Bi-directional Learning of Logical Rules with Type Constraints for Knowledge Graph Completion

Kunxun Qi
School of Computer Science and Engineering, Sun Yat-sen University
Guangzhou, China
qikx@mail2.sysu.edu.cn

Jianfeng Du*
Guangdong University of Foreign Studies, Guangzhou, China
Bigmath Technology, Shenzhen
jfdu@gdufs.edu.cn

Hai Wan*
School of Computer Science and Engineering, Sun Yat-sen University
Guangzhou, China
wanhai@mail.sysu.edu.cn

Abstract

Knowledge graph completion (KGC) aims to infer missing facts from existing facts. Learning logical rules plays a pivotal role in KGC, as logical rules excel in explaining why a missing fact is inferred. Most existing rule learning methods focus merely on learning chain-like rules, neglecting type constraints on entities. In practice, type constraints are crucial in expressing precise rules. Therefore, we propose a novel formalism for logical rules named *TC-rules*, which complements chain-like rules with both explicit and implicit type constraints on entity variables. Accordingly, we propose an end-to-end approach to effectively learn TC-rules, by parameterizing a neural model to simulate the inference of TC-rules. Considering that existing end-to-end methods learn two different sets of logical rules to respectively answer a head query $(?, r^{\text{new}}, t)$ and a tail query $(h, r^{\text{new}}, ?)$, leading to confusing explanations for supporting a new fact (h, r^{new}, t), we propose a bi-directional learning mechanism to ensure that the TC-rules learnt for answering $(?, r^{\text{new}}, t)$ are the same as the TC-rules learnt for answering $(h, r^{\text{new}}, ?)$. Experimental results on eight benchmark datasets demonstrate that the proposed method outperforms state-of-the-art rule learners in both the link prediction task and the triple classification task. Furthermore, our case study confirms that expressive TC-rules can be extracted from the parameter assignment of the learnt neural model.

CCS Concepts

• **Computing methodologies** → **Statistical relational learning**.

Keywords

Rule learning, Knowledge graph completion, Neural network

ACM Reference Format:
Kunxun Qi, Jianfeng Du, and Hai Wan. 2024. Bi-directional Learning of Logical Rules with Type Constraints for Knowledge Graph Completion. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM '24), October 21–25, 2024, Boise, ID, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3627673.3679695

*Both authors are corresponding authors.

CIKM '24, October 21–25, 2024, Boise, ID, USA

ACM ISBN 979-8-4007-0436-9/24/10
https://doi.org/10.1145/3627673.3679695

1 Introduction

Knowledge graphs (KGs) have been widely used in many real-world applications, including question answering [18], recommendation [15] and information retrieval [38]. A knowledge graph is usually represented as a directed graph with vertices labeled by entities and edges labeled by relations. It consists of a set of *facts* (also called *triples*) of the form (h, r, t), where h is the *head* entity, r the *relation* and t the *tail* entity. Nowadays large-scale KGs like Freebase [3], Wikidata [35], YAGO [29] and DBpedia [1] consist of hundreds of millions of facts, serving massive downstream applications. Nevertheless, KGs are usually far from complete since collecting the complete set of facts is laborious and error-prone. Thus, it is crucial to infer missing facts from existing facts in a KG. This task is often referred to as *knowledge graph completion* (KGC).

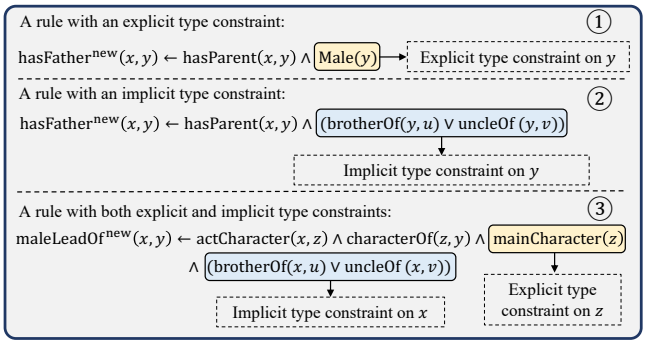


Figure 1: Examples of rules with different type constraints, where r^{new} denotes the relation of a new fact.

Logical rules play a pivotal role in KGC, as they can provide explanations for inferred facts. Most existing rule learning methods focus merely on learning chain-like rules, where every body atom in the rule shares one variable with the previous body atom and the other variable with the next body atom. This form of rules does not express type constraints on entity variables, which may lead to wrong predictions. Figure 1 illustrates an example. In the first sub-figure, a chain-like rule is depicted as "$\text{hasFather}^{\text{new}}(x, y) \leftarrow \text{hasParent}(x, y)$", extended with a type constraint "$\text{Male}(y)$". This kind of type constraints relies on type information explicitly declared on entities, and thus we call them *explicit type constraint*. Without this type constraint, we may inaccurately infer a false triple $(h, \text{hasFather}, t)$ to be true, where t is the mother of h. To address this issue, typed rules [37] are proposed to extend chain-like rules with unary atoms to express explicit type constraints on entities. However, it is challenging to apply typed rules in real-world

scenarios, as type information is usually missing or incomplete in real-world KGs, including most benchmark datasets.

In this paper, we argue that type constraints can be implicitly inferred by neighbor triples in the KG. The second sub-figure in Figure 1 showcases a rule that expresses a similar meaning as the first rule, saying that "the parent of x is y and y is the brother of u or the uncle of v implies that the father of x is y", where "y is the brother of u or the uncle of v" implies that the type of y is male. This type constraint on y is not explicitly given and thus we call it an *implicit type constraint*. This kind of constraints provides an alternative way to express type information on entities. The third sub-figure in Figure 1 illustrates a rule that involves both explicit and implicit type constraints.

On the other hand, explainability is a desirable property that KGC pursues. However, most existing methods, especially end-to-end methods [25, 40, 41], usually learn two different sets of logical rules to answer head queries $(?, r^{\text{new}}, t)$ and tail queries $(h, r^{\text{new}}, ?)$, respectively. Consider the second rule in Figure 1 again. For inferring that t is an answer to the query $(h, \text{hasFather}, ?)$, the rule "$\text{hasFather}^{\text{new}}(x, y) \leftarrow \text{hasParent}(x, y) \wedge (\text{brotherOf}(y, u) \vee \text{uncleOf}(y, v))$" is learnt from a training set. But for inferring that h is an answer to $(?, \text{hasFather}, t)$, it is possible that the rule "$\text{hasFather}^{\text{new}}(x, y) \leftarrow \text{hasParent}(x, y)$" is learnt since type constraints on the given entity x may not affect the inference of answers in the same training set, while a model tends to learn shorter rules based on Occam's Razor. This yields two confusing explanations to support $(h, \text{hasFather}, t)$, where one is $(h, \text{hasParent}, t)$ and the other is $(h, \text{hasParent}, t) \wedge ((t, \text{brotherOf}, e_1) \vee (t, \text{uncleOf}, e_2))$, ruining the explainability for KGC results. Thus it is required to compute consistent explanations to support a new triple.

To incorporate both explicit and implicit types while avoiding confusions in explanations, we propose a novel end-to-end approach for rule-based KGC. First of all, we introduce a new formalism for rules to be learnt, named *Type-constraint enhanced Chain-like rules* or *TC-rules* for short. A TC-rule is extended from a chain-like rule by adding, for every entity variable in the rule body, a disjunction of unary atoms to express explicit types on the entity variable as well as a disjunction of binary atoms to express implicit types on the entity variable. We then propose a neural model named *Type Constraint Learning Model* or *TCLM* for short to approximate TC-rules. The overview of TCLM is shown in Figure 2. TCLM generates approximate TC-rules one by one, where in the course of generating a TC-rule, it generates the chained atoms and type constraints on variables step by step. We show that a set of TC-rules can be encoded by a certain parameter assignment of TCLM (see Theorem 1). To enable extracting the same set of TC-rules to answer both head and tail queries, we propose a bi-directional learning mechanism through parameter sharing. We show that a triple being true or false can be explained in the same way from both the angle of answering head queries and the angle of answering tail queries (see Theorem 2).

Experimental results on eight benchmark datasets demonstrate that TCLM is superior to state-of-the-art (SOTA) rule learners in both the link prediction task and the triple classification task. Besides, TCLM is able to complete the learning process and the evaluation process within a reasonable time. Our case study further confirms that expressive TC-rules can be extracted from the parameter assignment of the learnt model.

The main contributions of this work are three-fold.

(1) We extend the formalism of chain-like rules to TC-rules, which are able to concisely express both explicit and implicit type constraints on entity variables.
(2) We propose an end-to-end learning approach to approximating TC-rules, which is equipped with a bi-directional learning mechanism to guarantee that the same set of TC-rules can be extracted to answer both head and tail queries.
(3) We conduct extensive experiments on eight benchmark datasets to demonstrate the effectiveness of TCLM.

