PROVER: Proof Generation for Interpretable Reasoning over Rules

Swarnadeep Saha **Sayan Ghosh** **Shashank Srivastava** **Mohit Bansal**
UNC Chapel Hill
{swarna, sayghosh, ssrivastava, mbansal}@cs.unc.edu

Abstract

Recent work by Clark et al. (2020) shows that transformers can act as "soft theorem provers" by answering questions over explicitly provided knowledge in natural language. In our work, we take a step closer to emulating formal theorem provers, by proposing PROVER, an interpretable transformer-based model that jointly answers binary questions over rule-bases and generates the corresponding proofs. Our model learns to predict nodes and edges corresponding to proof graphs in an efficient constrained training paradigm. During inference, a valid proof, satisfying a set of global constraints is generated. We conduct experiments on synthetic, hand-authored, and human-paraphrased rule-bases to show promising results for QA and proof generation, with strong generalization performance. First, PROVER generates proofs with an accuracy of 87%, while retaining or improving performance on the QA task, compared to RuleTakers (up to 6% improvement on zero-shot evaluation). Second, when trained on questions requiring lower depths of reasoning, it generalizes significantly better to higher depths (up to 15% improvement). Third, PROVER obtains near perfect QA accuracy of 98% using only 40% of the training data. However, generating proofs for questions requiring higher depths of reasoning becomes challenging, and the accuracy drops to 65% for "depth 5", indicating significant scope for future work.[1]

1 Introduction

Developing systems that can understand and reason over explicitly provided knowledge has been a fundamental goal of AI (Newell and Simon, 1956). Owing to the challenges posed in reasoning over formal representations (Musen and Van Der Lei, 1988), and backed by the recent successes of transformers (Vaswani et al., 2017) in NLP, Clark et al. (2020) propose a new version of the problem by replacing the formal representations of rule-bases with natural language (English). Specifically, their task requires predicting the truth value of a statement by reasoning over a set of facts and rules, all expressed in natural language. Figure 2 shows some examples of the task. Clark et al. (2020) propose RuleTakers, a fine-tuned RoBERTa model (Liu et al., 2019b) to show that transformers can act as "soft theorem provers" by predicting the final answer in such reasoning-based problems with high accuracy.

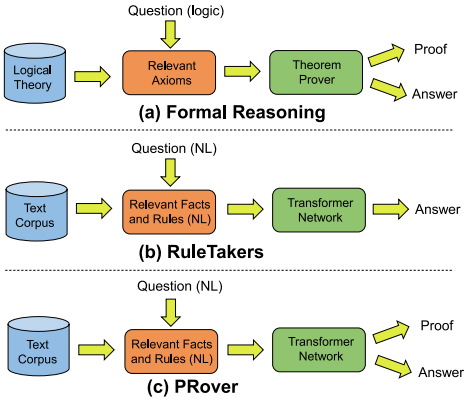


Figure 1: Block diagram showing that PROVER is a closer linguistic analog of formal reasoning.

We argue that to use transformers for natural language reasoning reliably, they should be able to generate proofs that provide rationales for the predicted answer. Proof generation is vital for emulating formal reasoning but also for moving towards human-interpretable models that alleviate concerns about the black-box nature of deep architectures (Rudin, 2019). Towards this, we present PROVER, a transformer-based model that jointly answers questions over natural language rule-bases and generates corresponding proofs. Figure 1 illustrates our method as a closer linguistic analog

[1]Our code and models are publicly available at https://github.com/swarnaHub/PRover.

Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, pages 122–136,
November 16–20, 2020.

of formal reasoning, as it generates proofs along with answers. However, unlike formal reasoners, PROVER can operate on natural language text that provides the underlying theory, rather than rely on formal logical representations. Such methods that combine interpretability and flexibility in reasoning can have wide applications across domains.

