revolves around predicting missing facts by leveraging existing facts. We have provided an exhaustive introduction to various approaches for KG completion, including (1) traditional symbolic reasoning techniques, (2) recent representation learning-based methods, and (3) integration-based approaches that merge neural and symbolic components. While attempts have been made to combine neural and symbolic methods through probabilistic programming frameworks, scalability issues have hindered their effectiveness. To address these limitations, we introduce the UniKER algorithm [24] as the state-of-the-art (SOTA) solution that seamlessly combines symbolic reasoning and representation learning for KG completion tasks, achieving both effectiveness and efficiency.

- **Logical Rule Learning**: The goal of Logical Rule Learning is to automatically derive logical rules from KGs, which can be broadly classified into two categories: (1) traditional

search-based methods and (2) more recent neural-symbolic approaches. Despite achieving remarkable performance in learning logical rules, neural-symbolic integration methods heavily rely on observed data for rule identification. This poses challenges when attempting to identify rules that lack sufficient instances for support. To address this limitation, we present the state-of-the-art RLogic approach [23] and NCRL [21] which avoids sole dependence on rule instances by suggesting the direct learning of logical rules at the schema level using a representation learning-based model.

- **Combining Large Language Models and Knowledge Graphs**: While KGs are invaluable for their ability to depict complex relational data and trace extended entity relationships, their construction and evolution present significant challenges. Concurrently, large language models (LLMs) like ChatGPT and GPT-4 are advancing the domains of natural language processing and artificial intelligence with their remarkable emergent capabilities and adaptability. Thus, integrating LLMs with KGs can be synergistic, harnessing the strengths of both to enhance data representation and cognitive processing. Given that graphs excel in encoding data with complex relations and in capturing extended interrelations between entities, this chapter concentrates on strategies that combines LLMs and KGs. Specifically, we introduce our *Structure Guided Prompt* framework [22], an innovative three-stage task-agnostic prompting framework to explicitly convert unstructured text into a graph via LLMs and instruct them to navigate this graph using task-specific strategies to formulate responses.

We have also discussed the challenges and limitations of neural-symbolic reasoning systems, such as the interpretability issues associated with neural networks and the scalability concerns when handling large-scale data. However, we believe that the potential benefits of combining neural networks with symbolic reasoning for KG reasoning far outweigh the challenges and that this approach will continue to be an active area of research in the coming years. There are many exciting research frontiers in this field. We illustrate a few of them here:

- **Explainable neural-symbolic reasoning**: Improving the interpretability of neural-symbolic reasoning systems is a crucial direction for advancing the field. Users need to understand the

reasoning behind the system's decisions to build trust in it. While logical rule learning is one way to generate explanations, there are other possible ways that have not been explored in this dissertation. For instance, attention mechanisms can identify the most critical input data that led to a particular output, generating explanations that are easy to understand and helping users build trust in the system. Additionally, visualization techniques can be used to display the internal workings of the neural-symbolic reasoning system. For example, representing the decision-making process as a graph can enable users to understand how the system arrived at a specific conclusion. By exploring these and other approaches, the interpretability of neural-symbolic reasoning systems can be improved, leading to more trust and acceptance of the system in real-world applications.

- **Multi-modal reasoning**: Another crucial direction for advancing neural-symbolic reasoning systems is to develop multi-modal systems that can handle multiple types of data. In many real-world applications, such as autonomous driving or medical diagnosis, reasoning systems need to analyze data from different modalities, including text, image, and video. Developing multi-modal neural-symbolic reasoning systems that can handle and integrate different data types is therefore essential. Future research could focus on developing techniques for multi-modal feature extraction, where features are extracted from each modality and combined into a joint representation that can be used for reasoning. This can be achieved by using neural networks to extract features from each modality, then combining them using techniques such as late fusion or cross-modal attention mechanisms. These systems can provide more accurate and comprehensive reasoning capabilities and have the potential to impact a wide range of applications in fields such as healthcare, finance, and autonomous systems.