2 Preliminaries

Knowledge Graph. Let $\mathcal{E}$ be a set of entities, $\mathcal{R}$ a set of relations and C a set of types. A *knowledge graph* $\mathcal{G}$ can be separated into two parts, i.e., $\mathcal{G} = \mathcal{G}_{\text{rel}} \cup \mathcal{G}_{\text{type}}$, where $\mathcal{G}_{\text{rel}} = \{(h_i, r_i, t_i)\}_{1 \le i \le N_{\text{rel}}}$, $\mathcal{G}_{\text{type}} = \{(e_i, \text{Type}, c_i)\}_{1 \le i \le N_{\text{type}}}$, N_{rel} denotes the number of triples in $\mathcal{G}_{\text{rel}}$, N_{type} the number of triples in $\mathcal{G}_{\text{type}}$, $h_i \in \mathcal{E}$ (resp. $t_i \in \mathcal{E}$ or $r_i \in \mathcal{R}$) is the *head* entity (resp. *tail* entity or *relation*) for the i^{th} triple in $\mathcal{G}_{\text{rel}}$, and $e_i \in \mathcal{E}$ (resp. $c_i \in C$) is the entity (resp. type) for the i^{th} triple in $\mathcal{G}_{\text{type}}$. By r^- we denote the inverse relation of $r \in \mathcal{R}$. The set of inverse relations for $\mathcal{R}$, namely $\{r^- \mid r \in \mathcal{R}\}$, is denoted by $\mathcal{R}^-$. Accordingly, the equivalent set of triples for $\mathcal{G}_{\text{rel}}$ composed by inverse relations, namely $\{(t, r^-, h) \mid (h, r, t) \in \mathcal{G}_{\text{rel}}\}$, is denoted by $\mathcal{G}^-_{\text{rel}}$.

Inference Rule. An *atom* is a basic first-order logic formula of the form $p(t_1, \ldots, t_n)$, where p is a *predicate* and $t_1, \ldots, t_n$ are terms that denote either constants or variables. An r-specific inference rule R for the target relation r can be written of the form $r^{\text{new}}(x, y) \leftarrow \exists \vec{z} : \varphi(x, y, \vec{z})$, where $\varphi(x, y, \vec{z})$ is a conjunction of atoms on variables x, y and $\vec{z}$, and r^{new} denotes the predicate of a new fact that inferred by a r-specific inference rule. The part of R at the left (resp. right) of $\leftarrow$ is called the *head* (resp. *body*) of R. By H_R and B_R we denote the atom in the head of R and the set of atoms in the body of R, respectively. Note that we distinguish r^{new} from r to avoid recursive inference of new facts on r. An atom or a rule is *ground* if it does not contain any variable. An r-specific inference rule R is called a fact if B_R is empty and H_R is ground. To uniformly represent r-specific inference rules using fixed-length bodies, we introduce the *identity relation* (denoted by I) to rule bodies. For example, $r^{\text{new}}(x, y) \leftarrow p(x, y)$ can be converted into a rule with two body atoms, namely $r^{\text{new}}(x, y) \leftarrow p(x, z) \wedge I(z, y)$. We also allow to use both relations and inverse relations as predicates in inference rules. Throughout this paper, a triple (h_i, r_i, t_i) in $\mathcal{G}_{\text{rel}}$ and a binary atom $r_i(h_i, t_i)$, as well as a triple (e_i, Type, c_i) in $\mathcal{G}_{\text{type}}$ and a unary atom $c_i(e_i)$, are used interchangeably.

A *substitution* σ is a function that maps a set T of variables to a set T' of variables or constants. It is called ground if it maps all variables to constants. By $t\sigma = t'$ we denote that $t \in T$ is mapped to $t' \in T'$ by σ. Given an atom $A(t_1, \ldots, t_n)$, we have $A(t_1, \ldots, t_n)\sigma = A(t_1\sigma, \ldots, t_n\sigma)$, and this naturally extends to a set of atoms. Given a knowledge graph $\mathcal{G}$, an r-specific inference rule R and a new fact $r^{\text{new}}(a, b)$, let $\mathcal{K} = \mathcal{G}_{\text{rel}} \cup \mathcal{G}^-_{\text{rel}} \cup \mathcal{G}_{\text{type}} \cup \{I(e, e) \mid e \in \mathcal{E}\}$, we say $\mathcal{K} \models_R r^{\text{new}}(a, b)$ if there exists a ground substitution σ such that $H_R\sigma = r^{\text{new}}(a, b)$ and $B_R\sigma \subseteq \mathcal{K}$. Given a set Σ of r-specific

inference rules, we say $\mathcal{K} \vdash_\Sigma r^{\text{new}}(a,b)$ if there exists an r-specific inference rule $R \in \Sigma$ such that $\mathcal{K} \models_R r^{\text{new}}(a,b)$. We say $r^{\text{new}}(a,b)$ is *plausible* in $\mathcal{G}$ if there exists a set of possibly correct r-specific inference rules Σ such that $\mathcal{K} \vdash_\Sigma r^{\text{new}}(a,b)$.

Chain-like Rule. An r-specific inference rule is said to be *chain-like* if every body atom shares one variable with the previous body atom and the other variable with the next body atom. Formally, an r-specific chain-like rule (CR) is of the form:

$$r^{\text{new}}(x,y) \leftarrow p_1(x,z_1) \wedge p_2(z_1,z_2) \wedge \cdots \wedge p_L(z_{L-1},y)$$

where $p_1,\ldots,p_L$ are relations in $\mathcal{R} \cup \mathcal{R}^- \cup \{I\}$. Throughout this paper, the atoms $p_1(x,z_1), p_2(z_1,z_2), \cdots, p_L(z_{L-1},y)$ are called *chained atoms*. A CR is called an L-CR if it has L chained atoms.

Typed Rule. A typed rule [37] extends an r-specific L-CR with unary atoms. Let C be the set of types. An r-specific L-typed rule R is of the form:

$$r^{\text{new}}(x,y) \leftarrow c_1(x) \wedge c_2(z_1) \wedge p_1(x,z_1) \wedge \cdots \wedge p_L(z_{L-1},y) \wedge c_{L+1}(y)$$

where $c_1, c_2, \ldots, c_{L+1}$ are types in C.

Link Prediction. Given a knowledge graph $\mathcal{G}$, a head query $(?, r^{\text{new}}, t)$ or a tail query $(h, r^{\text{new}}, ?)$, link prediction aims to find all entities $e \in \mathcal{E}$ such that (e, r^{new}, t) for $(?, r^{\text{new}}, t)$ or (h, r^{new}, e) for $(h, r^{\text{new}}, ?)$ is plausible in $\mathcal{G}$.

Triple Classification. Given a knowledge graph $\mathcal{G}$ and a triple (h, r^{new}, t) where $h \in \mathcal{E}$, $t \in \mathcal{E}$ and $r \in \mathcal{R}$, triple classification aims to estimate whether (h, r^{new}, t) is plausible in $\mathcal{G}$.

3 Related Work

Logical Rule Learning. Learning logical rules was previously studied in the field of Inductive Logic Programming (ILP) [19, 23], where logical rules are learnt by a generate-and-test manner. Generate-and-test is a two-step pipeline that first generates logical rules from relational paths in a KG, and then filters rules with high confidence scores based on some tests. Classical ILP methods such as FOIL [20] and QuickFOIL [42] cannot be directly applied to learning from KGs due to the lack of negative examples. Modern ILP methods like AMIE+ [10] and AnyBURL [16] treat triples outside a KG as negative examples and exploit various search heuristics to efficiently learn rules, thus they are able to learn rules from a large KG. In addition, statistical relational learning (SRL) [14, 21, 24] is a kind of probabilistic ILP method. It aims to learn a scalar weight for each rule by using various machine learning algorithms.

More recently, there has been an emerging interest in exploiting end-to-end neural-based methods for rule learning. Typical end-to-end methods include NeuralLP [40], DRUM [25], and NLIL [41]. They usually employ Tensorlog [7] operators to learn logical rules in an end-to-end fashion. In addition to applying Tensorlog, there are several methods that extend the generate-and-test manner to learn logical rules by neural models, including RNNLogic [22], RLogic [6] and NCRL [5]. Compared with traditional ILP methods, neural-based methods excel at learning rules from imperfect data. Our proposed approach follows this research line but aims to learn more complex rules, namely TC-rules. Most existing neural-based methods merely consider chain-like rules. It is worth noting that TyRuLe [37] is a new approach that learns typed rules by exploiting both path-based and embedding-based strategies. However, TyRuLe is designed to handle explicit type constraints only. Different from TyRuLe, our approach can learn both explicit and implicit type constraints. In particular, even when there is no explicit type constraint on entities, our approach is still able to automatically discover implicit type constraints. Besides, our approach can discover the same set of logical rules for answering both head queries and tail queries.

Embedding-based Methods. Our work is also related to knowledge graph emebedding (KGE) [4, 30, 34, 36, 39]. It aims to represent entities and relations in KGs as low-dimensional real-value vectors. Existing KGE methods usually estimate the truth degree of a triple based on the semantic distance or similarity calculated from entity and relation embeddings. Although KGE methods have been shown to work well in large-scale KGC, they fail to measure the triples involving previously unseen entities as their embeddings have not been trained. Besides, the learnt embeddings are real-value vectors that can hardly be interpreted.

Recently, graph neural networks (GNNs) such as R-GCN [26], GraIL [32] and MGNN [8] are proposed to address KGC. They can handle the inductive setting where missing triples involve unseen entities. However, most GNN-based methods are black-box models that are difficult to interpret. Although there exist a few methods such as MGNN that can interpret logical rules as explanations, they rely on some restrictions on the test data and are hard to generalize in link prediction. For example, MGNN restricts that every test head-tail pair should appear in the background KG. Thus, we do not follow the KGE and GNN approaches in this work.