PROVER's architecture consists of three modules that together generate answers along with proofs. In this work, proofs are represented as directed graphs consisting of the relevant rules and facts needed to prove or disprove the question statement. Section 3.1 contains details of this representation. A QA module predicts a binary answer for the question, a node module chooses which rules and facts are part of the proof, and an edge module predicts the presence and the direction of the edges between the chosen nodes. Model training minimizes a joint cross-entropy loss over the three modules. To guide the model to predict edges between valid nodes only, we enforce global constraints on the structure of the proof during training, by masking out labels for impossible edges, resulting in a more efficient learning problem. PROVER generates valid proofs during inference by solving an ILP over the edge potentials, subject to multiple semantic constraints, such as ensuring proof graph connectivity. Our contributions are:

- We present PROVER, an interpretable joint model that learns to reason over natural language rule-bases and generate corresponding proofs.
- PROVER performs similarly or improves upon state-of-the-art QA accuracy for the task, with up to 6% improvement on zero-shot evaluation, and generates exact proofs at 87% accuracy. Unlike RuleTakers, it does not require additional fine-tuning on the RACE (Lai et al., 2017) dataset.
- PROVER demonstrates significantly better generalization. When trained on lower depth questions, it shows better QA accuracy (up to 15%) on higher depth ones.

2 Related Work

Our work is related to multiple bodies of previous work in NLP and formal reasoning.

QA and NLI: The rule reasoning task is related to reasoning tasks that have been proposed recently. These include tasks in the bAbI dataset (Weston et al., 2015), synthetically generated probe tasks (Richardson et al., 2020) or reading comprehension tasks in datasets such as QuaRTz (Tafjord et al., 2019) and ROPES (Lin et al., 2019). Unlike our task, most of these require reasoning over implicit rules, the focus being on language understanding and one step of rule application. Multi-hop QA datasets like HotpotQA (Yang et al., 2018) require multiple reasoning steps, but the inference rules needed are again implicitly inferred, rather than explicitly provided. Our task also bears similarity with Natural Language Inference (MacCartney and Manning, 2014), but NLI also allows unsupported inferences by filling gaps in explicitly stated knowledge (Dagan et al., 2013).

Formal Reasoning and Neural Theorem Proving: Semantic parsing (Zettlemoyer and Collins, 2005; Berant et al., 2013; Berant and Liang, 2014) of multi-sentence texts into logical forms has proved to be challenging, restricting the application of semantic parsers to formal reasoning systems (Kamath and Das, 2019). PROVER bypasses this expensive and error-prone process and attempts to solve the problem in an end-to-end manner, without any intermediate logical representations.

Our approach is conceptually similar to a body of work on Neural Theorem Proving (Weber et al., 2019) that has focused on developing theorem provers by combining reasoning from symbolic techniques with the possibility of differentiable learning from neural networks. These include neuro-symbolic methods for table comprehension (Neelakantan et al., 2016), executing basic compositional programs (Reed and de Freitas, 2016), SAT solving (Selsam et al., 2019), formula embedding (Abdelaziz et al., 2020), approximate (DNF) model counting (Abboud et al., 2020), etc. However, PROVER diverges from these in working with free-form natural language input to generate proofs similar to formal reasoners.

Model Interpretability: PROVER follows a significant body of previous work on developing interpretable neural models for NLP tasks to foster explainability. Several approaches have focused on formalizing the notion of interpretability (Rudin, 2019; Doshi-Velez and Kim, 2017; Hase and Bansal, 2020), tweaking features for local model interpretability (Ribeiro et al., 2016, 2018) and exploring interpretability in latent spaces (Joshi et al., 2018; Samangouei et al., 2018). Our work can be seen as generating explanations in the form of proofs for an NLP task. While there has been prior work on generating natural language explana-

