DIFFERENTIABLE LOGIC PROGRAMMING FOR PROBABILISTIC REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper studies inductive logic programming for probabilistic reasoning. The key problems, i.e. learning rule structures and learning rule weights, have been extensively studied with traditional discrete searching methods as well as recent neural-based approaches. In this paper, we present a new approach called Differentiable Logic Programming (DLP), which provides a flexible framework for learning first-order logical rules for reasoning. We propose a continuous version of optimization problem for learning high-quality rules as a proxy and generalize rule learning and forward chaining algorithms in a differentiable manner, which enables us to efficiently learn rule structures and weights via gradient-based methods. Theoretical analysis and empirical results show effectiveness of our approach.

1 INTRODUCTION

Learning to reason and predict is a fundamental problem in the fields of machine learning. Representative efforts on this task include neural networks (NN) and inductive logic programming (ILP). The NNs and ILP methods represent learning strategies of two extremes: the ideas behind NNs are to use fully differentiable real-valued parameters to perceive the patterns of data, while in the fields of ILP, we search for determined and discrete structures to match the patterns of data. Over the years, the former approaches, i.e. neural-based methods, have achieved state-of-the-art performance in solving tasks from many different fields, while the latter ones have fallen behind due to their inherited inferior in fitting noisy and probabilistic data.

However, it was pointed out that there is a debate over the problems of systematicity and explanability in connectionist models, as they are black-box models that are hard to be explained. To tackle the problem, numerous methods have been proposed to combine the advantages of both connectionist and symbolic systems. Most existing efforts focus on two different manners: using logic to enhance neural networks and using neural networks to help logical reasoning. The former approaches (Rocktäschel & Riedel (2017), Minervini et al. (2020c), Vedantam et al. (2019), Dong et al. (2019)) modify the structures of NNs to capture some features of logic. Some of them, known as neural theorem provers (Rocktäschel & Riedel (2017), Minervini et al. (2020a)), represent entities with embedding vectors that implies the semantics of them. Further more, they absorb symbolic logical structures into the neural reasoning framework to enhance the expressiveness of the models. For example, to prove the existence of $(grandfather, Q, Bart)$ where Q is the target entity we wish to find, these systems use logic rules such as $grandfather \leftarrow father_of, parent_of$ to translate the original goal $(grandfather, Q, Bart)$ into subgoals that can be subsequently proved by operating on entity embeddings. Thus, the expressiveness and interpretability of the systems is improved with the help of logic.

The latter approaches (Yang et al. (2017), Xiong et al. (2017) Sadeghian et al. (2019), Qu et al. (2021)) enhance traditional inductive logic programming with the help of neural networks. Generally, they use different techniques to solve the key problems of ILP, which is to learn structures of logical rules from exponential large space. Some of them (Yang et al. (2017), Sadeghian et al. (2019), Yang & Song (2020)) approximate the evaluation of all possible chain-like logic rules in a single model, making learning of the model differentiable. However, as mentioned in Sadeghian et al. (2019), these models inevitable give incorrect rules with high confidence values due to the low-rank approximation of evaluating exponential many logic rules at the same time, which also

makes it hard to identify high-quality logic rules and explain the predictions made by these models. The other line of the research (Yang & Song (2020), Qu et al. (2021)) propose different methods to generate high-value rules such as reinforce learning and EM algorithms. However, since structure learning of logical rules is a very hard problem, they are limited in only searching chain-like horn clauses, which is less expressive and general.

In this paper, we propose a novel differentiable programming framework, called Differentiable Logic Programming (DLP), to build a bridge between the ideas of differentiable programming and symbolic reasoning. Our approach enjoys the merits of connectionist systems, i.e., high expressiveness and easy to learn, as well as the merits of ILP systems, i.e., explanability and clear structures for decision making. We study the construction of a probabilistic reasoning model, and discuss the properties of valuable logic rules. Based on that, we propose a novel rule learning framework that approximates the combinatory search problem with a continuous relaxation which enables us to learn structures of logic rules via a differentiable program. Once valuable rules are learnt, we can further fine-tune the rule weights and perform probabilistic forward chaining to predict the existence of unobserved terms.

2 Related Work

Our work is related to previous efforts on Inductive Logic Programming (ILP) fields and their extensions. Representative methods of ILP includes FOIL (Quinlan (2004)), MDIE (Muggleton (2009)), AMIE (Galárraga et al. (2015)), Inspire (Schüller & Kazmi (2018)), RLvLR (Omran et al. (2018)) and so on. Generally, these methods search for logic rules in exponential large space to obtain valuable logic rules and make predictions based on them. However, despite the well-designed searching algorithms and pruning techniques, these methods suffer from their inherent limitations of relying on discrete counting and predefined confidence.

More recently, different learning algorithms have been proposed to overcome the drawbacks of ordinary ILP methods. Many of them consider a special kind of ILP tasks namely knowledge graph completion, where most of the proposed methods (Yang et al. (2017), Rocktäschel & Riedel (2017), Sadeghian et al. (2019), Minervini et al. (2020b), Yang & Song (2020), Qu et al. (2021)) focus on learning chain-like rules, and these methods use different learning strategies to learn valuable rules. Some of them are based on reinforcement learning (Xiong et al. (2017), Chen et al. (2018), Das et al. (2018), Lin et al. (2018), Shen et al. (2018)), and they train agents to find the right reasoning paths to answer the questions in knowledge graphs. Qu et al. (2021) uses recurrent neural networks as rule generators and train them with EM algorithms. Yang et al. (2017), Sadeghian et al. (2019) and Yang & Song (2020) propose end-to-end differentiable methods, which can be trained efficiently with gradient-based optimizers. These methods are similar in spirit with our approach, as they claim to be able to learn rule structures in a differentiable manner. However, what they actually do is to find a low-rank tensor approximation for simultaneous execution of all possible rules of exponential space with different confidence scores, and by doing so, they suffer from the risk of assigning wrong rules with high scores (Sadeghian et al. (2019)). Also, although Yang & Song (2020) claims that their attentions usually becomes highly concentrate after convergence, there is no theoretical guarantee so extracting logic rules implying these model could be problematic because there might be exponential potential rules that have confidence scores higher than zero. Compared with them, our method explicitly learns logic rules for reasoning with a more flexible rule search space while keeping the rule learning procedure differentiable.

There are other methods that focus on different types of ILP problems. Lu et al. (2022) treats relation prediction task as a decision making process, and they use reinforcement learning agents to select the right paths between heads and tails. Our approach is more general and is able to deal with different tasks. Rocktäschel & Riedel (2017) and Minervini et al. (2020b) propose a generalized version of backward chaining with the help of neural embedding methods, and show great performance on both relation prediction and knowledge graph completion tasks. Compared to them, our approach doesn't require the help of embeddings, thus our predictions are more explainable.

There are also interesting methods based on embedding and neural networks (Bordes et al. (2013), Wang et al. (2014), Yang et al. (2015), Nickel et al. (2016), Trouillon et al. (2016), Cai & Wang (2018), Dettmers et al. (2018), Balazevic et al. (2019), Sun et al. (2019), Teru et al. (2020), Zhu et al. (2021)). Since they are less relevant to logic reasoning, we do not cover them in details here.

3 PRELIMINARY

3.1 FIRST-ORDER LOGIC

This paper focus on learning first-order logic (FOL) rules for reasoning, and here we first introduce main concepts of FOL briefly. Across this paper, we assume predicates are from a countable universe $\mathcal{P}$ where we use uppercase $A, B, C, ... \in \mathcal{P}$ to represent predicates. We use P to represent unspecified predicates, Q to represent query predicates and target predicates of logic rules. We use $a, b, c, ... \in \mathcal{V}$ to represent constants and $x, y, z \in \mathcal{V}$ to represent variables. An example way for constructing FOL formulas would be as follows:

$$\begin{aligned} \varphi(x) &:= P(x) \mid \neg\varphi(x) \mid \varphi(x) \wedge \varphi(x) \mid \varphi(x) \vee \varphi(x) \mid \exists z(\varphi(x, y)), \\ \varphi(x, y) &:= P(x, y) \mid P(x) \mid \exists z : \varphi(x, z) \wedge \varphi(z, y). \end{aligned} \tag{1}$$

Across this paper, we call definitions of logic formulas like Eq. 1 the generative definitions of FOL. Such definitions are critical for ILP systems, because they not only define the syntax of a FOL formula, but also determine the expressive power and search space of FOL formulas. However, in this paper we will not restrict the specific formulation of the generative definitions. Instead, we use a common form to represent them and our approach is equivalently applicable to any reasonable generative definitions:

$$\varphi(\mathbf{x}) := P(\mathbf{x}) \mid \mathrm{F}_1(\mathbf{x}) \mid \mathrm{F}_2(\mathbf{x}) \mid \mathrm{F}_3(\mathbf{x}) \mid ... \tag{2}$$

We use $\mathbf{x}$ to represent tuples of variables for notation simplification given it's clear in the context. F_i represents a possible format that φ could take. For example, in the definition of $\varphi(x)$ in Eq. 1, we have $\mathbf{x} = x$ and $\mathrm{F}_1(\mathbf{x}) = \neg\varphi(x)$, $\mathrm{F}_2(\mathbf{x}) = \varphi(x) \wedge \varphi(x)$; In the definition of $\varphi(x, y)$ in Eq. 1, we have $\mathbf{x} = (x, y)$ and $\mathrm{F}_1(\mathbf{x}) = P(\mathbf{x})$, $\mathrm{F}_2(\mathbf{x}) = \exists z : \mathrm{F}'_2(\mathbf{x}, z)$ and $\mathrm{F}'_2(\mathbf{x}, z) = \varphi(x, z) \wedge \varphi(z, y)$.

Logic classifiers and logic rules Logic formulas φ can be regarded as classifiers (Barceló et al. (2020)). For example, consider $\varphi(x) := \exists y : \varphi'(y) \wedge \mathrm{Edge}(x, y)$, $\varphi'(x) := \mathrm{Red}(x) \vee \mathrm{Blue}(x)$ and $\mathrm{Blue(x)} \leftarrow \varphi(x)$. In this case, φ and φ' can be regarded as classifiers, where we have $\varphi(x) = 1$ for nodes x with red or blue neighbors and 0 otherwise. Generally, logic classifiers take (set of) entities (e.g. node x) as input and compute the output (e.g. $\varphi(x)$) by applying the logic formula on the background statements (e.g. $\{\mathrm{Edge}(x, y), \mathrm{Red}(y)\}$), and often one classifier (e.g. $\varphi(x)$) is built upon other classifiers (e.g. $\varphi'(x)$) to enhance its expressive power. The last term, i.e. $\mathrm{Blue(x)} \leftarrow \varphi(x)$, is a logic rule for deducing the target predicate $\mathrm{Blue(x)}$, indicating we may conclude a node is Blue if it satisfies φ. In the fields of probabilistic reasoning, each logic rule can be assigned with a weight indicating the degree of certainty of the rule.

Forward chaining and backward chaining Forward chaining methods are critical in automated deduction, as they enable us to repeatedly deduce new lemmas from known theorems. Forward chaining starts from known conditions and logic rules, and move forward towards a conclusion by applying the logic rules. Then, they absorb the deduced conclusions into known conditions and apply the rules again until there are no further new conclusions deduced. Backward chaining methods are the opposite of forward chaining, as they move backwards from the conclusions to the potential conditions implied by the rules.

3.2 PROBLEM STATEMENT

This paper studies probabilistic inductive logic programming. The input data is composed of sets of First-order Logic (FOL) statements of the form $P(\mathbf{x})$, e.g., $\mathrm{Father}(a, b)$ and $\mathrm{Happy(c)}$. Each data instance can be discribed as $(a, Q(\mathbf{x}), \mathcal{B})$ where $\mathcal{B} = \{P(\mathbf{x})\}$ is the background statements, $Q(\mathbf{x})$ is the query and $a \in \{0, 1\}$ is the answer indicating whether the query is true. Our target is to learn a set of probabilistic logic classifiers and use them to perform inductive reasoning to predict the existence of queries based on background statements, which leads to the following problems:

- **Recognition of Logical Patterns.** Learn a set of logic classifiers $\Phi = \{\varphi\}$ that explain how the answers are deduced.
- **Probabilistic Reasoning.** Learn how to make decisions $p_\Phi(Q(\mathbf{x})|\mathcal{B})$ based on the groundings of classifiers Φ.