- **Combining KGs with Large Language Models** Large language models, such as ChatGPT and GPT-4, are revolutionizing the field of natural language processing and artificial intelligence, showcasing their impressive ability to generate human-like text across a wide range of tasks. However, despite their remarkable performance, LLMs have limitations when it comes to accessing and incorporating factual knowledge. These models primarily rely on patterns learned from massive amounts of text data, which may result in inaccuracies or the propaga-

tion of false information. This is where Knowledge Graphs (KGs), such as Wikipedia, come into play. KGs serve as structured repositories of factual knowledge, explicitly representing relationships and semantic connections between entities. By integrating KGs with LLMs, we can address the limitations of both approaches and leverage their respective advantages. LLMs can benefit from the external knowledge provided by KGs, enabling them to access accurate and reliable information for inference and improving their interpretability. This integration allows LLMs to go beyond the limitations of their training data and make more informed and contextually grounded responses. For example, OreoLM introduces a Knowledge Interaction Layer (KIL) that is inserted between the layers of a Language Model (LM). The KIL interacts with a KG reasoning module, enabling the discovery of various reasoning paths. These paths are then utilized by the reasoning module to generate answers. On the other hand, KGs face their own challenges. Constructing KGs requires significant manual effort and expertise, and they need to constantly evolve to capture new facts and represent previously unseen knowledge. Existing methods for KG construction may struggle to keep up with the rapid growth and dynamic nature of knowledge. By unifying LLMs and KGs, we can address these challenges by leveraging the language generation capabilities of LLMs to generate new facts and extend the representation of KGs. The integration of LLMs and KGs allows researchers to create more robust and knowledgeable AI systems that can leverage both structured knowledge and the generative power of language models.

- **Incorporating temporal reasoning** Many real-world applications involve reasoning about changes over time, such as in financial markets or epidemiological models. Future research could focus on developing techniques to incorporate temporal reasoning into neural-symbolic reasoning systems, allowing them to reason over dynamic KGs and make predictions about future trends. This work would involve addressing the challenges of representing and reasoning with temporal information within KGs, as well as designing algorithms that can effectively capture and utilize temporal dependencies. Additionally, efforts could be directed towards developing methods for learning temporal patterns and trends from historical data, enabling the system to extrapolate and forecast future states of the KG. By incorporating temporal

reasoning into KG reasoning, researchers can enhance the capabilities of existing neural-symbolic systems and empower them to provide more accurate and informed predictions, thereby facilitating better decision-making in various domains.

- **Federated KG reasoning** Traditional KG reasoning approaches typically rely on centralized data repositories, where the entire KG is stored and processed in a single location. However, in real-world scenarios, KGs are often distributed across multiple sources due to privacy concerns, ownership constraints, or scalability issues. Federated KG reasoning addresses this challenge by enabling reasoning and inference across distributed KGs without centralizing the data. Instead of sharing the complete KG, participants can selectively share specific parts or perform reasoning tasks locally on their private KGs. This approach ensures data privacy and allows participants to retain control over their sensitive or proprietary information while still benefiting from collective reasoning capabilities. By embracing federated KG reasoning, collaboration and knowledge sharing can thrive among diverse organizations or entities. Each participant in the federation can contribute their local KG, which may contain unique or domain-specific information. By federating these KGs, reasoning can be performed across multiple sources, resulting in a more comprehensive and diverse knowledge base. This collaborative aspect facilitates the generation of cross-domain insights and a broader understanding of complex phenomena.

Overall, combining neural networks with symbolic reasoning for KG reasoning is a rich and rapidly evolving research area. These additional future directions demonstrate the broad range of applications and challenges that these systems can address, and highlight the potential for continued innovation and impact in this field. We hope this dissertation has provided readers with a comprehensive understanding of this field and inspired them to explore further the exciting possibilities of neural-symbolic reasoning.

# BIBLIOGRAPHY


[1] Ralph Abboud, Ismail Ceylan, Thomas Lukasiewicz, and Tommaso Salvatori. Boxe: A box embedding model for knowledge base completion. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:9649–9661, 2020.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.