4 Methodology

To express both the explicit and implicit type constraints on entity variables, we extend chain-like rules by adding a disjunction of unary atoms to express explicit type constraints as well as a disjunction of binary atoms to express implicit type constraints for every entity variable in the rule. We call such extended rules *Type-constraint enhanced Chain-like rules* or *TC-rules* for short. They are formally defined below.

Definition 1. *An r-specific L-TC-rule (simply a TC-rule if r and L are clear from the context) R is of the form: $r^{\text{new}}(x,y) \leftarrow p_1(x,z_1) \wedge p_2(z_1,z_2) \wedge \cdots \wedge p_L(z_{L-1},y) \wedge C_1(x) \wedge C_2(z_1) \wedge \ldots \wedge C_L(z_{L-1}) \wedge C_{L+1}(y)$, where $p_1,\ldots,p_L$ are relations in $\mathcal{R} \cup \mathcal{R}^- \cup \{I\}$, $C_l(u) \in \{E_l(u), H_l(u), E_l(u) \vee H_l(u)\}$, $E_l(u) = \bigvee_{i=1}^{m_l} g_{l,i}(u)$ with $g_{l,i}$ being different predicates in C and $0 \le m_l \le |C|$ is called an* explicit type constraint *on u, and $H_l(u) = \bigvee_{i=1}^{n_l} q_{l,i}(u, v_{l,i})$ with $v_{l,i}$ being new variables, $q_{l,i}$ being different predicates in $\mathcal{R} \cup \mathcal{R}^-$ and $0 \le n_l \le |\mathcal{R} \cup \mathcal{R}^-|$ is called an* implicit type constraint *on u. Some entity variables v can have no type constraint; in this case $C_l(v)$ is empty, i.e., $m_l = 0$ and $n_l = 0$.*

The following Example 1 reveals that implicit type constraints can help to deal with the scenarios where the type information is missing or incomplete in the given KG.

Example 1. *Suppose there is no explicit type information in the KG. The following TC-rule R : $\text{maleLeadOf}^{\text{new}}(x,y) \leftarrow \text{actCharacter}(x,z) \wedge \text{characterOf}(z,y) \wedge (\text{brotherOf}(x,u) \vee \text{uncleOf}(x,v)) \wedge (\text{leadInStory}(z,w))$ restricts that x should be male and z should be a lead character by declaring that a certain entity u or v exists for supporting $\text{brotherOf}(x,u) \vee \text{uncleOf}(x,v)$ and that a certain entity w exists for supporting $\text{leadInStory}(z,w)$.*

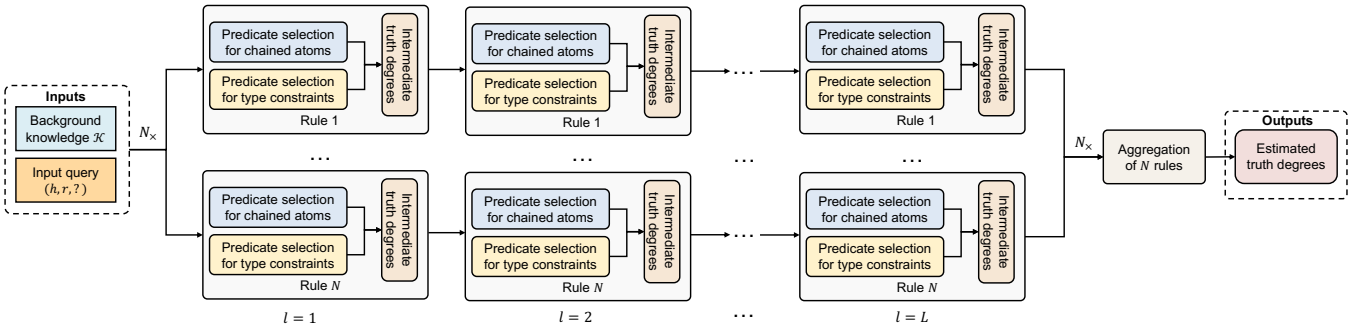


Figure 2: Illustration of the proposed Type Constraint Learning Model (TCLM).

Our intuition for incorporating disjunctions into TC-rules stems from the scenario depicted in Example 1, wherein a disjunction of binary atoms ($\text{brotherOf}(x,u) \vee \text{uncleOf}(x,v)$) suffices to articulate a type constraint $\text{Male}(x)$. Besides, we argue that the introduction of disjunctions makes TC-rules more compact. The reason is as follows. A TC-rule with implicit type constraints can be decomposed into several chain-like rules. For example, to express R in Example 1, we require two chain-like rules: R'_1 : $\text{maleLeadOf}^{\text{new}}(x,y) \leftarrow \text{brotherOf}(x,u) \wedge \text{brotherOf}^-(u,v) \wedge \text{actCharacter}(v,t) \wedge \text{leadInStory}(t,w) \wedge \text{leadInStory}^-(w,z) \wedge \text{characterOf}(z,y)$ and R'_2 : $\text{maleLeadOf}^{\text{new}}(x,y) \leftarrow \text{uncleOf}(x,u) \wedge \text{uncleOf}^-(u,v) \wedge \text{actCharacter}(v,t) \wedge \text{leadInStory}(t,w) \wedge \text{leadInStory}^-(w,z) \wedge \text{characterOf}(z,y)$. Every chain-like rule should introduce successive atomic and inverse relations such as brotherOf and brotherOf^- to express an implicit type constraint, making the rule much longer. To generalize this case, suppose a TC-rule contains L chained atoms and m implicit type constraints in the body, and every implicit type constraint uniformly consists of k disjunctive atoms. To achieve the same semantics, this rule needs to be decomposed into k^m chain-like rules each of which has $L + 2m$ atoms in the body. It is clear that TC-rules are more compact than chain-like rules. In practice, it is harder to learn longer chain-like rules due to the exponentially larger search space.

4.1 Formalization of TCLM

To learn TC-rules in an end-to-end manner, we propose a neural model named *Type Constraint Learning Model* or *TCLM* for short. The intuition of TCLM is to transform the rule learning problem in discrete space into the parameter learning problem in continuous space. By and large, we parameterize TCLM to simulate the inference of TC-rules and then optimize the parameters of TCLM to distinguish true triples from false triples. An overview of TCLM is illustrated in Figure 2.

Let $\mathcal{G}$ be a given knowledge graph, N the maximum number of rules to be learnt, L the maximum number of chained atoms, $\mathcal{K} = \mathcal{G}_{\text{rel}} \cup \mathcal{G}^-_{\text{rel}} \cup \{(e, I, e) \mid e \in \mathcal{E}\}$ the background knowledge, $C = \{c_1, \cdots, c_m\}$ the set of explicit types, and $n = |\mathcal{R}|$. Suppose $\mathcal{R} = \{r_i\}_{1 \le i \le n}$, its corresponding set of inverse relations $\mathcal{R}^- = \{r_i\}_{n+1 \le i \le 2n}$, and $I = r_{2n+1}$. The goal of TCLM is to estimate a truth degree $\xi^{N,L}_{r,x,y}$ for the triple $(x, r, y) \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, where the estimated truth degree $\xi^{N,L}_{r,x,y}$ reflects the degree of whether the triple (x, r, y) can be inferred by a certain rule among N L-TC-rules. For $1 \le k \le N, 1 \le l \le L$, the intermediate estimated truth degree $s^{(k,l)}_{r,x,y}$ for the l^{th} atom in the k^{th} rule is defined as below.

For $l = 1$, the truth degree is calculated by:

$$s^{(k,1)}_{r,x,y} = \phi^{(k,1)}_r(x)\phi^{(k,l+1)}_r(y) \sum_{i=1}^{2n+1} w^{(r,k,1)}_i \mathbb{I}((x, r_i, y) \in \mathcal{K}) \tag{1}$$

For $2 \le l \le L$, the truth degree is calculated by:

$$s^{(k,l)}_{r,x,y} = \phi^{(k,l+1)}_r(y) \sum_{i=1}^{2n+1} w^{(r,k,l)}_i \sum_{z:(z,r_i,y)\in\mathcal{K}} s^{(k,l-1)}_{r,x,z} \tag{2}$$

where $w^{(r,k,l)} \in [0,1]^{2n+1}$ denotes the trainable relational selection weights for the l^{th} atoms in the k^{th} rule for the head relation r. $\mathbb{I}(\psi)$ is an indicator function that returns 1 if ψ is true or 0 otherwise. $w^{(r,k,l)}$ is confined to $[0,1]^{2n+1}$ by a softmax layer. Intuitively, the use of the softmax function enables $w^{(r,k,l)}$ to approximate the one-hot distribution, thereby simulating the predicate selection process. $\phi^{(k,l)}_r(u)$ is a scoring function for the type constraints on u. Formally, $\phi^{(k,l)}_r(u)$ is defined as:

$$\begin{aligned}\phi^{(k,l)}_r(u) = \sigma_{01}(&\alpha^{(r,k,l)} \sum_{i=1}^{m} h^{(r,k,l)}_i \mathbb{I}((u, \text{Type}, c_i) \in \mathcal{G}_{\text{type}}) \\ &+ \beta^{(r,k,l)} \sum_{i=1}^{2n} h^{(r,k,l)}_{i+m} \mathbb{I}(\exists z : (u, r_i, z) \in \mathcal{G}_{\text{rel}} \cup \mathcal{G}^-_{\text{rel}}) \\ &+ (1 - \sigma_{01}(\alpha^{(r,k,l)} + \beta^{(r,k,l)}))\end{aligned} \tag{3}$$

where $\sigma_{01}(x) = \max(\min(x, 1), 0)$, $h^{(r,k,l)} \in [0,1]^{m+2n}$ denotes the trainable type selection weights of the l^{th} type constraint in the l^{th} rule for relation r. $h^{(r,k,l)}$ is confined to $[0,1]$ by σ_{01}. We use $\alpha^{(r,k,l)}$ (resp. $\beta^{(r,k,l)}$) to control whether the entity has an explicit (resp. implicit) type constraint. For example, $\alpha^{(r,k,l)} = 1$ (resp. $\beta^{(r,k,l)} = 1$) implies that there is an explicit (resp. implicit) type constraint for the given entity. Note that $\alpha^{(r,k,l)} = 0$ and $\beta^{(r,k,l)} = 0$ imply that there is no type constraint for the given entity.