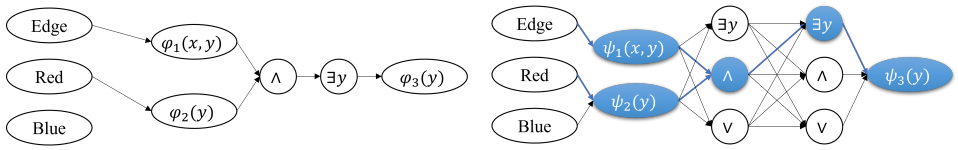

Figure 1: Illustration of logic classifiers (left) and DLP framework (right). The target rule is $\text{Blue}(x) \leftarrow \exists y : \text{Red}(y) \wedge \text{Edge}(x, y)$. In the left figure, $\varphi_3(y)$ directly captures the target rule, while in the right figure, the output unit ψ_3 learns to identify the target logic rule via gradient descent and converges at the correct (colored as blue) reasoning paths.

4 MODEL

In this section, we introduce our proposed Differentiable Logic Programming (DLP) framework. The general idea is to use differentiable programs to solve the problems of learning logic classifiers and learning rule weights for probabilistic reasoning.

As mentioned in Sec. 3, the learning problems require us to identify important logic classifiers from discrete space and assign feasible weights to them. In this paper, we introduce a differentiable module called Logic Perceptron (LP) to help us deal with the problem. Given a generative definition of logic classifiers φ, we provide a method to construct corresponding LPs ψ that share the expressive power of φ so that with different parameter settings, ψ is able to capture any single logic classifier φ. We can stack multiple LPs together to compose a network so that the LPs at the output level can capture more complex logic classifiers as well as keeping the whole network end-to-end differentiable.

Learning the parameters of the network is corresponded to learning the structures and weights of logic classifiers. We construct a new optimization problem as a proxy and each output LP learns to capture a logic classifier by optimizing on it. This learning procedure can be repeated for multiple times while dynamically enlarging the network by adding more LPs to obtain more valuable rules. The same network can also be used to learn the weights of rules simply by fixing some of the parameters and optimizing by maximizing the likelihood. Since the network is differentiable, each optimization problem can be easily solved with gradient-based methods. Figure 1 presents a brief illustration of general ideas behind our model. Next, we introduce the details of our approach.

4.1 OVERVIEW

In this section we first introduce the general ideas behind our approach, as well as highlighting the key challenges in our learning framework. The details of model implementation are discussed in the next sections. We start with an example which illustrates the general ideas of probabilistic reasoning as well as providing the intuitions and motivations behind our approach.

Example 4.1 (Human Reasoning). *Consider the query "*Are a and b friends?*". In this case, we have* $Q(\mathbf{x}) = \text{Friend}(a, b)$. *To answer the query, one may first ask "*Do a and b know each other?*" which corresponds to a logical classifier* $\varphi_1(a, b) = \text{Know}(a, b)$. *If* $\varphi_1(a, b) = 1$, *our confidence in x and y being friends is increased, and we call this* φ_1 *proved Q, also Q is the target predicate of* φ_1, *denoted as* $Q \leftarrow \varphi_1$. *With the answer of* φ_1, *one may continue to ask* φ_2*: "*Do a and b live in the same town?*", ..., where each evaluation of* φ_i *serves as an evidence that proves the existence of* Q.

In the example, we use different logic classifiers to prove the query from different aspects, and all the classifiers used are highly relevant to the target predicate Friend. This makes sense because our belief in x and y being friends is higher when we realize they know each other, which corresponds to a relative high value of $p(\text{Friend}(x, y)|\text{Know}(x, y) = 1)$, but irrelevant facts such as "they both drink water" won't help. In fact, this simple example illustrates the overall ideas of our model, formally described as follows.

Proposition 4.2 (Properties of Φ). *Given the input data* $\mathcal{B}$, $\Phi_Q = \{\varphi_1, \varphi_2, ..., \varphi_L\}$ *and* $p \in (0, 1]$, *if there is a permutation of elements of* Φ_Q *as* $\varphi_{i1}, \varphi_{i2}, ...$ *that satisfies: (1) For* φ_{i1}, *its precision*

on $\mathcal{B}$: $p_{\mathcal{B}}(Q(\mathbf{x})|\varphi_{i1}(\mathbf{x})=1)=\frac{\sum_{\mathbf{x}} Q(\mathbf{x})\varphi_{i1}(\mathbf{x})}{\sum_{\mathbf{x}} \varphi_{i1}(\mathbf{x})} \geq p$; (2) For φ_{ij}, we remove all queries $Q(\mathbf{x})=1$ and also proved true by previous $\varphi_{i1}, \varphi_{i2}, ..., \varphi_{i(j-1)}$ and (3) on the remaining data, we still have $p_{\mathcal{B}}(Q(\mathbf{x})|\varphi_{ij}(\mathbf{x})=1) \geq p$; (4) There exists k such that for φ_{ik}, there is no true statements $Q(\mathbf{x})=1$ left. Then, we can construct a prediction model $p_w(Q(\mathbf{x})|\Phi_Q(\mathbf{x}))$ such that its error rate on $\mathcal{B}$ satisfies:

$$\mathrm{Err}[p_w;\mathcal{B}] \leq \frac{(1-p)N_Q}{pN} \leq 1-p, \tag{3}$$

where $N_Q=\sum_{\mathbf{x}} Q(\mathbf{x})$ and $N=\sum_{\mathbf{x}} 1$.

Prop. 4.2 not only provides justification for the intuitions above, but also implies how we learn such logic classifiers for proving Q as follows. In this paper, instead of finding logic classifiers that have a precision higher than a fixed threshold p, we simply find logic classifiers having high precision

$$\max_{\varphi} \quad p(Q(\mathbf{x})|\varphi(\mathbf{x})=1). \tag{4}$$

After identifying logic classifiers via the optimization problem 4, we apply the logic classifiers learnt on training data and remove, or reduce the weights of data instances where $Q(\mathbf{x})=\varphi_i(\mathbf{x})=1$ and again use the modified (reweighted) data to find another classifiers until the learnt classifiers are enough to make accurate predictions.

So far we have introduced the overall learning framework of our approach except two problems: how we identify high-precision logic classifiers (Eq. 4) and how we learn the prediction model $p_{\Phi_Q}(Q(\mathbf{x})|\mathcal{B})$. In this paper, we propose a uniform inference model to solve these problems. By optimizing on different target functions, our model is able to identify valuable logic rules as well as learning the prediction model for them. In the next sections we discuss the implementation of our model in details. In Sec. 4.2, we introduce logic perceptrons as building blocks of our model. In Sec. 4.3 we introduce how the whole inference model is constructed with logic perceptrons. Then, we present our proxy problem for learning the model parameters in Sec. 4.4.

4.2 Logic Perceptrons

A Logic Perceptron (LP) is a differentiable model that generalizes the ordinary logic classifiers into continuous space. It is a function parameterized over real-valued vectors. Given the generative definition of logic classifiers:

$$\varphi(\mathbf{x}) := A(\mathbf{x}) \mid \mathrm{F}_1(\mathbf{x}) \mid \mathrm{F}_2(\mathbf{x}) \mid \mathrm{F}_3(\mathbf{x}) \mid ..., \tag{5}$$

We can construct LPs that capture the expressive power of φ as follows.

$$\begin{aligned} &\psi(\mathbf{x};\alpha) = (\boldsymbol{\theta}^T)^{\alpha}\left[\mathcal{F}_1(\mathbf{x};\alpha), \mathcal{F}_2(\mathbf{x};\alpha), \mathcal{F}_3(\mathbf{x};\alpha), ...\right]^T, \\ &s.t. \quad \sum_i \theta_i = 1, \quad \theta_i \geq 0 \; for \; i=1,2,... , \end{aligned} \tag{6}$$

where $\boldsymbol{\theta}$ is a learnable parameter of ψ, and α serves as a hyperparameter. The evaluation of each $\mathcal{F}_i(\mathbf{x};\alpha)$ in Eq. 6 is corresponded to each $\mathrm{F}_i(\mathbf{x})$ in Eq. 5, where we define

$$\begin{aligned} &\mathrm{F}_i(\mathbf{x}) := \varphi(\mathbf{x}) && \Longleftrightarrow \mathcal{F}_i(\mathbf{x};\alpha) = \left[P_1(\mathbf{x}), P_2(\mathbf{x}), ..., P_{|\mathcal{P}|}(\mathbf{x}), \psi'_i(\mathbf{x})\right]\left(\boldsymbol{w}^{(i)}\right)^{\alpha}, \\ &\mathrm{F}_i(\mathbf{x}) := \mathrm{F}_j(\mathbf{x}) \wedge \mathrm{F}_k(\mathbf{x}) && \Longleftrightarrow \mathcal{F}_i(\mathbf{x};\alpha) = \mathcal{F}_j(\mathbf{x};\alpha)\mathcal{F}_k(\mathbf{x};\alpha), \\ &\mathrm{F}_i(\mathbf{x}) := \mathrm{F}_j(\mathbf{x}) \vee \mathrm{F}_k(\mathbf{x}) && \Longleftrightarrow \mathcal{F}_i(\mathbf{x};\alpha) = \left[\mathcal{F}_j(\mathbf{x};\alpha), \mathcal{F}_k(\mathbf{x};\alpha)\right]\left(\boldsymbol{w}^{(i)}\right)^{\alpha}, \\ &\mathrm{F}_i(\mathbf{x}) := \neg \mathrm{F}_j(\mathbf{x}) && \Longleftrightarrow \mathcal{F}_i(\mathbf{x};\alpha) = 1-\mathcal{F}_j(\mathbf{x};\alpha), \\ &\mathrm{F}_i(\mathbf{x}) := \exists \mathbf{y} : \mathrm{F}_j(\mathbf{x},\mathbf{y}) && \Longleftrightarrow \mathcal{F}_i(\mathbf{x};\alpha) = \sum_{\mathbf{y}} \mathcal{F}_j(\mathbf{x},\mathbf{y};\alpha), \\ &\mathrm{F}_i(\mathbf{x},\mathbf{y}) := \mathrm{F}_j(\mathbf{x}) && \Longleftrightarrow \mathcal{F}_i(\mathbf{x},\mathbf{y};\alpha) = \mathcal{F}_j(\mathbf{x};\alpha), \end{aligned} \tag{7}$$

where $\sum_j w_j^{(i)} = 1$, $w_j^{(i)} \geq 0$ are learnable parameters, $P_i \in \mathcal{P}$ and $\psi'(\mathbf{x})$ refers to another LP. We exclude the universal quantifier $\forall$ because this can be equivalent expressed by $\neg \exists \neg$. Thus, with the above procedure, we can construct LPs given generative definitions of logic and express the various structures of logic classifiers within one type of LPs.

4.3 INFERENCE NETWORK

In this section, we introduce how we stack LPs to construct the inference network and how we use them for inference. We first assume we already have a set of LPs $\Psi = \{\psi\}$, and let $\Psi_Q \subset \Psi$ be the set of LPs that prove Q, i.e., $Q \leftarrow \psi_Q \in \Psi_Q$. We describe the inference network as a graph $\mathcal{G}_\Psi$, where each $\psi \in \Psi$, each $\mathcal{F}_i$ we used for building ψ and every predicate correspond to a node in the graph. Recall that in the definitions in Eq. 6 and Eq. 7, each node $\mathcal{F}_i$ and ψ is built upon other nodes, for example, in Eq. 7 where $\mathcal{F}_i = \mathcal{F}_j\mathcal{F}_k$, the node $\mathcal{F}_i$ is built upon $\mathcal{F}_j$ and $\mathcal{F}_k$. As a result, for each node (e.g. $\mathcal{F}_i$) in $\mathcal{G}_\Psi$, we connect an edge from other nodes it is built upon (e.g. $\mathcal{F}_j$ and $\mathcal{F}_k$) to itself.