[3] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007.

[4] Stephen H Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. Hinge-loss markov random fields and probabilistic soft logic. *Journal of Machine Learning Research*, 18:1–67, 2017.

[5] Ivana Balažević, Carl Allen, and Timothy M Hospedales. Hypernetwork knowledge graph embeddings. In *International Conference on Artificial Neural Networks*, pages 553–565. Springer, 2019.

[6] Ivana Balazevic, Carl Allen, and Timothy M. Hospedales. Tucker: Tensor factorization for knowledge graph completion. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5184–5193. Association for Computational Linguistics, 2019.

[7] Julian Besag. Statistical analysis of non-lattice data. *Journal of the Royal Statistical Society Series D: The Statistician*, 24(3):179–195, 1975.

[8] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Michal Podstawski, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. *arXiv preprint arXiv:2308.09687*, 2023.

[9] Guram Bezhanishvili and Wesley Fussner. An introduction to symbolic logic. *Convergence*, 2013.

[10] Roi Blanco, Berkant Barla Cambazoglu, Peter Mika, and Nicolas Torzec. Entity recommendations in web search. In *Proceedings of the International Semantic Web Conference*, pages 33–48. Springer, 2013.

[11] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

[12] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of*

*the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. ACM, 2008.

[13] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2787–2795, 2013.

[14] Patricia Branum and Bethany Sehon. Knowledge graph pilot improves data quality while providing a customer 360 view. In *Knowledge Graph Conference.(Invited talk)*, 2019.

[15] Liwei Cai and William Yang Wang. Kbgan: Adversarial learning for knowledge graph embeddings. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 1470–1480, 2018.

[16] Alison Callahan, Jose Cruz-Toledo, Peter Ansell, and Michel Dumontier. Bio2rdf release 2: improved coverage, interoperability and provenance of life science linked data. In *Proceedings of the Extended Semantic Web Conference (ESWC)*, pages 200–212. Springer, 2013.

[17] Chain Monte Carlo. Markov chain monte carlo and gibbs sampling. *Lecture notes for EEB*, 581, 2004.

[18] Rudolf Carnap. *Introduction to symbolic logic and its applications*. Courier Corporation, 2012.

[19] Xiaojun Chen, Shengbin Jia, and Yang Xiang. A review: Knowledge reasoning over knowledge graph. *Expert Systems with Applications*, 141:112948, 2020.

[20] Xuelu Chen, Muhao Chen, Weijia Shi, Yizhou Sun, and Carlo Zaniolo. Embedding uncertain knowledge graphs. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, pages 3363–3370. AAAI Press, 2019.

[21] Kewei Cheng, Nesreen Ahmed, and Yizhou Sun. Neural compositional rule learning for knowledge graph reasoning. In *International Conference on Learning Representations (ICLR)*.

[22] Kewei Cheng, Nesreen K Ahmed, Theodore Willke, and Yizhou Sun. Structure guided prompt: Instructing large language model in multi-step reasoning by exploring graph structure of the text. *arXiv preprint arXiv:2402.13415*, 2024.

[23] Kewei Cheng, Jiahao Liu, Wei Wang, and Yizhou Sun. Rlogic: Recursive logical rule learning from knowledge graphs. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 179–189, 2022.

[24] Kewei Cheng, Ziqing Yang, Ming Zhang, and Yizhou Sun. Uniker: A unified framework for combining embedding and definite horn rule reasoning for knowledge graph inference.

[25] William W Cohen. Tensorlog: A differentiable deductive database. *arXiv preprint arXiv:1605.06523*, 2016.

[26] Michael Collins, Sanjoy Dasgupta, and Robert E Schapire. A generalization of principal components analysis to the exponential family. In *Nips*, volume 13, page 23, 2001.

[27] Antonia Creswell, Murray Shanahan, and Irina Higgins. Selection-inference: Exploiting large language models for interpretable logical reasoning. *arXiv preprint arXiv:2205.09712*, 2022.