Intuitively, Equation (1-3) simulates the inference of TC-rules, where the part on the right side of $\phi^{(k,l)}_r(u)$ in Equation (1-2) simulates the inference of chain-like rules, while $\phi^{(k,l)}_r(u)$ captures the type constraints of entities.

Then the ultimate estimated truth degree is calculated by weight-summing the estimated truth degrees for N rules:

$$\xi^{N,L}_{r,x,y} = \sum_{k=1}^{N} \mu_k^{(r)} s^{(k,L)}_{r,x,y} \tag{4}$$

where $\mu_l^{(r)} \in [-1, 1]$ is a trainable weight that represents the weight of the k^{th} rule. $\mu_l^{(r)}$ is confined to $[-1, 1]$ by a tanh layer. Intuitively, the use of the tanh function enables TCLM to learn a positive or negative weight for each rule. By assigning different weights to each rule, TCLM can learn different numbers of rules for different head relations, as the rules with weights close to 0 can be omitted. The model is trained by minimizing the following objective function

$$\mathcal{L} = -\sum_{(x,r,y)\in\mathcal{G}} \log \frac{\exp(\xi^{N,L}_{r,x,y})}{\exp(\xi^{N,L}_{r,x,y}) + \sum_{e\in\mathcal{E},(x,r,e)\notin\mathcal{G}} \exp(\xi^{N,L}_{r,x,e})} \tag{5}$$

The intuition of the above objective is to distinguish a true triple $(x, r, y) \in \mathcal{G}$ from its corrupted, probably false triples $(x, r, e) \notin \mathcal{G}$. By introducing the following notion of induced parameter assignment, we show in Theorem 1 that the formalization of TCLM is faithful to a certain set of TC-rules.

Definition 2. *Given a set of N r-specific L-TC-rules $\Sigma = \{R_k\}_{1\le k\le N}$, where R_k is of the form $r^{\text{new}}(x, y) \leftarrow p_{k,1}(x, z_1) \wedge ... \wedge p_{k,L}(z_{L-1}, y) \wedge C_{k,1}(x) \wedge ... \wedge C_{k,L+1}(y)$, where $p_{k,l} \in \mathcal{R} \cup \mathcal{R}^- \cup \{I\}$, $C_{k,l}(u) \in \{E_{k,l}(u), H_{k,l}(u), E_{k,l}(u) \vee H_{k,l}(u)\}$, $E_{k,l}(u) = \bigvee_{i=1}^{m_{k,l}} g_{k,l,i}(u)$ with $g_{k,l,i}$ being different predicates in C and $0 \le m_{k,l} \le |C|$, $H_{k,l}(u) = \bigvee_{i=1}^{n_{k,l}} q_{k,l,i}(u, v_{k,l,i})$ with $v_{k,l,i}$ being new variables, $q_{k,l,i}$ being different predicates in $\mathcal{R} \cup \mathcal{R}^-$ and $0 \le n_{k,l} \le |\mathcal{R} \cup \mathcal{R}^-|$, we call a parameter assignment of TCLM $\theta_r^{N,L} = \{w_i^{(r,k,l)}\}_{1\le k\le N,1\le l\le L,1\le i\le 2n+1} \cup \{h_i^{(r,k,l)}\}_{1\le k\le N,1\le l\le L+1,1\le i\le 2n+m} \cup \{\alpha^{(r,k,l)}, \beta^{(r,k,l)}\}_{1\le k\le N,1\le l\le L+1} \cup \{\mu_k^{(r)}\}_{1\le l\le L}$ Σ-induced if it satisfies the following conditions for all $1 \le k \le N, 1 \le l \le L$:*

(1) $\forall 1 \le i \le 2n+1 : w_i^{(r,k,l)} = 1$ if $p_{k,l} = r_i$, otherwise $w_i^{(r,k,l)} = 0$.

(2) $\forall 1 \le i \le m : h_i^{(r,k,l)} = 1$ if $g_{k,l,j} = c_i$ for some $j \in \{1, \cdots, m_{k,l}\}$, otherwise $h_i^{(r,k,l)} = 0$.

(3) $\forall 1 \le i \le 2n : h_{i+m}^{(r,k,l)} = 1$ if $q_{k,l,j} = r_i$ for some $j \in \{1, \cdots, n_{k,l}\}$, otherwise $h_{i+m}^{(r,k,l)} = 0$.

(4) $\alpha^{(r,k,l)} = 1$ if there is some $g_{k,l,j}$ appearing in $C_{k,l}$, otherwise $\alpha^{(r,k,l)} = 0$.

(5) $\beta^{(r,k,l)} = 1$ if there is some $q_{k,l,j}$ appearing in $C_{k,l}$, otherwise $\beta^{(r,k,l)} = 0$.

(6) $u_r^{(k)} = 1$.

Theorem 1. *Let $\mathcal{G}$ be a knowledge graph, $\mathcal{K} = \mathcal{G}_{\text{rel}} \cup \mathcal{G}^-_{\text{rel}} \cup \mathcal{G}_{\text{type}} \cup \{I(e, e) \mid e \in \mathcal{E}\}$, $\Sigma = \{R_k\}_{1\le k\le N}$ a set of r-specific L-TC-rules and $\theta_r^{N,L}$ the Σ-induced parameter assignment of TCLM. Given an arbitrary triple (a, r^{new}, b), $\xi^{N,L}_{r,a,b} \ge 1$ if and only if $\mathcal{K} \vdash_\Sigma r^{\text{new}}(a, b)$.*

Theorem 1 enables us to extract explainable TC-rules from the learnt parameter assignment of TCLM. The rule extraction algorithm is shown in Algorithm 1. Intuitively, Algorithm 1 interprets TC-rules from the parameter assignment of TCLM using beam search. In this algorithm, f_l denotes the set of (R', ϵ)-pairs for the l^{th} atom, where R' is the currently interpreted (partial) rule and ϵ its estimated score. g_l (resp. d_l) denotes the currently interpreted (partial) explicit (resp. implicit) type constraints for the l^{th} variable. It should be noted that the process for interpreting r-specific TC-rules outputs up to b interpreted rules that share the same confidence score $\mu_r^{(k)}$ for the k^{th} target rule.

Algorithm 1: Interpreting r-specific L-TC-rules

Input: beam size $b \ge 1$, the threshold $\delta \in (0.0, 0.5]$ for determining whether to introduce type constraints, and a parameter assignment of TCLM for the relation r, namely $\theta_r^{N,L} = \{w_i^{(r,k,l)}\}_{1\le k\le N,1\le l\le L,1\le i\le 2n+1} \cup \{h_i^{(r,k,l)}\}_{1\le k\le N,1\le t\le L+1,1\le i\le 2n+m} \cup \{\alpha^{(r,k,l)}, \beta^{(r,k,l)}\}_{1\le k\le N,1\le l\le L+1}$
Output: up to bN r-specific L-TC-rules
$\mathbb{R} \leftarrow \emptyset$;
for $1 \le k \le N$ **do**
  $f_0 \leftarrow \{(\Delta^L, 1)\}$, $g_1 \leftarrow \emptyset$, $d_1 \leftarrow \emptyset$, where Δ denotes a placeholder to be filled;
  $\forall 1 \le l \le L : f_l \leftarrow \emptyset, g_{l+1} \leftarrow \emptyset, d_{l+1} \leftarrow \emptyset$;
  for $1 \le t \le L$ **do**
    for $(R, \epsilon) \in f_{t-1}$ **do**
      for $1 \le i \le 2n+1$ **do**
        $R' \leftarrow R$ with the l^{th} placeholder replaced with r_i;
        $f_t \leftarrow f_t \cup \{(R', \epsilon w_i^{(r,k,l)})\}$;
    sort $f_l = \{(R, \epsilon)_j\}_{1\le j\le b(2n+1)}$ in the descending order of ϵ and preserve the top-b in f_l;
  for $1 \le l \le L+1$ **do**
    if $\alpha^{(r,k,l)} \ge \delta$ **then**
      for $1 \le i \le m$ **do**
        if $h_i^{(r,k,l)} \ge \delta$ **then**
          $g_l \leftarrow g_l \cup \{c_i\}$;
    if $\beta^{(r,k,l)} \ge \delta$ **then**
      for $1 \le i \le 2n$ **do**
        if $h_{m+i}^{(r,k,l)} \ge \delta$ **then**
          $d_l \leftarrow d_l \cup \{r_i\}$;
  $\mathbb{K} \leftarrow \{R'$ rewritten from R and $\{g_l, d_l\}_{1\le l\le L+1}$ to the form of a TC-rule $\mid (R, \epsilon) \in f_L\}$;
  $\mathbb{R} \leftarrow \mathbb{R} \cup \mathbb{K}$;
return $\mathbb{R}$;

4.2 Bi-directional Learning

To explain why a given triple (h, r^{new}, t) is plausible in $\mathcal{G}$, we should avoid confusing explanations, i.e., the explanations for answering both $(?, r^{\text{new}}, t)$ and $(h, r^{\text{new}}, ?)$ should be the same. Therefore, we propose a bi-directional learning mechanism that enforces the model to yield the same set of logical rules by learning shared parameters for answering both $(?, r^{\text{new}}, t)$ and $(h, r^{\text{new}}, ?)$. Formally,

Table 1: Statistical information for experimental datasets.