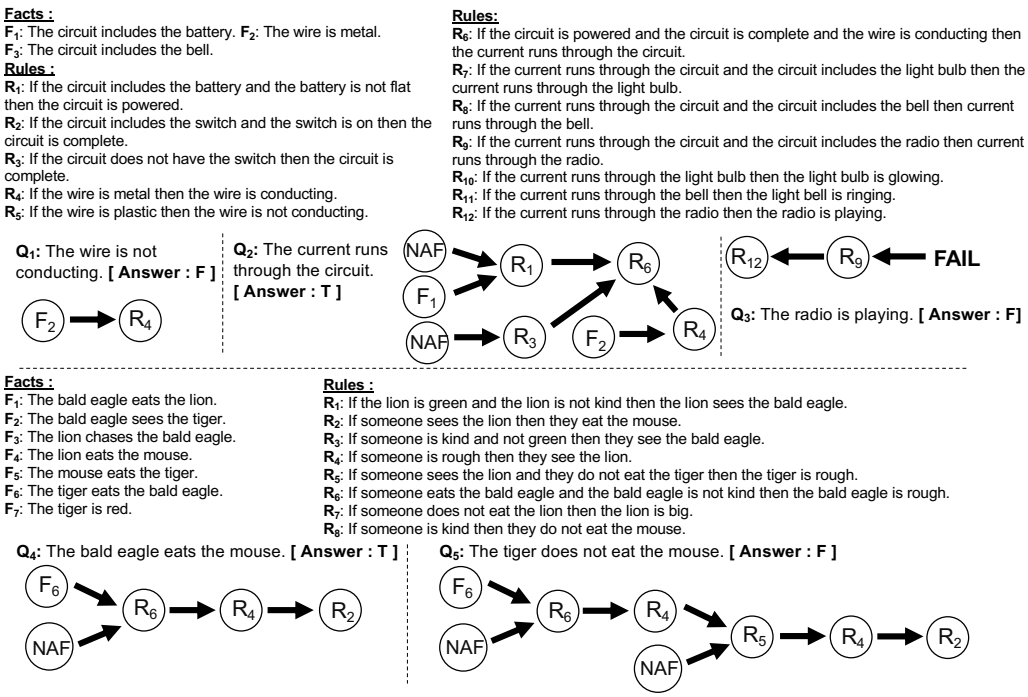


Figure 2: Diagram showing two rule-bases with rules, facts, questions, answers and proofs. PROVER answers all the questions correctly and also generates all the corresponding proofs accurately in the above scenarios.

tions for multiple NLP tasks, including NLI (Camburu et al., 2018), commonsense reasoning (Rajani et al., 2019; Zhang et al., 2020) and generic text classification tasks (Liu et al., 2019a), our novelty lies in generating compositional explanations consisting of proof graphs that detail the chain of reasoning, starting from language. We use a max-flow ILP formulation for checking proof graph connectivity (Even and Tarjan, 1975). Multiple approaches for NLP tasks such as sentiment analysis and content selection (Pang and Lee, 2004; Barzilay and Lapata, 2005; Bansal et al., 2008) have been framed as optimal flow problems on graphs.

Program Synthesis with Transformers: Existing works show that transformers already capture some knowledge from pre-training for algorithm emulation (Talmor et al., 2019) or can be fine-tuned for tasks like semantic parsing (He and Choi, 2020), translation (Wang et al., 2019), symbolic integration (Lample and Charton, 2020) and mathematics (Saxton et al., 2019). In our work, we also employ a transformer-based pre-trained language model (RoBERTa (Liu et al., 2019b)) but for the downstream task of rule-based reasoning.

3 Method

Each input to PROVER is a context C (consisting of facts F and rules R) and a question Q, about the context. PROVER predicts the answer $A \in \{True, False\}$ and generates a proof $\mathcal{P}$.