[28] Bhavana Dalvi, Peter Jansen, Oyvind Tafjord, Zhengnan Xie, Hannah Smith, Leighanna Pipatanangkura, and Peter Clark. Explaining answers with entailment trees. *arXiv preprint arXiv:2104.08661*, 2021.

[29] Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017.

[30] Rajarshi Das, Arvind Neelakantan, David Belanger, and Andrew McCallum. Chains of reasoning over entities, relations, and text using recurrent neural networks. *arXiv preprint arXiv:1607.01426*, 2016.

[31] Luc De Raedt, Anton Dries, Ingo Thon, Guy Van den Broeck, and Mathias Verbeke. Inducing probabilistic relational rules from probabilistic examples. In *Proceedings of 24th international joint conference on artificial intelligence (IJCAI)*, volume 2015, pages 1835–1842. IJCAI-INT JOINT CONF ARTIF INTELL, 2015.

[32] Luc De Raedt and Kristian Kersting. Probabilistic inductive logic programming. In *Probabilistic Inductive Logic Programming*, pages 1–27. Springer, 2008.

[33] Thomas Demeester, Tim Rocktäschel, and Sebastian Riedel. Lifted rule injection for relation embeddings. *arXiv preprint arXiv:1606.08359*, 2016.

[34] Woodrow W Denham. *The detection of patterns in Alyawara nonverbal behavior*. PhD thesis, University of Washington, Seattle., 1973.

[35] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[36] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[37] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 601–610, 2014.

[38] Xin Luna Dong. Building a broad knowledge graph for products. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 25–25. IEEE, 2019.

[39] Sebastijan Dumancic, Alberto García-Durán, and Mathias Niepert. On embeddings as an alternative paradigm for relational learning. *CoRR*, 2018.

[40] Ronald Fagin. Combining fuzzy information from multiple systems. *Journal of Computer and System Sciences*, 58(1):83–99, 1999.

[41] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M Suchanek. Fast rule mining in ontological knowledge bases with amie+. *The VLDB Journal—The International Journal on Very Large Data Bases*, 24(6):707–730, 2015.

[42] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. Amie: association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the International World Wide Web Conference (WWW)*, pages 413–422. ACM, 2013.

[43] Zhenxiang Gao, Pingjian Ding, and Rong Xu. Kg-predict: a knowledge graph computational framework for drug repurposing. *Journal of biomedical informatics*, 132:104133, 2022.

[44] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[45] Jens Graupmann, Ralf Schenkel, and Gerhard Weikum. The SphereSearch engine for unified ranked retrieval of heterogeneous XML and web documents. In *Proceedings of the 31st international conference on very large data bases*, pages 529–540. VLDB Endowment, 2005.

[46] Lingbing Guo, Zequn Sun, and Wei Hu. Learning to exploit long-term relational dependencies in knowledge graphs. In *International Conference on Machine Learning (ICML)*, pages 2505–2514. PMLR, 2019.

[47] Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. Jointly embedding knowledge graphs and logical rules. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 192–202, 2016.

[48] Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. Knowledge graph embedding with iterative guidance from soft rules. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, volume 32, 2018.

[49] Petr Hájek. *Metamathematics of fuzzy logic*, volume 4. Springer Science & Business Media, 2013.

[50] Petr Hájek, Lluís Godo, and Francesc Esteva. A complete many-valued logic with product-conjunction. *Archive for mathematical logic*, 35(3):191–208, 1996.

[51] William L. Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. Embedding logical queries on knowledge graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2030–2041, 2018.

[52] L Vivek Harsha Vardhan, Guo Jia, and Stanley Kok. Probabilistic logic graph attention networks for reasoning. In *Companion Proceedings of the Web Conference 2020*, WWW '20, page 669–673, New York, NY, USA, 2020. Association for Computing Machinery.

[53] Geoffrey E Hinton et al. Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, volume 1, page 12. Amherst, MA, 1986.

[54] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 2013.

[55] Alfred Horn. On sentences which are true of direct unions of algebras1. *The Journal of Symbolic Logic*, 16(1):14–21, 1951.