Dataset	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{G}_{\text{train}}\rvert$	$\lvert\mathcal{G}_{\text{valid}}\rvert$	$\lvert\mathcal{G}_{\text{test}}\rvert$	$\lvert C\rvert$	$\lvert\mathcal{G}_{\text{type}}\rvert$
AirGraph	64,782	40	324,585	-	36,065	24	64,771
YAGO26K906	26,078	34	351,664	-	39,074	106	9,677
Family	3,007	12	23,483	2,038	2,835	-	-
Kinship	104	25	3,206	2,137	5,343	-	-
UMLS	135	46	1,959	1,306	3,264	-	-
WN18RR	40,943	11	86,835	3,034	3,134	-	-
FB15k-237	14,541	237	272,115	17,535	20,466	-	-
YAGO3-10	123,182	37	1,079,040	5,000	5,000	-	-

the estimation of the truth degree of (h, r^{new}, t) from the angle of answering head queries is defined below.

For $l = 1$, the truth degree is calculated by:

$$\bar{s}^{(k,1)}_{r^-,y,x} = \phi^{(k,L+1)}_r(y)\phi^{(k,L)}_r(x)\sum_{i=1}^{2n+1} w^{(r,k,L)}_i \mathbb{I}((x, r_i, y) \in \mathcal{K}) \quad (6)$$

For $2 \le l \le L$, the truth degree is calculated by:

$$\bar{s}^{(k,l)}_{r^-,y,x} = \phi^{(k,L-l+1)}_r(x)\sum_{i=1}^{2n+1} w^{(r,k,L-l+1)}_i \sum_{z:(x,r_i,z)\in\mathcal{K}} \bar{s}^{(k,l-1)}_{r^-,y,z} \quad (7)$$

Then the ultimate estimated degree is formally defined as:

$$\bar{\xi}^{N,L}_{r^-,y,x} = \sum_{k=1}^{N} \mu^{(r)}_l \bar{s}^{(k,L)}_{r^-,y,x} \quad (8)$$

where all the trainable parameters are shared with $\xi^{N,L}_{r,x,y}$.

The following theorem shows the consistency of estimated truth degrees for answering both $(?, r^{\text{new}}, t)$ and $(h, r^{\text{new}}, ?)$.

Theorem 2. *Let $\mathcal{G}$ be a knowledge graph. For any triple $(a, r, b) \in \mathcal{E}\times\mathcal{R}\times\mathcal{E}$, $\forall N \ge 1, L \ge 1 : \xi^{N,L}_{r,a,b} = \bar{\xi}^{N,L}_{r^-,b,a}$.*

Theorem 2 implies that TCLM will always compute the same truth degree for supporting the head h in answering $(?, r^{\text{new}}, t)$ and for supporting the tail t in answering $(h, r^{\text{new}}, ?)$, thus we can introduce the same threshold for truth degrees to determine whether (h, r^{new}, t) is plausible in $\mathcal{G}$ from either the angle of answering $(?, r^{\text{new}}, t)$ or the angle of answering $(h, r^{\text{new}}, ?)$.

5 Evaluation

5.1 Experimental Setups

Datasets. To compare TCLM with TyRuLe [37] that learns typed rules for KGC, we conducted experiments on two datasets with explicit types on entities, including YAGO26K906 [37] and AirGraph [37]. To compare TCLM with other SOTA KGC methods, we used six well-known benchmark datasets for empirical evaluation, including Family [40], Kinship [13], UMLS [13], WN18RR [9], FB15k-237 [33] and YAGO3-10 [29]. Statistical details for these datasets are reported in Table 1.

Evaluation Metrics. For each test triple (h, r^{new}, t) in evaluation, we built two queries $(h, r^{\text{new}}, ?)$ and $(?, r^{\text{new}}, t)$. We computed the truth degrees for corrupted tail triples and then computed the rank of the correct answer. Based on the rank, we reported the Mean Reciprocal Rank (MRR for short) and Hit@k (H@k for short) metrics under the filtered setting introduced by [4]. Following [22], the rank of the correct answer is defined by $i + (j+1)/2$ in our proposed setting, where i is the number of corrupted triples with higher truth degrees than the correct answer and j the number of corrupted triples with the same truth degree as the correct answer. It is worth noting that in the preliminary setting for ranking, the rank of a correct answer is set to $i + 1$. As pointed out by [22, 31], this setting for ranking is problematic because in some algorithms the correct answer can be assigned the same truth degrees as other entities. For example, if an algorithm predicts all truth degrees to 0, then the rank for any answer will be computed as 1, leading to a wrong assessment of the algorithm. Therefore, we used $i + (j+1)/2$ to calculate the ranks for a more reasonable assessment. For triple classification, we used precision, recall and the F1 score as evaluation metrics.

Table 2: Comparison results on YAGO26K-906 and AirGraph under the preliminary setting (denoted by *) for ranking and the more reasonable setting (denoted by †) for ranking [22], where the underlined numbers denote our reproduced results. For each metric and dataset, the best results are highlighted.

Method	YAGO26K-906				AirGraph			
	MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10
JOIE-TransE*	0.306	18.6	-	51.7	-	-	-	-
JOIE-DistMult*	0.296	19.4	-	45.5	-	-	-	-
JOIE-HolE*	0.327	22.4	-	52.4	-	-	-	-
AMIE+*	<u>0.155</u>	<u>13.9</u>	<u>16.8</u>	<u>18.0</u>	0.123	11.9	12.7	12.9
AnyBURL*	-	-	-	-	0.313	26.2	34.3	43.1
TyRuLe*	0.428	37.7	-	52.4	0.353	30.6	37.5	45.1
TCLM*	**0.644**	**56.9**	**69.3**	**77.7**	**0.464**	**41.7**	**47.8**	**55.3**
TCLM†	0.643	56.8	69.3	77.6	0.441	39.5	45.3	52.8
TCLM (w/o types)†	0.593	50.1	65.2	75.5	0.364	29.8	40.6	48.2
TCLM (EC only)†	0.507	40.5	56.7	69.4	0.405	35.8	42.0	49.2
TCLM (IC only)†	0.625	54.5	67.6	76.4	0.430	38.0	44.4	53.2

Implementation Details. We implemented TCLM by Pytorch 2.0.0 on an NVIDIA A100 GPU with 40GB RAM.[1] The model was trained by Adam [12] with 100 training epochs for YAGO26K-906, AirGraph and WN18RR, and 50 for the other datasets. An early stopping strategy was applied to maximize the MRR score on the validation set. The initial learning rate was set to 1e-1 and the mini-batch size was set to 32 for Family, Kinship, UMLS and WN18RR, and 4 for the other datasets. We applied dropout [28] to TCLM by setting the dropout rate to 0.3 for Family, Kinship, UMLS, and 0.1 for the other datasets. The maximum length of rules L was set to 4 for WN18RR and 3 for other datasets. The maximum number of rules to be learnt N was set to 100 for WN18RR, 70 for Family, Kinship, UMLS and FB15k237, and 50 for others. All these hyper-parameters were set to maximize the MRR score on the validation set.

Baselines. We compared TCLM with baseline methods mainly in two categories. For the embedding-based category, we compared TCLM with KGE methods TransE [4], DistMult [39], ComplEx [34], ConvE [9], R-GCN [26], TuckER [2], RotatE [30] and JOIE [11]. For the rule-based category, we compared TCLM with SOTA neural-based methods NeuralLP [40], DRUM [25], NLIL [41], CTP [17], RNNLogic [22], RLogic [6] and NCRL [5]. For this category, we

[1] Code and data are available at: https://github.com/qikunxun/TCLM

Table 3: Comparison results on six benchmark datasets without explicit type information, where the results marked by † are estimated under the more reasonable setting for ranking [22], and the underlined numbers denote our reproduced results under such a setting. For each metric and dataset, the best results under the setting [22] are highlighted in bold.