3.1 Proof Representation

A proof, $\mathcal{P} = (\mathcal{N}, \mathcal{E})$, is a directed graph with nodes $n \in \mathcal{N}$ and edges $e \in \mathcal{E}$. Each node is either a fact $f \in F$, a rule $r \in R$ or a special NAF node (Negation As Failure, as described below). Edges in the proof are directed either from a fact (or NAF) to a rule or from a rule to another rule. These indicate that a fact (or NAF) is consumed by a rule, or the output of a rule (a new fact) is consumed by another rule, respectively. We use these constraints both during PROVER's training and inference, as described later in the paper. Formally, we have:

$$\mathcal{P} = (\mathcal{N}, \mathcal{E})$$
$$\mathcal{N} \subseteq R \cup F \cup NAF$$
$$\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$$

Figure 2 shows examples of two contexts (consisting of facts and rules), five questions about the contexts, along with their answers and proofs. Each proof has a depth (Q_1's proof has a depth of 1). The maximum proof depth in all the datasets considered in this work (Clark et al., 2020) is 5. Proofs in the datasets are of three types:

Successful proof with NAF: The proof of Q_1 in Figure 2 is one such such example. F_2 acts on R_4 to prove that "The wire is conducting." and hence the answer is false.

Successful proof with NAF: Given a statement s, NAF in logic programming is a non-monotonic inference rule used to derive "not s" (negation of the statement) from failure to derive s. Hence, a proof may contain NAF node(s), representing the truthfulness of the negation of statement(s) that cannot be proved using the set of rules. Consider the proofs for Q_4 and Q_5 where the NAF node in Q_4 represents "The bald eagle is not kind.".

Failed proof: This happens when a statement cannot be derived using the given rule-base and the shallowest branch of the proof tree that fails is shown. Q_3's proof in Figure 2 is an example as "The radio is playing." cannot be proved.

Note that a proof can have edges between two rules in both directions. E.g., consider the edges $R_4 \rightarrow R_5$ and $R_5 \rightarrow R_4$ in Q_5's proof in Figure 2. A node can have more than two incoming edges – the node R_6 in Q_2 has three incoming edges from R_1, R_3, and R_4.

3.2 Task Description

Each training example is a tuple $(C_i := \{F_i, R_i\}, Q_i, A_i, \mathcal{P}_i)$ consisting of a context (set of rules and facts), a question, the corresponding answer, and a proof. Generating a proof graph requires (1) identifying the nodes (set of relevant facts, NAF and rules) that are part of the proof, (2) identifying the edges connecting these nodes, and (3) verifying a set of global constraints such as proof connectivity that ensure a valid proof.

For the first, we predict a binary label over each rule, fact and NAF denoting their presence or absence in the proof. For the second, we also predict binary labels denoting the presence or absence of each edge. For the third, we enforce constraints during both training and inference (Section 3.4). During training, we mask out the edge labels[2] corresponding to (1) self-loops, (2) edges between absent nodes, and (3) edges between facts to facts and rules to facts. This enforces a semantic constraint that the set of candidate edges in the ensuing proof is consistent with the chosen set of nodes, and also simplifies the learning problem, since a smaller number of edges need to be labeled.

3.3 PROVER: Joint QA and Proof Generation Model

Figure 3 shows the architecture of PROVER, built on top of RoBERTa (Liu et al., 2019b). Our model consists of three modules: (1) QA module, (2) Node module, and (3) Edge module. The QA module is exactly the same as the RuleTakers model (Clark et al., 2020), thus allowing us to directly evaluate the effectiveness of our node and edge modules. The input to RoBERTa is the concatenation of the context and the question, separated by the $[SEP]$ tokens. The context is represented by concatenating the text consisting of facts and rules. Formally, if the rules and facts are denoted by $\{RF_i\}_{i=1}^{k}$ and the question by Q, the input is

$$[CLS]\ \{RF_i\}_{i=1}^{k}\ [SEP][SEP]\ Q\ [SEP]$$

QA Module: The output of RoBERTa contains an embedding for each token in the context and a global embedding corresponding to the $[CLS]$ token. The QA classification head H_{QA} is a sequence of two linear layers with dropout probability of p. Formally, if $t_{[CLS]}$ denotes the $[CLS]$ token embedding, we obtain the class-wise probability values P_{QA} using the softmax function σ.

$$P_{QA} = \sigma(H_{QA}(t_{[CLS]}))$$

Node Module: Let $\{w_j^{(i)}\}_{j=1}^{m}$ denotes the m tokens corresponding to RF_i. Assuming the corresponding RoBERTa embeddings are denoted by $\{t_{w_j^{(i)}}\}_{j=1}^{m}$, we learn a representation t_{RF_i} for each RF_i, by performing a mean pooling MP of the constituent token embeddings.