[56] Arcchit Jain, Tal Friedman, Ondrej Kuzelka, Guy Van den Broeck, and Luc De Raedt. Scalable rule learning in probabilistic knowledge bases. *Automated Knowledge Base Construction*, 2019.

[57] Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. Knowledge graph embedding via dynamic mapping matrix. In *Proceedings of the Annual Meeting of Associations for Computational Linguistics (ACL)*, pages 687–696. The Association for Computer Linguistics, 2015.

[58] Xiaotian Jiang, Quan Wang, and Bin Wang. Adaptive convolution for multi-relational learning. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 978–987, 2019.

[59] Seyed Mehran Kazemi, Najoung Kim, Deepti Bhatia, Xin Xu, and Deepak Ramachandran. Lambada: Backward chaining for automated reasoning in natural language. *arXiv preprint arXiv:2212.13894*, 2022.

[60] Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4284–4295, 2018.

[61] Angelika Kimmig, Stephen Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. A short introduction to probabilistic soft logic. In *Proceedings of the NIPS Workshop on Probabilistic Programming: Foundations and Applications*, 2012.

[62] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[63] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

[64] Ni Lao, Tom Mitchell, and William Cohen. Random walk inference and learning in a large scale knowledge base. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 529–539, 2011.

[65] Nada Lavrac and Saso Dzeroski. Inductive logic programming. In *WLP*, pages 146–160. Springer, 1994.

[66] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[67] Clarence Irving Lewis, Cooper Harold Langford, and P Lamprecht. *Symbolic logic*, volume 170. Dover Publications New York, 1959.

[68] Xi Victoria Lin, Richard Socher, and Caiming Xiong. Multi-hop knowledge graph reasoning with reward shaping. *arXiv preprint arXiv:1808.10568*, 2018.

[69] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2015.

[70] Hanxiao Liu, Yuexin Wu, and Yiming Yang. Analogical inference for multi-relational embeddings. In *International Conference on Machine Learning (ICML)*, pages 2168–2178. PMLR, 2017.

[71] Shengyao Lu, Bang Liu, Keith G Mills, SHANGLING JUI, and Di Niu. R5: Rule discovery with reinforced and recurrent relational reasoning. In *International Conference on Learning Representations*, 2022.

[72] Denis Lukovnikov, Asja Fischer, Jens Lehmann, and Sören Auer. Neural network-based question answering over knowledge graphs on word and character level. In *Proceedings of the International World Wide Web Conference (WWW)*, pages 1211–1220. International World Wide Web Conferences Steering Committee, 2017.

[73] Edgar Meij. Understanding news using the bloomberg knowledge graph. *Invited talk at the Big Data Innovators Gathering (TheWebConf).*, 2019.

[74] Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. Anytime bottom-up rule learning for knowledge graph completion. In *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 3137–3143, 2019.

[75] George A Miller. *WordNet: An electronic lexical database*. MIT press, 1998.

[76] Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. Recent advances in natural language processing via large pre-trained language models: A survey. *ACM Computing Surveys*, 56(2):1–40, 2023.

[77] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994.

[78] Radford M Neal and Geoffrey E Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer, 1998.

[79] Arvind Neelakantan, Benjamin Roth, and Andrew McCallum. Compositional vector space models for knowledge base completion. *arXiv preprint arXiv:1504.06662*, 2015.

[80] David Newman. Knowledge graphs and ai: The future of financial data. In *Knowledge Graph Conference.(Invited talk)*, 2019.

[81] Dai Quoc Nguyen, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Phung. A novel embedding model for knowledge base completion based on convolutional neural network. *arXiv preprint arXiv:1712.02121*, 2017.

[82] Maximilian Nickel, Lorenzo Rosasco, Tomaso A Poggio, et al. Holographic embeddings of knowledge graphs. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, pages 1955–1961. AAAI Press, 2016.

[83] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *International Conference on Machine Learning (ICML)*, volume 11, pages 809–816, 2011.

[84] Nils J Nilsson and Nils Johan Nilsson. *Artificial intelligence: a new synthesis*. Morgan Kaufmann, 1998.