We now discuss how we infer $p_\Psi(Q(\mathbf{x})|\mathcal{B})$ for every $\mathbf{x}$ by generalizing forward chaining. For each predicates $P \in \mathcal{P}$ and every $\mathbf{x}$, we assign $P_0(\mathbf{x}) = 1$ if it is observed as true and $P_0(\mathbf{x}) = 0$ otherwise. Then, at each iteration, we:
(1) Evaluate LPs: We evaluate every $\psi \in \Psi$ at every $\mathbf{x}$. We compute the values of ψs in topological order of $\mathcal{G}_\Psi$. This procedure costs $O(|\mathcal{V}|^n|\mathcal{P}|N)$ where N is the size of the network, n is the maximum arity of predicates and LPs when evaluating ψ.
(2) Update inferences: We update our inferences about $Q(\mathbf{x})$ for each $Q \in \mathcal{P}$ and $\mathbf{x}$. This procedure costs $O(|\mathcal{V}|^n R_\Psi)$, where R_Ψ is the amount of rules, i.e., the total size of Ψ_Q for each Q.

$$\begin{aligned} \hat{Q}(\mathbf{x}) &= \sigma\left(\mathrm{Update}\left(\{\psi_Q(\mathbf{x})|\psi_Q \in \Psi_Q\}\right)\right), \\ Q(\mathbf{x}) &= \max\{\hat{Q}(\mathbf{x}), Q_0(\mathbf{x})\}, \end{aligned} \tag{8}$$

where Update is the update function that specifies how we combine the evaluations of $\psi_Q \in \Psi_Q$ on $\mathbf{x}$ to predict the existence of $Q(\mathbf{x})$. A common implementation of the update function is

$$\mathrm{Update}\left(\{\psi_Q(\mathbf{x})|\psi_Q \in \Psi_Q\}\right) = \sum_{\psi_Q \in \Psi_Q} w_{\psi_Q}\psi_Q(\mathbf{x}), \tag{9}$$

which is a weighted sum of different LPs' evaluations, and w_{ψ_Q} corresponds to the weight of the rule $Q \leftarrow \psi_Q$. After T rounds of iterations, we directly pick the values of $Q(\mathbf{x})$ as an approximation of $p_\Psi(Q(\mathbf{x})|\mathcal{B})$.

Also, the time complexities we provided here are rather loose. In reality, input data is often very sparse, and it turns out that the $|\mathcal{V}|^n$ term in the time complexities can be significantly reduced. See appendix for more discussion.

Construction of networks So far we have described how we use a set of LPs to compose an inference network, and there remains the problem of how we construct such a network from scratch. The topological structure of $\mathcal{G}_\Psi$ is flexible and can represent some kinds of prior knowledge. A simple and straight approach is as follows: we create output LPs from the definitions in Eq. 7, and whenever another LP ψ' is needed to be specified, we simply create a new LP to serve as ψ'. This procedure is recursively repeated for a couple of times until in the end we simply assign an empty node as ψ' to end the recurrence.

4.4 LEARNING

In this section, we deal with the learning problem in Sec. 3 including learning the structures and weights of logic classifiers. As shown in the following, we can solve the two problems using the same inference model with different optimization problems and parameter settings.

Structure Learning via Continuous Optimization We now discuss how we learn the structures of logic classifiers, i.e., to solve the problem

$$\max_{\varphi} \quad p(Q(\mathbf{x})|\varphi(\mathbf{x}) = 1). \tag{10}$$

Proposition 4.3 (Proxy problem). *Consider the optimization problem*

$$\min_{\psi} \quad \mathcal{L}(\psi) = -\log \mathbb{E}\left[\psi(\mathbf{x};\alpha)|Q(\mathbf{x}) = 1\right] + \log \mathbb{E}\left[\psi(\mathbf{x};\beta)|Q(\mathbf{x}) \neq 1\right], \tag{11}$$

where $\psi(\mathbf{x};\alpha)$ and $\psi(\mathbf{x};\beta)$ are obtained by one iteration of the forward procedure. With the following constraints being satisfied: (1) $\alpha > \beta > 1$; (2) The unfixed nodes in the paths for computing ψ

compose a tree; (3) Negations are applied only on fixed nodes. Then we have:
(1) At each local minima of $\mathcal{L}(\psi)$, ψ captures a logic classifier;
(2) Minimization of the proxy problem yields a near-optimal solution for problem 10.

Here, $fixed$ nodes refers to predicates and LPs that are already trained by the proxy problem and converged so they have no internal trainable parameters; others are $unfixed$ nodes. A LP ψ captures a logic classifier φ when for all $\mathbf{x}$, $\psi(\mathbf{x}) > 0 \iff \varphi(\mathbf{x}) > 0$. With Prop. 4.3, we can find the solutions of the original problem by optimizing on the proxy problem instead. Intuitively, by training on the proxy problem, each LP learns to concentrate on the operators and predicates that give the most activation, and the whole structure is determinant within this procedure. The proxy problem 4.3 provides a method for learning the structures of a node ψ_Q to prove Q in the network, and this optimization procedure can be repeated multiple times for different $\psi'_Q \in \Psi_Q$ to learn more classifiers.

Note that the 3 constraints are sufficient conditions for the optimization problem to guarantee to converge at the points where ψ captures a logic classifier, but they are not always necessary. In most situations, we can ignore these constraints and set $\alpha = \beta = 1$ or 2 for simplification. Also, the construction procedure provided in Sec. 4.3 naturally satisfies the constraint (2). See appendix C for more discussion of how these constraints work.

Learning Weights To learn the weights of ψs, we fix the parameters of each $\psi \in \Psi$. Then, we proceed the inference steps for a fixed number or till convergence. The obtained $p_\Psi(Q(\mathbf{x})|\mathcal{B})$ for each $Q \in \mathcal{P}$ and $\mathbf{x}$ is differentiable w.r.t. weights w_ψs, so we can optimize the weights by simply maximizing the likelihood of $p_\Psi(Q(\mathbf{x})|\mathcal{B})$ on data.

With the proposed approaches, the whole learning procedure of our model described in Sec. 4.1 is realized where we use the inference network to both solve the problems of learning rule structures and learning the prediction model.

5 EXPERIMENT

5.1 DATASETS

We consider datasets from different fields to test the model's ability in solving ILP tasks, systematic reasoning and knowledge graph completion. These datasets include:

ILP tasks We test the model's expressiveness by applying the model to solve the 20 ILP tasks proposed in Evans & Grefenstette (2018). These tasks test the expressive power of an ILP model, including learning the concepts of even numbers, family relations, graph coloring, etc.

Systematicity We test the model's systematicity (Lu et al. (2022)) on CLUTRR (Sinha et al. (2019)) datasets. These tasks test a model's ability of generalizing on unseen data of different distributions. Models are trained on small scale of data and tested on larger data with longer resolution steps.

Knowledge graph completion We test the model's capability of performing probabilistic reasoning on knowledge graphs including UMLS, Kinship (Kok & Domingos (2007)), WN18RR (Dettmers et al. (2018)) and FB15k-237 (Toutanova & Chen (2015)). These tasks test a model's ability of dealing with probabilistic and noisy data. For Kinship and UMLS, since there are no standard data splits, we take the data splits from Qu et al. (2021).

5.2 COMPARED ALGORITHMS

We observe there are few models capable of solving all the tasks, so we pick different algorithms for comparison for different tasks. For ILP tasks, we choose Evans & Grefenstette (2018). For systematic reasoning, we choose Graph Attention Networks (GAT) (Velickovic et al. (2018)), Graph Convolutional Networks (GCN) (Kipf & Welling (2017)), Recurrent Neural Networks (RNN) (Schuster & Paliwal (1997)), Long Short-Term Memory Networks (LSTM) (Hochreiter & Schmidhuber (1997)), Gated Recurrent Units (GRU) (Cho et al. (2014)), Convolutional Neural Networks (CNN)

Table 1: Results on CLUTRR.

Method	Short Stories							Long Stories					
	4Hops	**5Hops**	**6Hops**	**7Hops**	**8Hops**	**9Hops**	**10Hops**	**5Hops**	**6Hops**	**7Hops**	**8Hops**	**9Hops**	**10Hops**
DLP	**.990±.02**	**.994±.01**	**1.0±.00**	**.996±.01**	**.997±.01**	**.996±.01**	**1.0±.00**	.989±.02	.990±.02	**.994±.03**	**1.0±.00**	**.992±.01**	**.996±.01**
R5	.98±.02	.99±.02	.98±.03	.96±.05	.97±.01	.98±.03	.97±.03	.99±.02	.99±.04	.99±.03	1.0±.02	.99±.02	.98±.03
CTP_L	.98±.02	.98±.03	.97±.05	.96±.04	.94±.05	.89±.07	.89±.07	.99±.02	.98±.04	.97±.04	.98±.03	.97±.04	.95±.04
CTP_A	.99±.02	.99±.01	.99±.02	.96±.04	.94±.05	.89±.08	.90±.07	.99±.04	.99±.03	.97±.03	.95±.06	.93±.07	.91±.05
CTP_M	.97±.03	.97±.03	.96±.06	.95±.06	.93±.05	.90±.06	.89±.06	.98±.04	.97±.06	.95±.06	.94±.08	.93±.08	.90±.09
GNTP	.49±.18	.45±.21	.38±.23	.37±.21	.32±.20	.31±.19	.31±.22	.68±.28	.63±.34	.62±.31	.59±.32	.57±.34	.52±.32
GAT_s	.91±.02	.76±.06	.54±.03	.56±.04	.54±.03	.55±.05	.45±.06	.99±.00	.85±.04	.80±.03	.71±.03	.70±.03	.68±.02
GCN_s	.84±.03	.68±.02	.53±.03	.47±.04	.42±.03	.45±.03	.39±.02	.94±.03	.79±.02	.61±.03	.53±.04	.53±.04	.41±.04
RNN_s	.86±.06	.76±.08	.67±.08	.66±.08	.56±.10	.55±.10	.48±.07	.93±.06	.87±.07	.79±.11	.73±.12	.65±.16	.64±.16
$LSTM_s$	.98±.04	.95±.03	.88±.05	.87±.04	.81±.07	.75±.10	.75±.09	.98±.03	.95±.04	.89±.10	.84±.07	.77±.11	.78±.11
GRU_s	.89±.05	.83±.06	.74±.12	.72±.09	.67±.12	.62±.10	.60±.12	.95±.04	.94±.03	.87±.08	.81±.13	.74±.15	.75±.15
$CNNH_s$	.90±.04	.81±.05	.69±.10	.64±.08	.56±.13	.52±.12	.50±.12	.99±.01	.97±.02	.94±.03	.88±.04	.86±.05	.84±.06
CNN_s	.95±.02	.90±.03	.89±.04	.80±.05	.76±.08	.69±.07	.70±.08	**1.0±.00**	**1.0±.01**	.98±.01	.95±.03	.93±.03	.92±.04
MHA_s	.81±.04	.76±.04	.74±.05	.70±.04	.69±.03	.64±.05	.67±.02	.88±.03	.83±.05	.76±.04	.72±.04	.74±.05	.70±.03

(Kim (2014)), CNN with Highway Encoders (CNNH) (Kim et al. (2016)), Greedy Neural Theorem Provers (GNTP) (Minervini et al. (2020a)), Multi-Headed Attention Networks (MHA) (Vaswani et al. (2017)), Conditional Theorem Provers (CTP) (Minervini et al. (2020b)), R5 (Lu et al. (2022)). For knowledge graph completion, we choose rule-based methods including Markov Logic Networks (MLN) (Richardson & Domingos (2006)), PathRank (Lao & Cohen (2010)), NeuralLP (Yang et al. (2017)), DRUM (Sadeghian et al. (2019)), CTP (Minervini et al. (2020b)), M-Walk (Shen et al. (2018)), MINERVA (Das et al. (2018)), NLIL (Yang & Song (2020)), RNNLogic (Qu et al. (2021)).

5.3 RESULTS

1. Comparison with other methods. The main results of systematicity tests and KG completion are shown in Tab. 1, Tab. 2 and Tab. 3. The 20 ILP tasks are pass-or-fail tests and our model is able to solve all of them, indicating that our model has sufficient expressive power to learn a variety of general ILP problems. In contrast to ∂ILP (Evans & Grefenstette (2018)), a principle ILP method which uses different program templates to solve the problems, our approach uses the same generative definition for all problems. With the same model architecture, our model is able to achieve high accuracy on the CLUTRR datasets. Our model is also able to learn valuable logic rules and make fairly accurate predictions on much noisier knowledge graphs.

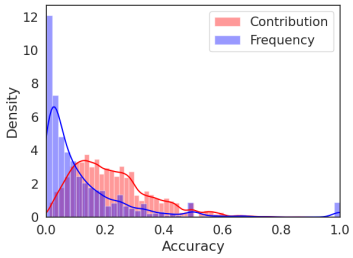

Figure 2: Distribution of rules.