$$t_{RF_i} = MP(\{t_{w_j^{(i)}}\}_{j=1}^{m})$$

We also learn a representation of the NAF node t_{NAF} as a linear transformation on $t_{[CLS]}$. Note that due to the self-attention layers of RoBERTa, $t_{[CLS]}$ summarizes the set of all derivable facts given the context and the question. We want the NAF node to encode information about all facts containing negation (e.g.,"The bald eagle is not kind" in Q_4's proof of Figure 2) in the context. These are taken as true as their positive counterparts ("The bald eagle is kind") are non-derivable given the context. Thus, if a statement s cannot be derived from the facts and the rules in a context, the NAF node should infer that "not s" is true. We model this notion of the negation of all unprovable statements (given a context) by learning NAF as a function of everything provable in the context, encoded by the $t_{[CLS]}$ embedding.[3]

[2]The masked edges do not contribute to the training loss.

[3]We note that a proof can have multiple NAF nodes, each representing a different negated fact. Since the datasets label

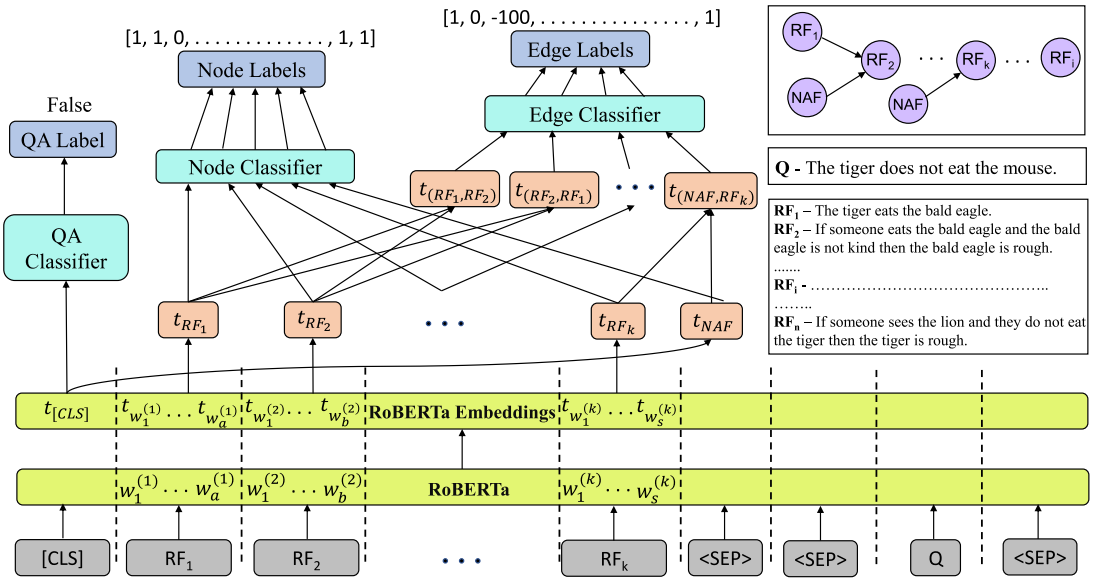


Figure 3: Architecture diagram of PROVER. The presence and absence of nodes/edges are labeled by 1 and 0 respectively while -100 represents masked out edges.