[85] Feng Niu, Christopher Ré, AnHai Doan, and Jude Shavlik. Tuffy: Scaling up statistical inference in markov logic networks using an rdbms. *Proceedings of the VLDB Endowment*, 4(6):373–384, 2011.

[86] OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023.

[87] Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[88] Christos H Papadimitriou. Computational complexity. In *Encyclopedia of computer science*, pages 260–265. 2003.

[89] Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. Art: Automatic multi-step reasoning and tool-use for large language models, 2023.

[90] RJ Pittman. Cracking the code on conversational commerce. https://www.ebayinc.com/stories/news/cracking-the-code-on-conversational-commerce/, 2017. [Online; accessed 06-Apr-2017].

[91] Hoifung Poon and Pedro Domingos. Sound and efficient inference with probabilistic and deterministic dependencies. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, volume 6, pages 458–463, 2006.

[92] Hoifung Poon, Pedro M Domingos, and Marc Sumner. A general method for reducing the complexity of relational inference and its application to mcmc. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, volume 8, pages 1075–1080. Chicago, IL, 2008.

[93] Richard Qian. Understand your world with bing. https://blogs.bing.com/search/2013/03/21/understand-your-world-with-bing/, 2013. [Online; accessed 21-Mar-2013].

[94] Meng Qu, Junkun Chen, Louis-Pascal Xhonneux, Yoshua Bengio, and Jian Tang. Rnnlogic: Learning logic rules for reasoning on knowledge graphs. *arXiv preprint arXiv:2010.04029*, 2020.

[95] Meng Qu and Jian Tang. Probabilistic logic neural networks for reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 7710–7720, 2019.

[96] J. Ross Quinlan. Learning logical definitions from relations. *Machine learning*, 5(3):239–266, 1990.

[97] Hongyu Ren, Weihua Hu, and Jure Leskovec. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2020.

[98] Hongyu Ren and Jure Leskovec. Beta embeddings for multi-hop logical reasoning in knowledge graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[99] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*, 2012.

[100] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 62(1-2):107–136, 2006.

[101] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3788–3800, 2017.

[102] Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. Injecting logical background knowledge into embeddings for relation extraction. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 1119–1129, 2015.

[103] Ali Sadeghian, Mohammadreza Armandpour, Patrick Ding, and Daisy Zhe Wang. Drum: End-to-end differentiable rule mining on knowledge graphs. *arXiv preprint arXiv:1911.00055*, 2019.

[104] Eric Salvat and Marie-Laure Mugnier. Sound and complete forward and backward chainings of graph rules. In *International Conference on Conceptual Structures*, pages 248–262. Springer, 1996.

[105] Abulhair Saparov and He He. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. *arXiv preprint arXiv:2210.01240*, 2022.

[106] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*, volume 10843 of *Lecture Notes in Computer Science*, pages 593–607. Springer, 2018.

[107] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[108] Yelong Shen, Jianshu Chen, Po-Sen Huang, Yuqing Guo, and Jianfeng Gao. M-walk: Learning to walk over graphs using monte carlo tree search. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018.

[109] Parag Singla and Pedro M Domingos. Lifted first-order belief propagation. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, volume 8, pages 1094–1099, 2008.

[110] Koustuv Sinha, Shagun Sodhani, Jin Dong, Joelle Pineau, and William L Hamilton. Clutrr: A diagnostic benchmark for inductive reasoning from text. *arXiv preprint arXiv:1908.06177*, 2019.

[111] Koustuv Sinha, Shagun Sodhani, Joelle Pineau, and William L Hamilton. Evaluating logical generalization in graph neural networks. *arXiv preprint arXiv:2003.06560*, 2020.

[112] Steven A Sloman. The empirical case for two systems of reasoning. *Psychological bulletin*, 119(1):3, 1996.

[113] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. *Advances in Neural Information Processing Systems (NeurIPS)*, 26, 2013.

[114] Robert Speer, Joshua Chin, and Catherine Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2017.