Method	Family				Kinship				UMLS				WN18RR				FB15k-237				YAGO3-10			
	MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10
TransE†	0.450	22.1	-	87.4	0.310	0.9	-	84.1	0.690	52.3	-	89.7	0.230	2.2	-	52.4	0.294	18.9	-	46.5	0.360	25.1	-	58.0
DistMult†	0.540	36.0	-	88.5	0.354	18.9	40.0	75.5	0.391	25.6	44.5	66.9	0.430	39.0	44.0	49.0	0.241	15.5	26.3	41.9	0.340	24.3	-	53.3
ComplEx†	0.810	72.7	-	94.6	0.418	24.2	49.9	81.2	0.411	27.3	46.8	70.0	0.440	41.0	46.0	51.0	0.247	15.8	27.5	42.8	0.340	24.8	-	54.9
ConvE†	-	-	-	-	-	-	-	-	-	-	-	-	0.430	40.0	44.0	52.0	0.320	21.6	-	50.1	0.440	35.5	-	61.6
R-GCN†	-	-	-	-	-	-	-	-	-	-	-	-	0.402	34.5	43.7	49.4	0.273	18.2	30.3	45.6	-	-	-	-
TuckER†	-	-	-	-	0.630	46.2	69.8	86.3	0.732	62.5	81.2	90.9	0.470	44.3	48.2	52.6	**0.358**	**26.6**	**39.4**	**54.4**	-	-	-	-
RotatE†	0.860	78.7	-	93.3	0.651	50.4	75.5	93.2	0.744	63.6	82.2	**93.9**	0.476	42.8	49.2	**57.1**	0.338	24.1	37.5	53.3	0.490	40.2	-	**67.0**
AMIE+†	0.541	48.7	59.2	59.6	0.416	23.7	49.3	83.0	0.417	25.3	51.9	68.5	0.192	19.0	19.5	19.6	0.118	8.8	13.1	17.3	0.259	23.1	28.4	30.3
MLN†	-	-	-	-	0.351	18.9	40.8	70.7	0.688	58.7	75.5	86.9	-	-	-	-	-	-	-	-	-	-	-	-
PathRank†	-	-	-	-	0.369	27.2	41.6	67.3	0.197	14.8	21.4	25.2	0.189	17.1	20.0	22.5	0.087	7.4	9.2	11.2	-	-	-	-
NeuralLP†	0.880	80.1	-	98.5	0.302	16.7	33.9	59.6	0.483	33.2	56.3	77.5	0.381	36.8	38.6	40.8	0.237	17.3	25.9	36.1	-	-	-	-
DRUM†	0.890	82.6	-	**99.2**	0.334	18.3	37.8	67.5	0.548	35.8	69.9	81.4	0.382	36.9	38.8	41.0	0.238	17.4	26.1	36.4	-	-	-	-
NLIL†	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.250	-	-	32.4	-	-	-	-
CTP†	-	-	-	-	0.335	17.7	37.6	70.3	0.404	28.8	43.0	67.4	-	-	-	-	-	-	-	-	-	-	-	-
RLogic†	0.880	81.3	-	97.2	0.580	43.4	-	87.2	0.710	56.6	-	93.2	0.470	44.3	-	53.7	0.310	20.3	-	50.1	0.360	25.2	-	50.4
NCRL*	0.910	85.2	-	99.3	0.640	49.0	-	92.9	0.790	65.9	-	95.1	0.670	56.3	-	85.0	0.300	20.9	-	47.3	0.380	27.4	-	53.6
NCRL†	0.806	74.0	86.0	90.1	0.537	38.6	62.6	83.4	0.562	45.9	61.6	71.5	0.263	23.2	28.0	31.3	0.141	7.5	17.0	26.6	0.012	0.3	1.3	2.8
RNNLogic†	0.860	79.2	-	95.7	0.639	49.5	73.1	92.4	0.745	63.0	83.3	92.4	0.455	41.4	47.5	53.1	0.288	20.8	31.5	44.5	0.379	30.2	42.1	53.3
TCLM†	**0.985**	**98.1**	**98.9**	99.1	**0.686**	**54.3**	**79.5**	**95.3**	**0.808**	**73.0**	**87.1**	92.8	**0.483**	**44.7**	**49.7**	55.2	0.311	23.0	34.0	47.2	**0.505**	**43.4**	**54.7**	63.5
TCLM (w/o IC)†	0.984	97.9	**98.9**	99.0	0.684	54.1	79.1	95.1	0.788	70.0	86.2	92.8	0.462	42.7	47.6	53.6	0.309	22.8	33.8	46.7	0.430	34.3	47.4	59.1

Table 4: Comparison results for triple classification on UMLS and Kinship.

Method	UMLS						Kinship					
	Triple classification (head)			Triple classification (tail)			Triple classification (head)			Triple classification (tail)		
	P(%)	R(%)	F1(%)	P(%)	R(%)	F1(%)	P(%)	R(%)	F1(%)	P(%)	R(%)	F1(%)
NeuralLP	58.4	67.0	62.4	59.9	67.2	63.3	47.6	63.7	54.5	49.1	70.0	57.7
DRUM	60.1	77.9	67.9	62.8	77.4	69.3	55.1	68.4	61.0	55.8	65.8	60.4
RNNLogic	87.8	90.3	89.0	88.3	**92.7**	90.4	89.4	93.1	91.2	87.5	**95.3**	91.3
TCLM	**94.5**	90.0	**92.2**	**94.5**	90.0	**92.2**	**92.0**	**94.8**	**93.4**	**92.0**	94.8	**93.4**
TCLM (w/o bi-directional learning)	88.1	81.2	84.5	89.2	84.9	87.0	87.0	91.1	89.0	89.5	93.1	91.7

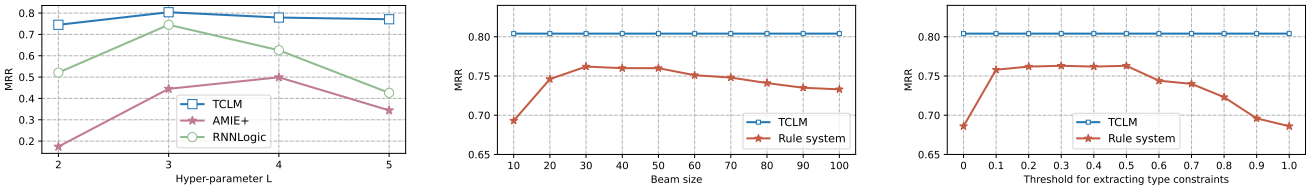
also compared TCLM with three SOTA search-based ILP methods TyRuLe [37], AnyBURL [16], and AMIE+ [10]. Besides, we compared TCLM with two statistical relational learning methods Markov logic network (MLN) [24] and PathRanking [14].

5.2 Main Empirical Results

As mentioned before, some methods apply a preliminary setting to calculate the ranks as $i+1$, which may lead to a wrong assessment of methods. For all compared methods, we have tried to reproduce their results by applying the more reasonable setting [22] to calculate the ranks as $i+(j+1)/2$, but we failed for some methods due to their incomplete source code. To ensure a fair comparison, we reported the results under both settings if needed.

Table 2 reports the comparison results for link prediction on two datasets with explicit types on entities, where the results of baselines are sourced from [37] if not otherwise specified. Results show that both TCLM* and TCLM† significantly outperform TyRuLe*. In particular, TCLM* outperforms TyRuLe* by absolute gains of 19.2% and 11.1% in the Hit@1 scores on YAGO26K-906 and AirGraph, respectively. We further conducted ablation experiments on three variant models of TCLM. The first variant model, denoted by TCLM (w/o types), omits the $\phi_r^{(k,l)}(.)$ terms in Equations (1-2) and Equations (6-7). The second variant model, denoted by TCLM (EC only), omits the learning of implicit type constraints in Equation (3) by fixing $\beta^{(r,k,l)}=0$. The third variant model, denoted by TCLM (IC only), omits the learning of explicit type constraints by fixing $\alpha^{(r,k,l)}=0$. Results show that the learning of type constraints pushes TCLM by absolute gains of 6.8% and 9.7% in the Hit@1 scores on YAGO26K-906 and AirGraph, respectively. Meanwhile, the performance significantly drops when the explicit type constraints or implicit type constraints are omitted. These results confirm that learning both explicit and implicit type constraints helps to significantly improve performance in KGC.

Table 3 reports the comparison results between TCLM with SOTA baselines for link prediction on six benchmark datasets, where the results of baselines are sourced from [5, 6, 22] if not otherwise specified. Since these datasets have no explicit type information on entities, we fixed $\alpha^{(r,k,l)}=0$ for TCLM. Results show that TCLM outperforms existing methods in metrics MRR and Hit@1 on five datasets Family, Kinship, UMLS, WN18RR and YAGO3-10, while embedding-based methods such as TuckER and RotatE achieve higher metric scores on FB15k-237. The superiority of embedding-based methods on FB15k-237 may be due to that there



(a) MRR scores against different rule lengths. (b) MRR scores against different beam sizes. (c) MRR scores against different thresholds.

Figure 3: Comparison under different hyper-parameter settings on the UMLS dataset.

are more relations in FB15k-237 (see Table 1), making the search space hard to be sufficiently explored by a rule-based method.

It is worth noting that NCRL [5] (refer to NCRL*) uses $i+1$ to calculate the ranks and a different data split for evaluation, thus we reproduce their results (refer to NCRL†) to ensure a fair comparison. We can see that the performance of NCRL significantly drops when it is evaluated under the more reasonable setting. This indicates that NCRL cannot precisely estimate the truth degrees of test triples. In contrast, compared with the SOTA rule-based method RNNLogic, TCLM improves performance by absolute gains of 8.9%, 4.8%, 10.0%, 3.3%, 2.2% and 13.2% in the Hit@1 scores on Family, Kinship, UMLS, WN18RR, FB15k-237 and YAGO3-10, respectively. The mean performance difference between TCLM and RNNLogic is statistically significant with a p-value=4.4e-6<0.05 by a two-tailed t-test. This implies that TCLM is superior to SOTA rule-based methods.