2. Performance w.r.t. rule complexity. We conduct experiments to study the model performance under different rule complexity on UMLS dataset in Tab. 4. We can see that if we force the rules to be too simple, it's hard to capture informative patterns of data; On the other hand, if we force the rules to be too complex, the performance is also decreased due to the loss of rule generality.

3. Performance w.r.t. reweighting techniques. We study the effects of reweighting training data in Tab. 5. We train models on UML dataset with different reweighting methods. We can see that even without reweighting, our model is able to capture various logical patterns by merely randomly initializing model parameters. Besides, replacing, i.e., removing data instances that are correctly predicted, achieved worst results. This is because for noisy data it's better to learn more different logical patterns to prove the targets, and removing them prevents the model to learn more information about them.

4. Effects of fine-tuning rule weights. We find on most situations the model is able to make fairly precise predictions without training rule weights as in Tab. 6. We set the weights of each rule to be its precision on training data, and conduct comparison experiments based on that.

Table 2: Results on Kinship and UMLS.

Method	Kinship					UMLS				
	MR	**MRR**	**H@1**	**H@3**	**H@10**	**MR**	**MRR**	**H@1**	**H@3**	**H@10**
MLN	10.0	0.351	0.189	0.408	0.707	7.6	0.688	0.587	0.755	0.869
PathRank	-	0.369	0.272	0.416	0.673	-	0.197	0.148	0 214	0.252
NeuralLP	16.9	0.302	0.167	0.339	0.596	10.3	0.483	0.332	0.563	0.775
DRUM	11.6	0.334	0.183	0.378	0.675	8.4	0.548	0.358	0.699	0.854
MINERVA	-	0.401	0.235	0.467	0.766	-	0.564	0.426	0.658	0.814
CTP	-	0.335	0.177	0.376	0.703	-	0.404	0.288	0.430	0.674
RNNLogic (w/o emb.)	3.9	0.639	0.495	0.731	0.924	5.3	0.745	0.630	0.833	0.924
DLP	**3.7**	**0.645**	**0.504**	**0.733**	**0.927**	**3.1**	**0.810**	**0.708**	**0.896**	**0.959**

Table 3: Results on FB15k-237 and WN18RR.

Method	FB15k-237					WN18RR				
	MR	**MRR**	**H@1**	**H@3**	**H@10**	**MR**	**MRR**	**H@1**	**H@3**	**H@10**
PathRank	-	0.087	0.074	0.092	0.112	-	0.189	0.171	0.200	0.225
NeuralLP	-	0.237	0.173	0.259	0.361	-	0.381	0.368	0.386	0.408
DRUM	-	0.238	0.174	0.261	0.364	-	0.382	0.369	0.388	0.410
NLIL	-	0.25	-	-	0.324	-	-	-	-	-
M-Walk	-	0.232	0.165	0.243	-	-	0.437	0.414	0.445	-
RNNLogic (w/o emb.)	538	0.288	0.208	0.315	0.445	7527	0.455	0.414	0.475	0.531
RNNLogic+ (w/o emb.)	480	**0.299**	**0.215**	**0.328**	**0.464**	7204	0.489	0.453	0.506	**0.563**
DLP	**432**	0.285	0.208	0.310	0.436	**7190**	**0.501**	**0.472**	**0.514**	0.556

5. Effects of hyperparameters α and β. To show that in most situations we can safely set α and β to a relative small value, we conduct experiments on UMLS dataset with different settings of α and β, summarized in Tab. 7. Suprisingly, the performance under different α and β greater or equal than 1.0 is quite similar. This implies that on most situations we can safely set α and β to 1.0 to both simplify computation and stabilize the learning of model.

6. Distribution of rules accuracy. We summarize the distributions of learnt rule accuracy and average rule contributions for UMLS in Fig. 2. The average contribution of a rule equals to the average decrement of scores of queries in test data if we remove the rule and hence it measures how important a rule is on average for predicting the correct answers. We can see that most learnt rule are rather inaccurate, as there are barely any rules having precision higher than 0.5, but putting them together, rules of accuracy $0 - 0.4$ contribute the most for proving the target queries.

Table 4: Study of rule complexity.

Rule Length	2	3	4
MRR	0.682	0.810	0.786

Table 5: Study of reweighting.

Methods	Reweight	Replace	None
MRR	0.810	0.584	0.733

Table 6: Study of fine-tuning.

Fine-tune	Y	N
MRR	0.810	0.802

Table 7: Study of α and β.

α and β	0.5	1.0	2.0	3.0
MRR	0.687	0.810	0.809	0.802

6 Conclusion

This paper studies inductive logic programming, and we propose Differentiable Logic Programming framework to solve the problems of structure learning and weights learning. We generalize the discrete rule search problem and forward chaining algorithm in a continuous and probabilistic manner and use a differentiable program with a proxy problem to solve the learning problem. Both theoretical and empirical evidences are present to prove the efficiency of our algorithm. In the future, we plan to explore the possibility of combining neural network architecture to help the model discovery more complex and accurate logical patterns.

REFERENCES

Ivana Balazevic, Carl Allen, and Timothy M. Hospedales. Tucker: Tensor factorization for knowledge graph completion. *ArXiv*, abs/1901.09590, 2019.

Pablo Barceló, Egor V. Kostylev, Mikaël Monet, Jorge Pérez, Juan L. Reutter, and Juan Pablo Silva. The logical expressiveness of graph neural networks. In *ICLR*, 2020.

Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, 2013.

Liwei Cai and William Yang Wang. Kbgan: Adversarial learning for knowledge graph embeddings. *ArXiv*, abs/1711.04071, 2018.

Wenhu Chen, Wenhan Xiong, Xifeng Yan, and William Yang Wang. Variational knowledge graph reasoning. In *NAACL*, 2018.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *EMNLP*, 2014.

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. *ArXiv*, abs/1711.05851, 2018.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *AAAI*, 2018.

Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. Neural logic machines. *ArXiv*, abs/1904.11694, 2019.

Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *ArXiv*, abs/1711.04574, 2018.

Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. Fast rule mining in ontological knowledge bases with amie+. *The VLDB Journal*, 24:707–730, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

Yoon Kim. Convolutional neural networks for sentence classification. In *EMNLP*, 2014.

Yoon Kim, Yacine Jernite, David A. Sontag, and Alexander M. Rush. Character-aware neural language models. In *AAAI*, 2016.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ArXiv*, abs/1609.02907, 2017.

Stanley Kok and Pedro M. Domingos. Statistical predicate invention. In *ICML '07*, 2007.

N. Lao and William W. Cohen. Relational retrieval using a combination of path-constrained random walks. *Machine Learning*, 81:53–67, 2010.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. Multi-hop knowledge graph reasoning with reward shaping. In *EMNLP*, 2018.

Shengyao Lu, Bang Liu, Keith G. Mills, Shangling Jui, and Di Niu. R5: Rule discovery with reinforced and recurrent relational reasoning. *ArXiv*, abs/2205.06454, 2022.

Pasquale Minervini, Matko Bovsnjak, Tim Rocktäschel, Sebastian Riedel, and Edward Grefenstette. Differentiable reasoning on large knowledge bases and natural language. *ArXiv*, abs/1912.10824, 2020a.

Pasquale Minervini, Sebastian Riedel, Pontus Stenetorp, Edward Grefenstette, and Tim Rocktäschel. Learning reasoning strategies in end-to-end differentiable proving. In *ICML*, 2020b.

Pasquale Minervini, Sebastian Riedel, Pontus Stenetorp, Edward Grefenstette, and Tim Rocktäschel. Learning reasoning strategies in end-to-end differentiable proving. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pp. 6938–6949. PMLR, 2020c.

Stephen Muggleton. Inverse entailment and progol. *New Generation Computing*, 13:245–286, 2009.

Maximilian Nickel, Lorenzo Rosasco, and Tomaso A. Poggio. Holographic embeddings of knowledge graphs. *ArXiv*, abs/1510.04935, 2016.

Pouya Ghiasnezhad Omran, Kewen Wang, and Zhe Wang. Scalable rule learning via learning representation. In *IJCAI*, 2018.

Meng Qu, Junkun Chen, Louis-Pascal Xhonneux, Yoshua Bengio, and Jian Tang. Rnnlogic: Learning logic rules for reasoning on knowledge graphs. *ArXiv*, abs/2010.04029, 2021.

J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 2004.

Matthew Richardson and Pedro M. Domingos. Markov logic networks. *Machine Learning*, 62: 107–136, 2006.

Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 3791–3803, 2017. URL `http://papers.nips.cc/paper/6969-end-to-end-differentiable-proving`.

Ali Sadeghian, Mohammadreza Armandpour, Patrick Ding, and Daisy Zhe Wang. Drum: End-to-end differentiable rule mining on knowledge graphs. In *NeurIPS*, 2019.

Peter Schüller and Mishal Kazmi. Best-effort inductive logic programming via fine-grained cost-based hypothesis generation. *Machine Learning*, 107:1141–1169, 2018.

Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Trans. Signal Process.*, 45:2673–2681, 1997.

Yelong Shen, Jianshu Chen, Po-Sen Huang, Yuqing Guo, and Jianfeng Gao. M-walk: Learning to walk over graphs using monte carlo tree search. In *NeurIPS*, 2018.

Koustuv Sinha, Shagun Sodhani, Jin Dong, Joelle Pineau, and William L. Hamilton. Clutrr: A diagnostic benchmark for inductive reasoning from text. In *EMNLP*, 2019.

Zhiqing Sun, Zhihong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. *ArXiv*, abs/1902.10197, 2019.

Komal K. Teru, E. Denis, and William L. Hamilton. Inductive relation prediction by subgraph reasoning. In *ICML*, 2020.

Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, 2015.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *ICML*, 2016.

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

Ramakrishna Vedantam, Karan Desai, Stefan Lee, Marcus Rohrbach, Dhruv Batra, and Devi Parikh. Probabilistic neural-symbolic models for interpretable visual question answering. In *ICML*, 2019.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio', and Yoshua Bengio. Graph attention networks. *ArXiv*, abs/1710.10903, 2018.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, 2014.

Wenhan Xiong, Thi-Lan-Giao Hoang, and William Yang Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *EMNLP*, 2017.

Bishan Yang, Wen tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *CoRR*, abs/1412.6575, 2015.

Fan Yang, Zhilin Yang, and William W. Cohen. Differentiable learning of logical rules for knowledge base completion. *ArXiv*, abs/1702.08367, 2017.

Yuan Yang and Le Song. Learn to explain efficiently via neural logic inductive learning. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=SJlh8CEYDB`.

Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. In *NeurIPS*, 2021.

A PROOF OF PROPOSITION 4.2

We now prove Prop. 4.2.

Proposition A.1 (Properties of Φ). *Given the input data $\mathcal{B}$, $\Phi_Q = \{\varphi_1, \varphi_2, ..., \varphi_L\}$ and $p \in (0, 1]$, if there is a permutation of elements of Φ_Q as $\varphi_{i1}, \varphi_{i2}, ...$ that satisfies:*

- *For φ_{i1}, its precision on $\mathcal{B}$: $p_{\mathcal{B}}(Q(\mathbf{x})|\varphi_{i1}(\mathbf{x}) = 1) = \frac{\sum_{\mathbf{x}} Q(\mathbf{x})\varphi_{i1}(\mathbf{x})}{\sum_{\mathbf{x}} \varphi_{i1}(\mathbf{x})} \geq p$;*
- *For φ_{ij}, we remove all queries $Q(\mathbf{x}) = 1$ and also proved true by previous $\varphi_{i1}, \varphi_{i2}, ..., \varphi_{i(j-1)}$, and on the remaining data, we still have $p_{\mathcal{B}}(Q(\mathbf{x})|\varphi_{ij}(\mathbf{x}) = 1) \geq p$;*
- *There exists k such that for φ_{ik}, there is no true statements $Q(\mathbf{x}) = 1$ remaining.*

Then, we can construct a prediction model $p_w(Q(\mathbf{x})|\Phi_Q(\mathbf{x}))$ such that its error rate on $\mathcal{B}$ satisfies:

$$\text{Err}[p_w; \mathcal{B}] \leq \frac{(1-p)N_Q}{pN} \leq 1 - p, \tag{12}$$

where $N_Q = \sum_{\mathbf{x}} Q(\mathbf{x})$ and $N = \sum_{\mathbf{x}} 1$.