[115] Frank Spitzer. *Principles of random walk*, volume 34. Springer Science & Business Media, 2013.

[116] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. YAGO: a core of semantic knowledge. In *Proceedings of the International World Wide Web Conference (WWW)*, pages 697–706. ACM, 2007.

[117] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations*, 2018.

[118] Zhiqing Sun, Shikhar Vashishth, Soumya Sanyal, Partha Talukdar, and Yiming Yang. A re-evaluation of knowledge graph completion methods. *arXiv preprint arXiv:1911.03903*, 2019.

[119] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, , and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

[120] Damian Szklarczyk, Annika L Gable, David Lyon, Alexander Junge, Stefan Wyder, Jaime Huerta-Cepas, Milan Simonovic, Nadezhda T Doncheva, John H Morris, Peer Bork, et al. String v11: protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. *Nucleic acids research*, 47(D1):D607–D613, 2019.

[121] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd workshop on continuous vector space models and their compositionality*, pages 57–66, 2015.

[122] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *International Conference on Machine Learning (ICML)*, pages 2071–2080, 2016.

[123] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, Nilesh Agrawal, and Partha Talukdar. Interacte: Improving convolution-based knowledge graph embeddings by increasing feature interactions. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, volume 34, pages 3009–3016, 2020.

[124] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems (NeurIPS)*, 30, 2017.

[125] Martin J Wainwright, Michael I Jordan, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305, 2008.

[126] Quan Wang, Pingping Huang, Haifeng Wang, Songtai Dai, Wenbin Jiang, Jing Liu, Yajuan Lyu, Yong Zhu, and Hua Wu. Coke: Contextualized knowledge graph embedding. *arXiv preprint arXiv:1911.02168*, 2019.

[127] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

[128] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2014.

[129] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

[130] Yilin Wen, Zifeng Wang, and Jimeng Sun. Mindmap: Knowledge graph prompting sparks graph of thoughts in large language models. *arXiv preprint arXiv:2308.09729*, 2023.

[131] Alfred North Whitehead and Bertrand Russell. *Principia mathematica to* 56*, volume 2. Cambridge University Press, 1997.

[132] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[133] Chenyan Xiong, Russell Power, and Jamie Callan. Explicit semantic ranking for academic search via knowledge graph embedding. In *Proceedings of the International World Wide Web Conference (WWW)*, pages 1271–1279. International World Wide Web Conferences Steering Committee, 2017.

[134] Wenhan Xiong, Thien Hoang, and William Yang Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. In Martha Palmer, Rebecca Hwa, and Sebastian Riedel, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 564–573. Association for Computational Linguistics, 2017.

[135] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014.

[136] Fan Yang, Zhilin Yang, and William W Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2319–2328, 2017.

[137] Yuan Yang and Le Song. Learn to explain efficiently via neural logic inductive learning. *arXiv preprint arXiv:1910.02481*, 2019.

[138] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.

[139] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

[140] Jonathan S Yedidia, William T Freeman, and Yair Weiss. Generalized belief propagation. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 689–695, 2001.

[141] Scott Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the Joint Conference of the 53rd Annual Meeting of the ACL and the 7th International Joint Conference on Natural Language Processing of the AFNLP*, 2015.

[142] Jing Zhang, Bo Chen, Lingxi Zhang, Xirui Ke, and Haipeng Ding. Neural, symbolic and neural-symbolic reasoning on knowledge graphs. *AI Open*, 2:14–35, 2021.

[143] Ningyu Zhang, Qianghuai Jia, Shumin Deng, Xiang Chen, Hongbin Ye, Hui Chen, Huaixiao Tou, Gang Huang, Zhao Wang, Nengwei Hua, et al. Alicg: Fine-grained and evolvable conceptual graph construction for semantic search at alibaba. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 3895–3905, 2021.

[144] Yuyu Zhang, Xinshi Chen, Yuan Yang, Arun Ramamurthy, Bo Li, Yuan Qi, and Le Song. Can graph neural networks help logic reasoning? *arXiv preprint arXiv:1906.02111*, 2019.

[145] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.