To clarify the cause of these improvements, we further conducted an ablation study. Specifically, we created a variant model, denoted by TCLM (w/o IC), by fixing $\alpha^{(r,k,l)} = 0$ in Equation (3) to omit the learning of implicit type constraints. Results show that the performance drops on all datasets when the implicit type constraints were omitted. The mean performance difference between TCLM and TCLM (w/o IC) is statistically significant with a p-value=1.9e-3<0.05 by a two-tailed t-test. This implies that TCLM effectively learns implicit type constraints. Besides, the learning of implicit type constraints achieves significant gains on UMLS, WN18RR and YAGO3-10, as well as slight gains on Family, Kinship and FB15k-237. This improvement difference may be caused by different degrees of effectiveness for type constraints on different datasets.

We also conducted experiments to verify whether the proposed bi-directional learning mechanism benefits for the triple classification task. In more detail, we followed the same setting used in [27] for the evaluation, i.e., judging whether a given triple is true or false by binary classification, where false triples are obtained by corrupting true triples. Table 4 reports the comparison results on UMLS and Kinship, where "Triple classification (head)" (resp. "Triple classification (tail)") denotes that the truth degrees of triples are computed from the angle of answering head (resp. tail) queries. We can observe that the compared baselines and the variant model of TCLM that omits the bi-directional learning mechanism achieve different performance results from different angles of answering queries. In contrast, TCLM is able to achieve the same performance results as expected. Furthermore, TCLM outperforms all compared methods in terms of the F1 score on both datasets. These results confirm that learning the same model to answer both head queries and tail queries helps to improve the performance in triple classification.

Table 5: Comparison on training time against different rule lengths L on UMLS, where TT abbreviates the training time, MC the memory cost on GPU and GB the Gigabytes.

Method	$L=2$		$L=3$		$L=4$		$L=5$	
	TT	MC	TT	MC	TT	MC	TT	MC
AMIE+	0.15s	-	14s	-	> 1day	-	> 1day	-
RNNLogic	320s	0.7GB	350s	0.7GB	969s	0.7GB	> 1day	0.8GB
TCLM	736s	0.9GB	805s	1GB	935s	1.1GB	1281s	1.2GB

5.3 Analysis

Analysis for Learning Longer Rules. To verify whether existing rule-based methods can capture implicit type constraints by learning longer rules, we conducted experiments to compare TCLM with SOTA rule learners AMIE+ and RNNLogic for learning longer rules. Sub-figure (a) in Figure 3 illustrates the comparison results. It can be observed that both TCLM and RNNLogic achieve the best MRR scores when $L=3$ and that the MRR scores drop when L becomes larger. We can also observe that AMIE+ achieves the best MRR score when $L=4$. These results indicate that SOTA rule-based methods such as AMIE+ and RNNLogic cannot achieve better performance by learning longer rules. The reason may lie in the fact that the search space of target rules increases exponentially with the increase of L, imposing a greater challenge for rule-based methods to learn precise rules.

We further analyzed the training time and memory cost on the UMLS dataset, as reported in Table 5. It can be seen that the search-based ILP method AMIE+ is able to mine rules from UMLS in a few seconds when $L \leq 3$ but the training time of AMIE+ significantly increases when $L \geq 4$. We can also see that RNNLogic suffers from the same problem as AMIE+ when learning rules with $L=5$. Note that RNNLogic requires to first search relational paths from the KG, and then utilizes these paths to train a neural-based rule generator. In contrast, TCLM is able to efficiently learn more complex TC-rules even when $L=5$. These results indicate that the large search space prevents search-based methods from learning longer rules efficiently; however, thanks to the end-to-end learning fashion, TCLM excels in learning long TC-rules in moderate time.

Analysis on Hyper-parameters for Rule Extraction. TC-rules can be extracted from the parameters of TCLM by applying Algorithm 1, where the extracted TC-rules with their weights can be used to construct a rule system for KGC. Thus, we conducted experiments to show the impact of different settings of the beam size b and the threshold δ in Algorithm 1 on the test set of UMLS, as illustrated in sub-figures (b) and (c) of Figure 3. In sub-figure

Table 6: Comparison results for efficiency and memory costs on all datasets, where TT abbreviates the training time, ET the evaluation time and MC the memory cost on GPU. The unit h/m/s/GB abbreviates hours/minutes/seconds/Gigabytes.

Method	AirGraph			YAGO20K906			Family			Kinship			UMLS			WN18RR			FB15k-237			YAGO3-10		
	TT	ET	MC	TT	ET	MC	TT	ET	MC	TT	ET	MC	TT	ET	MC	TT	ET	MC	TT	ET	MC	TT	ET	MC
NeuralLP	-	-	-	-	-	-	8m	1m	0.5GB	2m	20s	0.5GB	2m	20s	0.5GB	2h	5m	2GB	>1day	-	15GB	>1day	-	17GB
DRUM	-	-	-	-	-	-	12m	1m	0.5GB	3m	20s	0.5GB	3m	20s	0.5GB	2h	5m	2GB	>1day	-	15GB	>1day	-	17GB
RNNLogic	-	-	-	-	-	-	7m	20s	0.7GB	5m	10s	0.7GB	6m	10s	0.7GB	>1day	-	0.7GB	>1day	-	0.8GB	>1day	-	0.7GB
NCRL	-	-	-	-	-	-	6m	2m	4GB	5m	5m	4GB	3m	>1day	5GB	1m	30s	2GB	7m	>1day	7GB	3m	>1day	7GB
TCLM	3h	20m	5GB	3h	10m	5GB	30m	2m	3GB	10m	1m	1GB	12m	1m	1GB	3h	5m	21GB	15h	30m	5GB	13h	15m	13GB

Table 7: Examples of rules learnt by different systems on AirGraph, where #TC denotes the number of variables with constraints.

Method	Logical rule	#TC
AMIE+	R_1 : capableOfLanding$^{\text{new}}$(x, y) ← capableOfLanding(x, z_0) ∧ hasPhysicalClass(z_0, z_1) ∧ hasGearConfig(z_1, y)	0
AnyBURL	R_2 : capableOfLanding$^{\text{new}}$(x, y) ← capableOfLanding(x, z_0) ∧ isAircraftOf(z_0, z_1) ∧ isAircraftOf(z_1, y)	0
TyRuLe	R_3 : capableOfLanding$^{\text{new}}$(x, y) ← capableOfLanding(x, z_0) ∧ isAircraftEventOf$^-(z_0, z_1)$ ∧ isAircraftEventOf(z_1, y)∧ Runway(x) ∧Airport(z_0) ∧ Airline(z_1)∧ Aircraft(y)	4
	R$_4$: capableOfLanding$^{\text{new}}$(x, y) ← isRunwayOf(x, z_0) ∧ hasBase$^-(z_0, z_1)$ ∧ isAircraftOf$^-(z_1, y)$∧ Runway(x) ∧Aircraft(z_0) ∧ Event(z_1)∧ Aircraft(y)	4
TCLM	R_5 : capableOfLanding$^{\text{new}}$(x, y) ← hasBase(x, z_0) ∧ hasPhysicalClass(z_0, z_1) ∧ hasPhysicalClass$^-(z_1, y)$∧ (Runway(x) ∨ (hasSurface(x, u_0) ∨ isRunwayOf(x, u_1) ∨ capableOfLanding(x, u_2))) ∧ (Aircraft(y) ∨ (hasWingtip(y, u_3) ∨ capableOfLanding$^-(y, u_4)$))	2
	R_6 : capableOfLanding$^{\text{new}}$(x, y) ← hasGearConfig(x, z_0) ∧ hasGearConfig$^-(z_0, y)$∧ (Runway(x) ∨ (hasSurface(x, u_0)∨ isRunwayOf(x, u_1) ∨ capableOfLanding(x, u_2))) ∧ (Aircraft(y) ∨ (hasWakeCategory(y, u_3) ∨ capableOfLanding$^-(y, u_4)$))	2

(b), it can be observed that the rule system (red line) achieves a comparable MRR score (0.762) to TCLM when the beam size is set as 30. Besides, we can observe from sub-figure (c) that the rule system achieves a comparable MRR score (0.763) to TCLM when δ is set to 0.5 and the performance drops when δ becomes larger. Note that $\delta = 1$ means that no type constraint can be extracted. These results imply that the rule system built from the extracted TC-rules can achieve comparable performance to TCLM, while the extracted type constraints have a significant impact on the performance.