Proof: The proof is constructive. We let $p_w\left(Q(\mathbf{x}) = 1|\Phi(\mathbf{x})\right) = \min\left\{1, \sum_{\varphi\in\Phi} \varphi(\mathbf{x})\right\}$, and let N_{φ_i} be the number of $Q(\mathbf{x})$s that are true and also predicted true by φ_i, with those $Q(\mathbf{x})$s related to $\varphi_1, \varphi_2, ..., \varphi_{i-1}$ removed first. It's easy to observe that the number of $Q(\mathbf{x})$s predicted true by φ_i but are actually false is $W_{\varphi_i} \leq \frac{1-p}{p} N_{\varphi_i}$. Thus, we have:

$$\begin{aligned}
\text{Err}\left[p_w; \mathcal{G}\right] &= \frac{\sum_{\mathbf{x}} \mathbf{1}_{Q(\mathbf{x}) \neq p_w(Q(\mathbf{x}))}}{N} \\
&= \frac{1}{N_{\mathcal{G}}} \sum_{\mathbf{x}} (1 - Q(\mathbf{x})) \min\left\{1, \sum_{\varphi\in\Phi} \varphi(\mathbf{x})\right\} \\
&\leq \frac{1}{N} \sum_i W_{\varphi_i} \\
&\leq \frac{1}{N} \sum_i \frac{1-p}{p} N_{\varphi_i} \\
&= \frac{(1-p)N_Q}{pN} \leq 1 - p.
\end{aligned} \tag{13}$$

B PROOF OF PROPOSITION 4.3

We now prove Prop. 4.3.

Proposition B.1 (Proxy problem). *Consider the optimization problem*

$$\min_{\psi} \quad \mathcal{L}(\psi) = -\log \mathbb{E}\left[\psi(\mathbf{x}; \alpha)|Q(\mathbf{x}) = 1\right] + \log \mathbb{E}\left[\psi(\mathbf{x}; \beta)|Q(\mathbf{x}) \neq 1\right], \tag{14}$$

where $\psi(\mathbf{x}; \alpha)$ and $\psi(\mathbf{x}; \beta)$ are obtained by one iteration of the forward procedure. With the following constraints being satisfied: (1) $\alpha > \beta > 1$; (2) The unfixed nodes in the paths for computing ψ compose a tree; (3) Negations are applied only on fixed nodes. Then we have:
(1) At each local minima of $\mathcal{L}(\psi)$, ψ captures a logic classifier;
(2) Minimization of the proxy problem yields a near-optimal solution for problem 10.

Proof: Before we proceed to prove the proposition, we need to first study in what situations does a LP ψ capture a logic classifier φ. Across this section we assume we are given a inference network $\mathcal{G}$ composed of $\{\psi_1, \psi_2, ..., \psi_L, \psi\}$ where ψ is the output node built upon $\{\psi_1, \psi_2, ..., \psi_L\}$ we care about. The following lemma explains the properties of ψ when it captures a logic classifier.

Lemma B.2 *Given $\mathcal{G}$ and $\{\psi_1, \psi_2, ...\}$ stated above. If we perform a restricted breadth first search across the inference network where:*

- *We start with the output node ψ;*
- *For each node we go through, the parameters of the node $\boldsymbol{\theta}$ or $\boldsymbol{w}$ defined in Eq. 6-7 are a one-hot vector where one of its dimensions equals to 1;*
- *From the current node, the next paths we go are a subset of the inverse edges of ψ_i, where we only consider the ones corresponding to the nonzero dimensions of $\boldsymbol{\theta}$ or $\boldsymbol{w}$ discussed above defined in Eq. 6-7.*

Then, ψ captures a logic classifier φ, i.e., for all $\mathbf{x}$, $\psi(\mathbf{x}) > 0 \iff \varphi(\mathbf{x}) > 0$.

The lemma is quite straightforward: by restricting a LP with the above constraints, its evaluation procedure naturally simulates the grounding of a logic classifier. We can construct a logic classifier φ captured by ψ as follows: for each node we went through in the procedure of lemma B.2, we replace the LP ψ with an actual logic classifier φ which is constructed with the correspondence defined in Eq. 7. Moreover, for each one-hot vectors $\boldsymbol{\theta}$ and $\boldsymbol{w}$, we pick the predicates or sub-formulas corresponding to the nonzero dimension of these vectors. By doing so, we observe that the evaluation results of φ for any $\mathbf{x}$ are the same as ψ does because every computational steps of both φ and ψ stay the same. Thus, to prove that a ψ captures a φ, we need to show that the parameters of ψ satisfy the constraints in lemma B.2.

We now proceed to prove Prop. 4.3, which is a conclusion of a more general case stated as follows:

Proposition B.3 (General proxy problem). *Consider the optimization problem*

$$\min_{\psi} \quad \mathcal{L}'(\psi) = -\log \mathbb{E}_{\mathbf{x}\sim p}[\psi(\mathbf{x};\alpha)] + \log \mathbb{E}_{\mathbf{x}\sim q}[\psi(\mathbf{x};\beta)]. \tag{15}$$

Assume we have $\mathbb{E}_{\mathbf{x}\sim p}[\psi(\mathbf{x};\alpha)] > 0$ $\mathbb{E}_{\mathbf{x}\sim q}[\psi(\mathbf{x};\beta)] > 0$, otherwise the results are meaningless. There exists α and β as stated in Prop. 4.3 such that at each local minima of $\mathcal{L}'(\psi)$, ψ captures one logic classifier φ.

It's easy to observe that Prop. B.1 is a special case of Prop. B.3 where we let $\mathbb{E}_{\mathbf{x}\sim p}[\psi(\mathbf{x};\alpha)] = \mathbb{E}[\psi(\mathbf{x};\alpha)|Q(\mathbf{x}) = 1]$ and $\mathbb{E}_{\mathbf{x}\sim q}[\psi(\mathbf{x};\alpha)] = \mathbb{E}[\psi(\mathbf{x};\alpha)|Q(\mathbf{x}) \neq 1]$. Thus, we prove Prop. B.3 first, which directly expresses the conclusion (1) of Prop. B.1, and then discuss the collaborations (2) and (3). We prove this proposition sequential: we first prove that the parameters $\boldsymbol{\theta}$ of ψ must satisfies the proposition, i.e. it is one-hot, no matter whether $\boldsymbol{w}^{(1)}$, $\boldsymbol{w}^{(2)}$, ... are or not; Then, we prove that the parameters $\boldsymbol{w}^{(i_1)}$, $\boldsymbol{w}^{(i_2)}$, ... w.r.t. the nonzero dimension θ_i of $\boldsymbol{\theta}$ must also be one-hot; After that, we show that if ψ satisfies the proposition, the next nodes of ψ in the restricted BFS path must also satisfies the proposition; Hence the proposition is proved.

To study $\boldsymbol{\theta}$, we fix all other parameters in the inference network $\mathcal{G}$ except $\boldsymbol{\theta}$ for ψ, and the evaluation of $\psi(\mathbf{x};\alpha)$ becomes a simple polynomial function:

$$\psi(\mathbf{x};\alpha) = \sum_i \mathcal{F}_i(\mathbf{x})\theta_i^{\alpha}, \tag{16}$$

and the evaluation of $\mathcal{L}'$ becomes:

$$\begin{aligned}
\mathcal{L}'(\psi) &= -\log \mathbb{E}_{\mathbf{x}\sim p}[\psi(\mathbf{x};\alpha)] + \log \mathbb{E}_{\mathbf{x}\sim q}[\psi(\mathbf{x};\beta)] \\
&= -\log \sum_{\mathbf{x}} \psi(\mathbf{x};\alpha)p(\mathbf{x}) + \log \sum_{\mathbf{x}} \psi(\mathbf{x};\beta)q(\mathbf{x}) \\
&= -\log \left\{ \left(\sum_{\mathbf{x}} \sum_i p(\mathbf{x})\mathcal{F}_i(\mathbf{x};\alpha) \right) \theta_i^{\alpha} \right\} + \log \left\{ \left(\sum_{\mathbf{x}} \sum_i q(\mathbf{x})\mathcal{F}_i(\mathbf{x};\beta) \right) \theta_i^{\beta} \right\}.
\end{aligned} \tag{17}$$

Hence, $\mathcal{L}'(\psi)$ is a polynomial function w.r.t. $\boldsymbol{\theta}$ and we need to show that at each local minima of $\mathcal{L}'(\psi)$ there cannot exist two θ_i and θ_j that are all larger than 0, which will be discussed later in this section. Now, assume we have already know that $\boldsymbol{\theta}$ of ψ satisfies the proposition, i.e., $\theta_i = 1$ for some i. To proceed, we need to prove that all parameters $\boldsymbol{w}^{(i_1)}$, $\boldsymbol{w}^{(i_2)}$, ... w.r.t. $\mathcal{F}_i$ are also one-hot.

We do this similarly as before: we fix all other parameters in the inference network except some $\boldsymbol{w}^{(i_k)}$ emerged in a neighbor of ψ in the inference network. By carefully checking the construction steps of $\mathcal{F}_i$ in Eq. 7, it's easy to show that $\mathcal{L}'(\psi)$ is also a polynomial function w.r.t. $\boldsymbol{w}^{(i_k)}$ under the constraints that $\boldsymbol{\theta}$ is one-hot. The following theorem directly proves that these functions indeed satisfy the constraints.

Theorem B.4 (Attention Shrinkage.) *The following function*

$$h(\boldsymbol{w}, \alpha, \beta) = \frac{\sum_i A_i(\alpha) w_i^\alpha}{\sum_i B_i(\beta) w_i^\beta} \tag{18}$$

has no local maxima w.r.t. $w_i \in (0, 1)$ *with the following constraints being satisfied:*
(1) Attention vector: $w_i \geq 0$ *for each dimension* i *and* $\sum_i w_i = 1$*;*
(2) Positive coefficients: $A_i(\alpha) \geq 0$ *and* $B_i(\beta) \geq 0$ *for every* i *and all* $\alpha, \beta > 0$*;*
(3) Non-empty results: $\sum_i A_i(\alpha) w_i^\alpha > 0$ *and* $\sum_i B_i(\beta) w_i^\beta > 0$*;*
(4) $\alpha > \beta > 1$.

Proof of theorem B.4: We now prove that $h(\boldsymbol{w}, \alpha, \beta)$ has no local maxima where each $w_i \in (0, 1)$ by contradiction. Assume we are now given some $\boldsymbol{w}_0$ where there are at least two dimensions of $\boldsymbol{w}_0$ that are nonzero, and the target is to prove that $\boldsymbol{w}_0$ is not a local maxima of h. To do so, we create a new vector $\boldsymbol{v}_0$ composed of nonzero dimensions of $\boldsymbol{w}_0$ together with a function $h'(\boldsymbol{v}, \alpha, \beta)$, and

$$h(\boldsymbol{w}, \alpha, \beta) = h'(\boldsymbol{v}, \alpha, \beta) = \frac{\sum_i C_i(\alpha) v_i^\alpha}{\sum_i D_i(\beta) v_i^\beta} = \frac{f(\boldsymbol{v}, \alpha)}{g(\boldsymbol{v}, \beta)}. \tag{19}$$

To prove that $h(\boldsymbol{w}, \alpha, \beta)$ is not at local maxima, we can instead show that $\boldsymbol{v}_0$ is not a local maxima of $h'(\boldsymbol{v}, \alpha, \beta)$ is not at local maxima. We first consider the situations where the partial derivatives of h' w.r.t. each v_i are not all the same, for example, suppose $\frac{\partial h'}{\partial v_1}|_{\boldsymbol{v}=\boldsymbol{v}_0} > \frac{\partial h'}{\partial v_2}|_{\boldsymbol{v}=\boldsymbol{v}_0}$. Directly study all dimensions of $\boldsymbol{v}$ is intractable with constraint $\sum_i v_i = 1$, so we instead fix all parameters $v_3, v_4, ...$ at the corresponding value of $\boldsymbol{v}_0$ and only study the two parameters v_1 and v_2 where we let $v_1 = v_{01} + x$, $v_2 = v_{02} - x$ and so the above constraint is naturally satisfied. Thus, we have $h'(\boldsymbol{v}, \alpha, \beta) = h''(x)$ and

$$\left.\frac{dh''(x)}{dx}\right|_{x=0} = \left.\frac{\partial h'(\boldsymbol{v}, \alpha, \beta)}{\partial v_1}\right|_{\boldsymbol{v}=\boldsymbol{v}_0} - \left.\frac{\partial h'(\boldsymbol{v}, \alpha, \beta)}{\partial v_2}\right|_{\boldsymbol{v}=\boldsymbol{v}_0} > 0. \tag{20}$$

Since the derivative w.r.t. x is larger than 0, $\boldsymbol{v}_0$ is not a local maxima.