Analysis on Efficiency and Memory Cost. According to Equation (1-4), the inference time complexity of TCLM is $O(N(L(2|\mathcal{R}|+1)|\mathcal{E}|^2 + (L+1)(2|\mathcal{R}|+|C|)|\mathcal{E}|+1))$, whereas the inference time complexity of TCLM (w/o types) is $O(NL(2|\mathcal{R}|+1)|\mathcal{E}|^2)$. In general, it holds that $L(2|\mathcal{R}|+1)|\mathcal{E}|^2 \gg (L+1)(2|\mathcal{R}|+|C|)|\mathcal{E}|$ in most real-world scenarios. This indicates that the introduction of type constraints does not significantly increase the inference time complexity of TCLM. To further verify whether TCLM can be learned and evaluated in a reasonable time, we compared TCLM with other neural-based methods in terms of efficiency and memory cost on GPU. The results of this comparison are presented in Table 6. It can be observed that the training time for TCLM across all datasets does not exceed 15 hours. In contrast, baseline methods included NeuralLP, DRUM, and RNNLogic spend more than one day to learn models on FB15k-237 and YAGO3-10. Although NCRL has a shorter training time than TCLM, its evaluation process spends much more time (one day more for three datasets). In contrast, the evaluation of TCLM on every dataset finishes in half an hour. Furthermore, the resource requirement of TCLM on every dataset is modest, with a maximum RAM usage of 21GB on a GPU. This suggests that TCLM can be efficiently executed on most GPU platforms.

Case Study on Learnt Rules. We present a case study for comparing the learnt rules from different systems on the AirGraph dataset, as shown in Table 7. These rules share the same head relation "capableOfLanding$^{\text{new}}$". It can be seen that both AMIE+ and AnyBURL learn chain-like rules only, TyRuLe is able to learn typed rules with explicit type constraints, whereas TCLM is able to learn TC-rules with both explicit and implicit type constraints. The implicit type constraint "(hasSurface(x, u_0) ∨ isRunwayOf(x, u_1) ∨ capableOfLanding(x, u_2))" in both R_5 and R_6 (marked in yellow) entails the explicit type constraint Runway(x) in R_3 and R_4 (marked in yellow). Meanwhile, both the implicit type constraints "(hasWingtip(y, u_3)∨capableOfLanding$^-(y, u_4)$)" and "(hasWakeCategory(y, u_3)∨capableOfLanding$^-(y, u_4)$)" in R_5 and R_6 (marked in blue) respectively entail the explicit type constraint Aircraft(y) in R_3 and R_4 (marked in blue). This case illustrates that expressive TC-rules with type constraints on entity variables can be extracted from the parameter assignment of TCLM.

6 Conclusion and Future Work

In this paper, we have introduced a novel formalism for logical rules named *TC-rules*, which enriches chain-like rules with both explicit and implicit type constraints on entity variables. Furthermore, we have proposed a novel neural model named TCLM for approximating TC-rules. A bi-directional learning mechanism is proposed to enable the extraction of the same set of TC-rules for answering both head and tail queries. Experimental results on eight datasets demonstrate that TCLM is superior to SOTA methods in both the link prediction task and the triple classification task. Future work will extend TCLM to learn TC-rules from multiple sources, such as from both facts in KGs and sentences in text corpora.

References

[1] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. 2007. DBpedia: A Nucleus for a Web of Open Data. In *ISWC*, Vol. 4825. 722–735.

[2] Ivana Balazevic, Carl Allen, and Timothy M. Hospedales. 2019. TuckER: Tensor Factorization for Knowledge Graph Completion. In *EMNLP*. 5184–5193.

[3] Kurt D. Bollacker, Colin Evans, Praveen K. Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*. 1247–1250.

[4] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *NIPS*. 2787–2795.

[5] Kewei Cheng, Nesreen K. Ahmed, and Yizhou Sun. 2023. Neural Compositional Rule Learning for Knowledge Graph Reasoning. In *ICLR*.

[6] Kewei Cheng, Jiahao Liu, Wei Wang, and Yizhou Sun. 2022. RLogic: Recursive Logical Rule Learning from Knowledge Graphs. In *KDD*. ACM, 179–189.

[7] William W. Cohen, Fan Yang, and Kathryn Mazaitis. 2020. TensorLog: A Probabilistic Database Implemented Using Deep-Learning Infrastructure. *J. Artif. Intell. Res. (JAIR)* 67 (2020), 285–325.

[8] David Jaime Tena Cucala, Bernardo Cuenca Grau, Egor V. Kostylev, and Boris Motik. 2022. Explainable GNN-Based Models over Knowledge Graphs. In *ICLR*.

[9] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2018. Convolutional 2D Knowledge Graph Embeddings. In *AAAI*. 1811–1818.

[10] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. 2015. Fast rule mining in ontological knowledge bases with AMIE+. *VLDB J.* 24, 6 (2015), 707–730.

[11] Junheng Hao, Muhao Chen, Wenchao Yu, Yizhou Sun, and Wei Wang. 2019. Universal Representation Learning of Knowledge Bases by Jointly Embedding Instances and Ontological Concepts. In *KDD*. 1709–1719.

[12] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*. http://arxiv.org/abs/1412.6980

[13] Stanley Kok and Pedro M. Domingos. 2007. Statistical predicate invention. In *ICML*, Vol. 227. 433–440.

[14] Ni Lao and William W. Cohen. 2010. Relational retrieval using a combination of path-constrained random walks. *Mach. Learn.* 81, 1 (2010), 53–67.

[15] Xinze Lyu, Guangyao Li, Jiacheng Huang, and Wei Hu. 2020. Rule-Guided Graph Neural Networks for Recommender Systems. In *ISWC*. 384–401.

[16] Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. 2019. Anytime Bottom-Up Rule Learning for Knowledge Graph Completion. In *IJCAI*. 3137–3143.

[17] Pasquale Minervini, Sebastian Riedel, Pontus Stenetorp, Edward Grefenstette, and Tim Rocktäschel. 2020. Learning Reasoning Strategies in End-to-End Differentiable Proving. In *ICML*, Vol. 119. 6938–6949.

[18] Arindam Mitra and Chitta Baral. 2016. Addressing a Question Answering Challenge by Combining Statistical Methods with Inductive Rule Learning and Reasoning. In *AAAI*. 2779–2785.

[19] Stephen H. Muggleton. 1995. Inverse Entailment and Progol. *New Gener. Comput.* 13, 3&4 (1995), 245–286.

[20] Stephen H. Muggleton and Luc De Raedt. 1994. Inductive Logic Programming: Theory and Methods. *J. Log. Program.* 19/20 (1994), 629–679.

[21] Sriraam Natarajan, Tushar Khot, Kristian Kersting, Bernd Gutmann, and Jude Shavlik. 2010. Boosting relational dependency networks. In *ICILP*. 1–8.

[22] Meng Qu, Junkun Chen, Louis-Pascal A. C. Xhonneux, Yoshua Bengio, and Jian Tang. 2021. RNNLogic: Learning Logic Rules for Reasoning on Knowledge Graphs. In *ICLR*.

[23] J. Ross Quinlan. 1990. Learning Logical Definitions from Relations. *Mach. Learn.* 5 (1990), 239–266.

[24] Matthew Richardson and Pedro M. Domingos. 2006. Markov logic networks. *Mach. Learn.* 62, 1-2 (2006), 107–136.

[25] Ali Sadeghian, Mohammadreza Armandpour, Patrick Ding, and Daisy Zhe Wang. 2019. DRUM: End-To-End Differentiable Rule Mining On Knowledge Graphs. In *NeurIPS*. 15321–15331.

[26] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling Relational Data with Graph Convolutional Networks. In *ESWC*, Vol. 10843. 593–607.

[27] Richard Socher, Danqi Chen, Christopher D. Manning, and Andrew Y. Ng. 2013. Reasoning With Neural Tensor Networks for Knowledge Base Completion. In *NIPS*. 926–934.

[28] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)* 15, 1 (2014), 1929–1958.

[29] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *WWW*. 697–706.

[30] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019. RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. In *ICLR*.

[31] Zhiqing Sun, Shikhar Vashishth, Soumya Sanyal, Partha P. Talukdar, and Yiming Yang. 2020. A Re-evaluation of Knowledge Graph Completion Methods. In *ACL*. 5516–5522.

[32] Komal K. Teru, Etienne G. Denis, and William L. Hamilton. 2020. Inductive Relation Prediction by Subgraph Reasoning. In *ICML*, Vol. 119. 9448–9457.

[33] Kristina Toutanova and Danqi Chen. 2015. Observed versus latent features for knowledge base and text inference. In *Workshop on Continuous Vector Space Models and their Compositionality*. 57–66.

[34] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex Embeddings for Simple Link Prediction. In *ICML*, Vol. 48. 2071–2080.

[35] Denny Vrandecic and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM* 57, 10 (2014), 78–85.

[36] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge Graph Embedding by Translating on Hyperplanes. In *AAAI*. 1112–1119.

[37] Hong Wu, Zhe Wang, Kewen Wang, and Yi-Dong Shen. 2022. Learning Typed Rules over Knowledge Graphs. In *KR*.

[38] Chenyan Xiong, Russell Power, and Jamie Callan. 2017. Explicit Semantic Ranking for Academic Search via Knowledge Graph Embedding. In *WWW*. 1271–1279.

[39] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *ICLR*.

[40] Fan Yang, Zhilin Yang, and William W. Cohen. 2017. Differentiable Learning of Logical Rules for Knowledge Base Reasoning. In *NIPS*. 2319–2328.

[41] Yuan Yang and Le Song. 2020. Learn to Explain Efficiently via Neural Logic Inductive Learning. In *ICLR*.

[42] Qiang Zeng, Jignesh M. Patel, and David Page. 2014. QuickFOIL: Scalable Inductive Logic Programming. *VLDB* 8, 3 (2014), 197–208.