Next, we discuss the situation where $\frac{\partial h'}{\partial v_i}|_{\boldsymbol{v}=\boldsymbol{v}_0} = \lambda$ are all the same. We again let $v_1 = v_{01} + x$, $v_2 = v_{02} - x$, and fix all other parameters of $\boldsymbol{v}$. We have

$$\begin{aligned}
\left.\frac{d^2 h''(x)}{dx^2}\right|_{x=0} = \frac{1}{g^3} \Bigg\{ & g^2 \alpha(\alpha - 1) \left(A_1 v_1^{\alpha-2} + A_2 v_2^{\alpha-2} \right) \\
& - 2g\alpha\beta(A_1 v_1^{\alpha-1} + A_2 v_2^{\alpha-1})(B_1 v_1^{\beta-1} + B_2 v_2^{\beta-2}) \\
& + 2f\beta^2 \left(B_1 v_1^{\beta-1} + B_2 v_2^{\beta-2} \right)^2 \\
& - gf\beta(\beta - 1) \left(B_1 v_1^{\beta-2} + B_2 v_2^{\beta-2} \right) \Bigg\}.
\end{aligned} \tag{21}$$

Since we assume $\frac{\partial h'}{\partial v_i}|_{\boldsymbol{v}=\boldsymbol{v}_0} = \lambda$, we have

$$\begin{aligned}
& \left.\frac{\partial h'}{\partial v_i}\right|_{\boldsymbol{v}=\boldsymbol{v}_0} = \frac{1}{g^2} \left(\alpha g A_i v_i^{\alpha-1} - \beta f B_i v_i^{\beta-1} \right) = \lambda, \\
\implies & \frac{1}{g^2} \left(\alpha g A_i v_i^{\alpha} - \beta f B_i v_i^{\beta} \right) = v_i \lambda, \\
\implies & \sum_i \frac{1}{g^2} \left(\alpha g A_i v_i^{\alpha} - \beta f B_i v_i^{\beta} \right) = \sum_i v_i \lambda, \\
\implies & \frac{f}{g}(\alpha - \beta) = \lambda.
\end{aligned} \tag{22}$$

Substituting Eq. 22 into Eq. 21, we have

$$\begin{aligned}
\left.\frac{d^2h''(x)}{dx^2}\right|_{x=0} &= \frac{f}{g}\left(\frac{1}{f}A_1\alpha(\alpha-1)v_1^{\alpha-2} - \frac{1}{g}B_1\beta(\beta-1)v_1^{\beta-2}\right) \\
&\quad + \frac{f}{g}\left(\frac{1}{f}A_2\alpha(\alpha-1)v_2^{\alpha-2} - \frac{1}{g}B_2\beta(\beta-1)v_2^{\beta-2}\right) \\
&\geq \frac{f}{gv_1}\left(\frac{1}{f}A_1\alpha v_1^{\alpha-1} - \frac{1}{g}B_1\beta v_1^{\beta-1}\right) + \frac{f}{gv_2}\left(\frac{1}{f}A_2\alpha v_2^{\alpha-1} - \frac{1}{g}B_2\beta v_2^{\beta-1}\right) \\
&= \frac{\lambda}{v_1} + \frac{\lambda}{v_2} \\
&= \frac{f}{g}(\alpha-\beta)\left(\frac{1}{v_1} + \frac{1}{v_2}\right) > 0.
\end{aligned} \tag{23}$$

So $\boldsymbol{v}_0$ is not a local maxima of h'. Thus, we have proved that $h(\boldsymbol{w}, \alpha, \beta)$ has no local maxima for $w_i \in (0, 1)$.

End of proof of theorem B.4.

Since f and $\log f$ share the same minimum points for any $f > 0$ we have shown that the output node ψ indeed satisfies the conditions in the proposition. It's straightforward to show the nodes along the paths that ψ is built on also satisfy the conditions. Suppose we are studying another ψ' that is used for computing ψ. By writing the detailed computation steps for evaluating ψ and fixing all irrelevant parameters, we can show that $\psi(\mathbf{x}; \alpha) = \sum_{\mathbf{x}'} A(\mathbf{x}', \alpha)\psi(\mathbf{x}'; \alpha)$ and thus

$$\mathbb{E}_{x\sim p}[\psi(\mathbf{x}; \alpha)] = \mathbb{E}_{x'\sim p'}[\psi(\mathbf{x}'; \alpha)], \tag{24}$$

where p' is an unnormalized distribution. Thus, the same conclusion holds for all ψ in the restricted BFS path.

We now have proved that every local minima of the proxy problem corresponds to a logic classifier, and it's much easier to prove the rest of the conclusions as collaborations of the first one. For conclusion (2), we notice that when ψ converges,

$$\begin{aligned}
p(Q(\mathbf{x})|\varphi(\mathbf{x}) = 1) &= \frac{p(\varphi(\mathbf{x}) = 1|Q(\mathbf{x}) = 1)}{p(\varphi(\mathbf{x}) = 1)} p(Q(\mathbf{x}) = 1) \\
&= \frac{\mathbb{E}[\varphi(\mathbf{x})|Q(\mathbf{x}) = 1]}{\mathbb{E}[\varphi(\mathbf{x})]} \mathbb{E}[Q(\mathbf{x})] \\
&\approx \frac{\mathbb{E}[\psi(\mathbf{x}; \alpha)|Q(\mathbf{x}) = 1)}{\mathbb{E}[\psi(\mathbf{x}; \beta)]} \frac{N_Q}{N} \\
&= \frac{\mathbb{E}[\psi(\mathbf{x}; \alpha)|Q(\mathbf{x}) = 1)}{\mathbb{E}[\psi(\mathbf{x}; \beta)|Q(\mathbf{x}) \neq 1] + \mathbb{E}[\psi(\mathbf{x}; \beta)|Q(\mathbf{x}) = 1]} \frac{N_Q}{N} \\
&= \frac{\mathbb{E}[\psi(\mathbf{x}; \alpha)|Q(\mathbf{x}) = 1)}{\mathbb{E}[\psi(\mathbf{x}; \beta)|Q(\mathbf{x}) \neq 1] + \mathbb{E}[\psi(\mathbf{x}; \alpha)|Q(\mathbf{x}) = 1]} \frac{N_Q}{N} \\
&= \frac{1}{1 + \frac{\mathbb{E}[\psi(\mathbf{x};\beta)|Q(\mathbf{x})\neq 1]}{\mathbb{E}[\psi(\mathbf{x};\alpha)|Q(\mathbf{x})=1]}} \frac{N_Q}{N} \\
&= \frac{N_Q}{(1 + \exp \mathcal{L}(\psi))N},
\end{aligned} \tag{25}$$

which is monotone decreasing w.r.t. $\mathcal{L}(\psi)$ so conclusion (2) holds. Here, we assume when ψ converges, $\mathbb{E}[\psi(\mathbf{x}; \alpha)|\psi(\mathbf{x}; \alpha) > 0] \approx \mathbb{E}_{\mathbf{x}\in Pos}[\psi(\mathbf{x}; \alpha)|\psi(\mathbf{x}; \alpha) > 0]$.

C Discussion of Constraints in Proxy Problem

In this section we discuss the properties and functionalities of the three constraints in the proxy problem as well as constructing example to illustrate how they work.

Proxy Problem

$$\min_{\psi} \quad \mathcal{L}(\psi) = -\log \mathbb{E}_{\mathbf{x}\sim p}\left[\psi(\mathbf{x};\alpha)\right] + \log \mathbb{E}_{\mathbf{x}\sim q}\left[\psi(\mathbf{x};\beta)\right], \tag{26}$$

(1) α and β are sufficiently large;
(2) The unfixed nodes in the paths for computing ψ compose a tree;
(3) Negations are applied only on fixed nodes.

Properties of α and β. We first discuss the properties of hyperparameters α and β. As show in appendix B, $\alpha > \beta$ is necessary to keep the second-order derivative positive. If we remove this constraints by simply setting $\alpha = \beta = 1$, then we observe that Eq. 22 actually becomes

$$\left.\frac{\partial h'}{\partial v_i}\right|_{\boldsymbol{v}=\boldsymbol{v}_0} = \frac{f}{g}(\alpha - \beta) = 0, \tag{27}$$

and we have the second-order derivative

$$\left.\frac{d^2 h''}{dx^2}\right|_{x=0} = \frac{f}{g}(\alpha - \beta)(\frac{1}{v_1} + \frac{1}{v_2}) = 0. \tag{28}$$

This means that while training the model, it is possible that the model's derivatives w.r.t. multiple nonzero dimensions w_i of some $\boldsymbol{w}$ become 0 and the model might falls into local minima where it doesn't capture any logic classifier. We now discuss the situations where these risks actually exist.

The first and most simple case is when $p = q$, i.e.,

$$\mathcal{L}(\psi) = -\log \mathbb{E}_{\mathbf{x}\sim p}\left[\psi(\mathbf{x};\alpha)\right] + \log \mathbb{E}_{\mathbf{x}\sim p}\left[\psi(\mathbf{x};\alpha)\right] = 0. \tag{29}$$

Apparently in this case, training the model provides nothing because $\mathcal{L}(\psi)$ is a fixed scalar irrelevant to ψ. We argue that this is not a big problem since it requires p and q, corresponding to the distributions of positive and negative data instances, to be the same.

By extending the above case, we can construct a more general situation. Recall that in theorem B.4, we have

$$h(\boldsymbol{w}, 1, 1) = \frac{\sum_i A_i(1) w_i}{\sum_i B_i(1) w_i} \leq \max_i \left\{\frac{A_i(1)}{B_i(1)}\right\} = \frac{A_m(1)}{B_m(1)}, \tag{30}$$

where we let m be the dimension corresponding to the global maxima. If there are some dimension k of $\boldsymbol{w}$ such that $A_k(1) = B_k(1) = 0$, then for any $\boldsymbol{w}'$ satisfying $w'_m > 0$ and $w'_k = 1 - w'_m$, we have $h(\boldsymbol{w}', 1, 1) = \frac{A_m(1)}{B_m(1)} = \sup\{h(\boldsymbol{w}, 1, 1)\}$. To solve this problem, we can add a normalization term to the original proxy problem, i.e.

$$\begin{aligned}\mathcal{L}_{\text{Aug}}(\psi) &= \mathcal{L}(\psi) + \mathcal{L}_{\text{Norm}}(\psi) \\ &= -\log \mathbb{E}_{\mathbf{x}\sim p}\left[\psi(\mathbf{x};1)\right] + \log \mathbb{E}_{\mathbf{x}\sim q}\left[\psi(\mathbf{x};1)\right] - \lambda \log \mathbb{E}_{\mathbf{x}\sim p}\left[\psi(\mathbf{x};1)\right].\end{aligned} \tag{31}$$

Thus, even when $A_k(1) = B_k(1) = 0$, the normalization term still encourages the model to assign larger value to w_m.

Another problematic situation still happens when there exists $i \neq j$ such that $A_i(1) = A_j(1) > 0$ and $B_i(1) = B_j(1) > 0$. This is not usual because it requires there exists two different logic classifiers φ_1 and φ_2 to have $\mathbb{E}_{\mathbf{x}\sim p}[\varphi_1(\mathbf{x})] = \mathbb{E}_{\mathbf{x}\sim p}[\varphi_2(\mathbf{x})]$ and $\mathbb{E}_{\mathbf{x}\sim q}[\varphi_1(\mathbf{x})] = \mathbb{E}_{\mathbf{x}\sim q}[\varphi_2(\mathbf{x})]$. Even when it happens, it's easy to deal with the problem, as during training we always reweight or sample different batches of training data, yielding a different data distribution p' and q', We can also set $\alpha = \beta > 1$, for example, $\alpha = \beta = 2$, to avoid such situations from happening.

Tree structured paths constraints. We now discuss the second constraint in the problem. If we remove this constraints and there are circles for some unfixed nodes, we observe that $\mathcal{L}(\psi)$ w.r.t. the corresponding $\boldsymbol{w}$ might no more holds the form of Eq. 18. Note that in Eq. 7 we define the conjunction as $\mathcal{F}_i(\mathbf{x}) = \mathcal{F}_j(\mathbf{x})\mathcal{F}_k(\mathbf{x})$. If the evaluation of $\mathcal{F}_j$ and $\mathcal{F}_k$ both contains some $\boldsymbol{w}$, i.e. $\mathcal{F}_j = \boldsymbol{a}_j^T(\boldsymbol{w})^\alpha$ and $\mathcal{F}_k = \boldsymbol{a}_k^T(\boldsymbol{w})^\alpha$ where $\boldsymbol{a}_j$ and $\boldsymbol{a}_k$ are irrelevant quantities with $\boldsymbol{w}$, then $\mathcal{F}_j = \boldsymbol{a}_j^T(\boldsymbol{w}\boldsymbol{w}^T)^\alpha \boldsymbol{a}_k$ which no more holds the form of Eq. 18. Instead, we have

$$h(\boldsymbol{w}, \alpha, \beta) = \frac{\sum_i A_i(\alpha) \prod_j w_j^{n_j \alpha}}{\sum_i A_i(\alpha) \prod_j w_j^{n_j \alpha}}. \tag{32}$$

This form of $h(\boldsymbol{w}, \alpha, \beta)$ no longer keeps the good properties of Eq. 18, as it's second order derivatives w.r.t. some dimension of $\boldsymbol{w}$ no longer guarantee to be larger or equal than 0. We provide an example to illustrate this.

Consider we want to learn logic rules of $Q(\mathbf{x}) \leftarrow \varphi_1(\mathbf{x}) := P_1(\mathbf{x}) \wedge P_1(\mathbf{x})$ and $Q(\mathbf{x}) \leftarrow \varphi_2(\mathbf{x}) := P_2(\mathbf{x}) \wedge P_2(\mathbf{x})$. This expression is redundant because the two terms of the conjunction are the same. Suppose we the model we used here is $\psi_1(\mathbf{x}; \alpha) = \psi_2(\mathbf{x}; \alpha)\psi_2(\mathbf{x}; \alpha)$ where $\psi_2(\mathbf{x}; \alpha) = \sum_i w_i^\alpha P_i(\mathbf{x})$ is a soft selection over all possible predicates. Unluckily, in the training data the logic rule $Q(\mathbf{x}) \leftarrow \varphi_1(\mathbf{x})$ and $Q(\mathbf{x}) \leftarrow \varphi_2(\mathbf{x})$ are never satisfied, i.e., $p(Q(\mathbf{x})|\varphi_1(\mathbf{x})) = p(Q(\mathbf{x})|\varphi_2(\mathbf{x})) = 0$, but for the rule $Q(\mathbf{x}) \leftarrow \varphi_3(\mathbf{x}) := P_1(\mathbf{x}) \wedge P_2(\mathbf{x})$ we have $p(Q(\mathbf{x})|\varphi_3(\mathbf{x})) > 0$. Then, it's easy to observe that the global minima of the proxy problem happens when $w_1 = w_2 = 0.5$.

Although this situation disobeys with the conclusions of the proxy problem, both the training data, the target logic rules and construction of LPs are ill-conditioned, as we placed the same ψ_2 between the conjunction as well as in the training data we are unable to find any high-precision logic rules. We argue that in reality this can often be avoided because we can increase the expressiveness of the model, and set α and β to a relative large value so that even the above situation happens, because $w_1^\alpha = w_2^\alpha$ are rather small values, it provides little stimulation to the model, and thus the model is encouraged to choose other rule structures that better explains the data. Also, because during training when the model is not at convergence, dimensions of $\boldsymbol{w}$ are rather small values, and the higher order terms $\prod_j w_j^{n_j \alpha}$ where $\sum_j n_j$ is large are much smaller than oridinary terms, which means they provide little influence to the overall derivatives of $\boldsymbol{w}$, so this problem is less serious.

Negations. When training completes and model converges, negations do not influence the validity of the model. However, during training, negations might be tricky to deal with, as adding negations to the training LPs might lead to the coefficients of corresponding w_i in $h(\mathbf{x}, \alpha, \beta)$ being negative. The following example illustrated how negations might influence the training process.

Consider we want to learn the logic rules $Q(\mathbf{x}) \leftarrow \varphi_1(\mathbf{x}) := \neg P_1(\mathbf{x})$ and $Q(\mathbf{x}) \leftarrow \varphi_2(\mathbf{x}) := \neg P_2(\mathbf{x})$, and we construct a model of $\psi_1(\mathbf{x}; \alpha) = 1 - \psi_2(\mathbf{x}; \alpha)$ where $\psi_2(\mathbf{x}; \alpha) = \sum_i w_i^\alpha P_i(\mathbf{x})$. Suppose $\mathbb{E}[\varphi_1(\mathbf{x})|Q(\mathbf{x}) = 1] = \mathbb{E}[\varphi_2(\mathbf{x})|Q(\mathbf{x}) = 1] = E_1$, $\mathbb{E}[\varphi_1(\mathbf{x})|Q(\mathbf{x}) \neq 1] = \mathbb{E}[\varphi_2(\mathbf{x})|Q(\mathbf{x}) \neq 1] = E_2$ Then, by letting $\psi_2(\mathbf{x}; \alpha) = 0.5^\alpha P_1(\mathbf{x}) + 0.5^\alpha P_2(\mathbf{x})$, we observe that

$$\begin{aligned}
\mathcal{L}(\psi) &= -\log \mathbb{E}\left[\psi(\mathbf{x}; \alpha)|Q(\mathbf{x}) = 1\right] + \log \mathbb{E}\left[\psi(\mathbf{x}; \beta)|Q(\mathbf{x}) \neq 1\right] \\
&= -\log \left(0.5^\alpha \left(\mathbb{E}[\varphi_1(\mathbf{x})|Q(\mathbf{x}) = 1] + \mathbb{E}[\varphi_2(\mathbf{x})|Q(\mathbf{x}) = 1]\right)\right) \\
&\quad + \log \left(0.5^\beta \left(\mathbb{E}[\varphi_1(\mathbf{x})|Q(\mathbf{x}) \neq 1] + \mathbb{E}[\varphi_2(\mathbf{x})|Q(\mathbf{x}) \neq 1]\right)\right) \\
&= -\alpha + \beta + \text{constant}.
\end{aligned} \tag{33}$$

Thus, we can see $\mathcal{L}(\psi)$ is monotonic decreasing w.r.t. $\alpha - \beta$, and such ψ_2 can reach a smaller value than φ_1 and φ_2 once $\alpha - \beta$ is sufficiently large. We can set $\alpha = \beta$ to a relative small value and carefully select the target LPs when trying to assign negations to them to avoid the problem from happening.

So far we have discussed the situations when the model might fail to converge if we discard the constraints of the proxy problem. As we can see, most of these invalidate situations can be avoided by setting $\alpha = \beta = 1$ or 2 as well as providing a reasonable generative definition of target logic rules. We argue that even if sometimes the model fails to converge at capturing a logic classifier, we can reinitialize the parameters of the relevant LPs randomly and train the model again so that it converges at other minimum points.

D Experiment Details

In this section we explain the detailed model configuration for each experiment.

D.1 ILP Tasks

The 20 ILP tasks introduced by Evans & Grefenstette (2018) cover problems from integer recognition, family tree reasoning, general graph algorithms and so on. We briefly summarize them here.

Task 1-6 In task 1 to task 6 we are provided with natural numbers from 0 to 9, which are defined as follows:

$$\mathcal{B} = \{zero(0), succ(0,1), ..., succ(8,9)\}. \tag{34}$$

The target is to learn to recognize predecessor, even / odd numbers, the less-than relation and divisible by 3 or 5.

Task 7-8 Task 7-8 requires us to learn the relation member and length of a list. Nodes in a list is encoded as follows: $cons(x, y)$ if the node after x is y, and $value(x, y)$ if the value of node x is y. Two background statements are given, corresponding to the list $[4, 3, 2, 1]$ and $[2, 3, 2, 4]$.

Task 9-14 In task 9-14 we are provided with different facts about family relations, and we need to learn the relations including son, grandparent, husband, uncle, relatedness and father. An example of rules would be $son(x, y) \leftarrow father(x, y) \wedge \varphi(x), \varphi(x) := brother(x, y) \vee father(x, y)$ where φ implies the $male$ property.

Task 15-20 In task 15-20 we are provided with labeled directed graphs, and we are asked to learn general concepts of graph algorithms. These tasks includes to learn whether a node is adjacent to a red node; whether a node has at least two children; whether a graph is well-colored, i.e., to identify if there are two adjacent nodes of the same color; whether two nodes are connected; and recognize graph cycles.

As a principled approach, ∂ILP (Evans & Grefenstette (2018)) is able to solve all these tasks. However, they need to construct different language templates and program templates for each task, for example, to solve the even numbers problem, they use the following templates:

$$\begin{aligned} P_e &: \{zero/1, succ/2\} \\ P_i &: \{target/1, pred/2\} \\ \tau_{target,1} &= (h = target, n_\exists = 0, int = False) \\ \tau_{target,2} &= (h = target, n_\exists = 1, int = True) \\ \tau_{pred,1} &= (h = pred, n_\exists = 1, int = False) \\ \tau_{pred,2} &= \text{null}. \end{aligned} \tag{35}$$

In contrast, we use a unified model to solve the tasks. Our generative definitions of logic classifiers are as follows:

$$\begin{aligned} \varphi(x) &:= P(x) \mid \varphi(x, x) \mid \exists y : \varphi(x, y) \wedge \varphi(y), \\ \varphi(x, y) &= A(x, y) \mid \exists z : \varphi(x, z) \wedge \varphi(z) \wedge \varphi(z, y). \end{aligned} \tag{36}$$

For some tasks if there are no observed predicates of arity 1 or 2, we simply create an invented predicates with all 1 values for every variables $\mathbf{x}$. On all tasks we set $\alpha = \beta = 1$. The construction of inference network of the model and how we train the model is the same as described in Sec. 4.1, Sec. 4.3 and Sec. 4.4. To be more specific, we train the model as follows: we first add a node $\psi(x)$ or $\psi(x, y)$ to the network, depending on the arity of the target predicates. If the model fails to recognize any targets queries, then we simply add more nodes to the network, until the model succeeds to deduce some of the target queries. Then, we remove theses identified queries and proceed to train the model as described in Sec. 4.1 until all queries are proved. The correctness of learnt model is confirmed by checking whether all positive queries are predicted as true and all negative queries are predicted as false by the model.

D.2 Systematicity Tests

All model configurations are the same as in the ILP tasks.

D.3 Knowledge graph Completion

The statistics of datasets are summarised as follows.

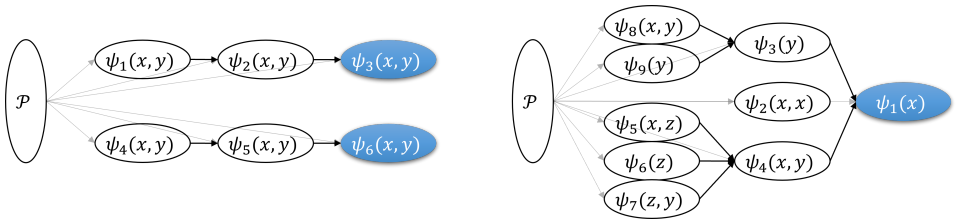

Figure 3: Illustration of two inference networks generated by procedure in Sec. 4.3. We remove $\mathcal{F}_i$ nodes for notation clarity. Output nodes are colored in blue. The left figure corresponds to $\varphi(x,y) := A(x,y) \mid \exists z : \varphi(x,z) \wedge A(z,y)$, which is applicable to the systematicity tests and knowledge graph completion tasks. The right figure corresponds to $\varphi(x) := A(x) \mid \exists y : \varphi(x,y) \wedge \varphi(y)$ and $\varphi(x,y) := A(x,y) \mid \exists z : \varphi(x,z) \wedge \varphi(z) \wedge \varphi(z,y)$, which is used in the ILP tasks.

Table 8: Statistics of datasets.

Dataset	#Entities	#Relations	#Trains	#Validation	#Test
FB15k-237	14514	237	272115	17535	20466
WN18RR	40943	11	86835	3034	3134
Kinship	104	25	3206	2137	5343
UMLS	135	46	1959	1306	3264

Generally, knowledge graphs are much noisier than ILP tasks, as shown in Fig. 2, most learnt rules have a rather small value of accuracy compared to ILP tasks (correct rule accuracy is 1) and systematicity tests (learnt rule accuracy usually more than 0.8). To handle such uncertainty, we set the number of inference iterations to be 1 and learn more rules for each predicate. On all tasks our generative definitions are:

$$\varphi(x,y) := \exists z_1, z_2 : P(x,z_1) \wedge P(z_1,z_2) \wedge P(z_2,y), \tag{37}$$

which corresponds to learning rules of length 3. We create negative statements via negative sampling. Instead of removing the proved queries, we reduce the weights of corresponding queries with the fixed ratio 0.8. We train for 400 times for each relation and remove duplicated rules. When testing, we choose different update functions for inference, including the original ones in Eq. 8, a modification of the original ones where we set the restriction $\psi(\mathbf{x}) = \max\{\psi(\mathbf{x}), 1\}$ thus the evaluation value provided by ψ is exactly the same as the corresponding logic classifier, and a multi-layer perceptron based on the validation data.

For evaluation, we use the standard filtered ranking (Bordes et al. (2013)) metrics, including Mean Rank (MR), Mean Reciprocal Rank (MRR) and Hit@k (H@k). When there are multiple tail nodes assigned with the same score, we compute the expectation of each evaluation metric over all random shuffles of entities (Qu et al. (2021)).

After all, in all of the experiments, we use Adam (Kingma & Ba (2015)) optimizer with $lr = \{0.01, 0.1\}$. We let $\alpha = \beta = \{1, 2\}$. We generate the structural parameters of LPs $\boldsymbol{w}$ and $\boldsymbol{\theta}$ using a softmax function as follows:

$$\begin{aligned} \boldsymbol{w} &= \mathrm{softmax}(\boldsymbol{w}'), \\ \boldsymbol{\theta} &= \mathrm{softmax}(\boldsymbol{\theta}'), \end{aligned} \tag{38}$$

where $\boldsymbol{w}' \in \mathbb{R}^{n_{\boldsymbol{w}}}$ and $\boldsymbol{\theta}' \in \mathbb{R}^{n_{\boldsymbol{\theta}}}$ are real-valued vectors with no constraints. We prepossess the data to add permutations for ordinary predicates, for example, for every 2-ary statement $P(a,b)$ in data we create an invented one $P'(b,a)$. For randomly initialized parameters, we draw them independently from Gaussian distribution $\mathcal{N}(0,1)$, but this is not necessary and other distributions (uniform, Xavier, ...) also work well. Illustrations of inference network architecture constructed with the procedure discribed in Sec. 4.3 are in Fig. 3.

E DISCUSSION OF TIME COMPLEXITY

In this section we discuss the model complexity. Suppose the inference network is composed of N LPs. Suppose the arities of predicates and LPs are at most n. We now discuss the time complexity of each part of the inference model.

General time complexity We first take a look at a single LP's behaviour, where we assume that all other LPs in the network are already evaluated. From the definitions of LPs Eq. 6 and Eq. 7, we can see that evaluating one LP invloves:

$$\psi(\mathbf{x};\alpha) = (\boldsymbol{\theta}^T)^\alpha[\mathcal{F}_1(\mathbf{x};\alpha), \mathcal{F}_2(\mathbf{x};\alpha), \mathcal{F}_3(\mathbf{x};\alpha), ...]^T. \tag{39}$$

Since the total amount of $\mathcal{F}_i$ is limited and does not scale as the network or input data grows, this step costs

$$T(\psi) \leq \sum_i T(\mathcal{F}_i) + O(|\mathcal{V}|^n), \tag{40}$$

where we let $T(\psi)$ be the time complexity for evaluating $\psi(\mathbf{x};\alpha)$ for all $\mathbf{x}$, $T(\mathcal{F}_i)$ be the time complexity for evaluating $\mathcal{F}_i(\mathbf{x};\alpha)$ for all $\mathbf{x}$, etc. For each $\mathcal{F}_i$, we have:

$$\begin{aligned}
&\mathcal{F}_i(\mathbf{x};\alpha) = \left[P_1(\mathbf{x}), P_2(\mathbf{x}), ..., P_{|\mathcal{P}|}(\mathbf{x}), \psi_i'(\mathbf{x})\right] \quad \left(\boldsymbol{w}^{(i)}\right)^\alpha && \iff T(\mathcal{F}_i) \leq O(|\mathcal{P}||\mathcal{V}|^n),\\
&\mathcal{F}_i(\mathbf{x};\alpha) = \mathcal{F}_j(\mathbf{x};\alpha)\mathcal{F}_k(\mathbf{x};\alpha) && \iff T(\mathcal{F}_i) \leq T(\mathcal{F}_j) + T(\mathcal{F}_k) + O(|\mathcal{V}|^n),\\
&\mathcal{F}_i(\mathbf{x};\alpha) = [\mathcal{F}_j(\mathbf{x};\alpha), \mathcal{F}_k(\mathbf{x};\alpha)] \left(\boldsymbol{w}^{(i)}\right)^\alpha && \iff T(\mathcal{F}_i) \leq T(\mathcal{F}_j) + T(\mathcal{F}_k) + O(|\mathcal{V}|^n),\\
&\mathcal{F}_i(\mathbf{x};\alpha) = 1 - \mathcal{F}_j(\mathbf{x};\alpha) && \iff T(\mathcal{F}_i) \leq T(\mathcal{F}_j) + O(|\mathcal{V}|^n),\\
&\mathcal{F}_i(\mathbf{x};\alpha) = \sum_{\mathbf{y}} \mathcal{F}_j(\mathbf{x},\mathbf{y};\alpha) && \iff T(\mathcal{F}_i) \leq T(\mathcal{F}_j) + O(|\mathcal{V}|^n),\\
&\mathcal{F}_i(\mathbf{x},\mathbf{y};\alpha) = \mathcal{F}_j(\mathbf{x};\alpha) && \iff T(\mathcal{F}_i) \leq T(\mathcal{F}_j) + O(|\mathcal{V}|^n).
\end{aligned} \tag{41}$$

Thus we can see that for one ψ, we have

$$T(\psi) \leq \sum_{\mathcal{F}_i} O(|\mathcal{P}||\mathcal{V}|^n) = O(|\mathcal{P}||\mathcal{V}|^n). \tag{42}$$

To evaluate all nodes in the network, we simple evaluate LPs one by one in topological order, which directly gives a total time complexity of

$$T(\Psi) = O(|\mathcal{P}||\mathcal{V}|^n N). \tag{43}$$

The update procedure for one $\mathbf{x}$ is a function with R_Ψ (amount of learnt rules) inputs, and all implementations we introduced here all make the evaluation of the function $O(R_\Psi)$), so the evaluation over all $\mathbf{x}$ takes $O(|\mathcal{V}|^n R_\Psi)$ time.

Time complexity on sparse graphs In reality often the input data is sparse, and the $|\mathcal{V}|^n$ term in time complexities can be reduced significantly. Here, we analyse the time complexity of the model under knowledge graph completion experiments.

Suppose the input graph has $|\mathcal{V}|$ nodes, $|\mathcal{P}|$ predicates and M edges, and we are learning chain-like rules of length L. Thus, each output unit, corresponding to a rule, is composed of L LPs, and can be written as follows:

$$\psi(x, y; \alpha) = \sum_{z_1, z_2, \ldots, z_{L-1}} \mathcal{P}_1(x, z_1; \alpha)\mathcal{P}_2(z_1, z_2; \alpha)...\mathcal{P}_L(z_{L-1}, y; \alpha), \tag{44}$$

where

$$\mathcal{P}_i(x, y; \alpha) = [P_1(x, y), P_2(x, y), ...]\boldsymbol{w}_i^\alpha. \tag{45}$$

We can efficiently calculate $\psi(x, y; \alpha)$ for all nodes y in the knowledge graph with the same source x. The algorithm is an extension of L-step breadth first search described as follows. S_L maps nodes y with nonzero value $\psi(x, y; \alpha)$ to $\psi(x, y; \alpha)$.

Algorithm 1 Inference in sparse graphs

Input: graph $\mathcal{G}$, predicates $\mathcal{P}$, source node x, rule length L, model parameters $\alpha, \boldsymbol{w}_1, \boldsymbol{w}_2, ..., \boldsymbol{w}_L$.
Output: S_L.

```
function EVALUATE(x, G, L, α, w_1, w_2, ..., w_L)
    S_0 ← MAP(∅)
    S_0[x] ← 1.0
    for l ← 1 to L do
        S_l ← MAP(∅)
        for P ∈ P, s ∈ S_{l-1}, t ∈ N_P(s) do
            if t ∉ S_l then
                S_l[t] ← 0.0
            end if
            S_l[t] ← S_l[t] + (w_l[P])^α  S_{l-1}[s]
        end for
    end for
    return S_L
end function
```

Here, we denote $\mathcal{N}(s)$ as the neighbors of s in the KG, and $\mathcal{N}_P(s)$ as the neighbors of s connected by edge type P. Thus, one run of the function takes at least

$$\begin{aligned} T_{\text{single}} &= \sum_{l} \sum_{s \in S_{l-1}} \sum_{P \in \mathcal{P}} \sum_{t \in \mathcal{N}_P(s)} O(1) \\ &= \sum_{l=1}^{L} \sum_{s \in S_{l-1}} O(|\mathcal{N}(s)|) \end{aligned} \tag{46}$$

Denoting $N^{(k)}$ as the max amount of nodes in a node's k-hop subgraph, we have

$$\begin{aligned} T_{\text{single}} &\leq \sum_{l=1}^{L} N^{(l-1)} O(N^{(1)}) \\ &\leq O(L N^{(L)} N^{(1)}). \end{aligned} \tag{47}$$

Thus, one-time of evaluating a total number of R_Ψ rules for all node pairs in the graph takes

$$\begin{aligned} T &\leq R_\Psi |\mathcal{V}| T_{\text{single}} \\ &= O(R_\Psi L N^{(L)} N^{(1)} |\mathcal{V}|). \end{aligned} \tag{48}$$

Since in sparse graphs we often have $N^{(1)} << N^{(L)} << |\mathcal{V}|$, this estimation of time complexity (Eq. 48) is much less than the original one (Eq. 43), which is $O(R_\Psi L |\mathcal{P}| |\mathcal{V}|^3)$ in this case.

F CASE STUDIES

In this section we illustrate part of the logic rules we learned on ILP tasks, systematicity tests and knowledge graph completion.

Target	$\leftarrow$	**Rules**
DivisibleBy3(x)	$\leftarrow$	$\psi_1(x) = \text{Zero}(x)$
	$\leftarrow$	$\psi_2(x) = \exists y : \psi_3(y) \wedge \text{Succ}(y, x)$
		$\psi_3(x) = \exists y : \psi_4(y) \wedge \text{Succ}(y, x)$
		$\psi_4(x) = \exists y : \text{DivisibleBy3}(y) \wedge \text{Succ}(y, x)$
AdjacentToRed(x)	$\leftarrow$	$\psi_1(x) = \exists y : \text{Red}(y) \wedge \text{Edge}(x, y)$
Connect(x, y)	$\leftarrow$	$\psi_1(x, y) = \text{Edge}(x, y)$
	$\leftarrow$	$\psi_2(x, y) = \exists z : \text{Connect}(x, z) \wedge \text{Edge}(z, y)$
Grandmother(x, y)	$\leftarrow$	$\psi_1(x, y) = \exists z : \text{Brother}(x, z) \wedge \text{Grandmother}(z, y)$
	$\leftarrow$	$\psi_2(x, y) = \exists z : \text{Father}(x, z) \wedge \text{Mother}(z, y)$
Causes(x, y)	$\leftarrow$	$\psi_1(x, y) = \exists z_1, z_2 : \text{Contains}(z_1, x) \wedge \text{LocationOf}(z_1, z_2)$ $\wedge\text{OccursIn}(z_2, y)$
	$\leftarrow$	$\psi_2(x, y) = \exists z_1, z_2 : \text{Contains}(z_1, x) \wedge \text{LocationOf}(z_1, z_2)$ $\wedge\text{Complicates}(y, z_2)$
	$\leftarrow$	$\psi_3(x, y) = \exists z_1, z_2 : \text{IngredientOf}(z_1, x) \wedge \text{IsA}(z_1, z_2)$ $\wedge\text{Causes}(z_2, y)$
	$\leftarrow$	$\psi_4(x, y) = \exists z_1, z_2 : \text{IngredientOf}(z_1, x) \wedge \text{InteractsWith}(z_2, z_1)$ $\wedge\text{Causes}(z_2, y)$