The node classifier H_{Node} has a similar architecture to the QA classifier and predicts a presence and absence probability score for each node.

$$P_{Node} = \sigma(H_{Node}(\{t_{RF_i}\}_{i=1}^{k}, t_{NAF}))$$

Edge Module: Now, given the representations of each fact, rule and NAF, we learn a representation for each edge between these. Formally, we define the edge embedding $t_{(RF_i,RF_j)}$ from node RF_i to node RF_j by concatenating their individual embeddings t_{RF_i} and t_{RF_j} with their element-wise difference (which gives the directionality vector).

$$t_{(RF_i,RF_j)} = [t_{RF_i}, t_{RF_j}, (t_{RF_j} - t_{RF_i})]$$

The above formulation also helps learn separate representations for edges $RF_i \rightarrow RF_j$ and $RF_j \rightarrow RF_i$. This is essential for our task as a proof can have edges between two rules in both directions. In Section 4.7, we see that this formulation leads to a near perfect empirical performance in predicting the directionality of edges. The edge classifier H_{Edge} outputs probability scores representing the presence and absence of each edge.

$$P_{Edge} = \sigma(H_{Edge}(\{t_{(RF_i,RF_j)}\}_{i,j=1}^{k+1}))$$
$$t_{RF_{k+1}} = t_{NAF}$$

We train our model by using binary cross-entropy loss for each of the three modules. Formally, if L_{QA}, L_{Node} and L_{Edge} denote the three losses, the overall loss L is given by:

$$L = L_{QA} + L_{Node} + L_{Edge}$$

3.4 ILP Inference for Global Constraints

As mentioned previously, during inference, we enforce additional constraints on the structure of the predicted proof graph. For this, we frame inference as Integer Linear Program (ILP) optimization, which we describe next. We follow the generative process of a graph wherein the nodes are defined first, followed by the edges on that set of nodes. Thus, we fix the nodes first based on the predictions by the node module of our model and maximize a global score over the set of edges only. This reduces the large search space and ensures that all constraints can be expressed as linear expressions.

Proof Connectivity Formulation: An important constraint is to ensure that the predicted proof graphs are connected.[4] To check if a proof graph $\mathcal{P}$ is connected, we define an augmented graph $\mathcal{P}_{aug} = (\mathcal{N}_{aug}, \mathcal{E}_{aug})$ with two added nodes "source" and "sink". We add an edge from the source to any one of the nodes x in $\mathcal{P}$. We also define edges from all nodes in $\mathcal{P}$ to the sink.

$$\mathcal{N}_{aug} = \mathcal{N} \cup \{source, sink\}$$
$$\mathcal{E}_{aug} = \mathcal{E} \cup \{(source, x)\} \cup \{(n, sink) \forall n \in \mathcal{N}\}$$

Having defined $\mathcal{P}_{aug}$, we can reduce the graph connectivity in $\mathcal{P}$ to a maximum flow problem

all these as NAF, we collapse all the NAF nodes into a single node and learn a unified representation for them.

[4] Proofs are directed graphs. We check connectivity in the equivalent undirected graphs.

(Leighton and Rao, 1999) in $\mathcal{P}_{aug}$ (Even and Tarjan, 1975). For this, we define the capacity variable $c_{(m,n)}$ for each edge, $m \rightarrow n$ in P_{aug}, as follows.

$$\begin{aligned} c_{(source,x)} &= |\mathcal{N}| \text{ and } c_{(x,source)} = 0 \\ \forall n \in \mathcal{N}, c_{(n,sink)} &= 1 \text{ and } c_{(sink,n)} = 0 \\ \forall m,n \in \mathcal{N}, c_{(m,n)} &= |\mathcal{N}| \\ c_{(m,n)} &= 0 \text{ if } m \notin \mathcal{N} \text{ or } n \notin \mathcal{N} \end{aligned}$$

Now, there can be a maximum total flow of $|\mathcal{N}|$ from the source to the sink, if and only if the graph is connected. We use this flow formulation to provide additional constraints for our ILP inference procedure that ensure connectivity of proof graphs.

Final Optimization Problem: Our maximization objective, subject to the connectivity constraint and all other constraints (that ensure a valid proof) is as follows. Let $\phi_{(m,n)}$ represent the probability that an edge $m \rightarrow n$ is present, as predicted by PROVER. We want to infer 0/1 assignments for our optimization variables $e_{(m,n)}$ (a value of 1 means the edge is part of the proof, while 0 means it is not) such that the following objective is maximized:

$$\begin{aligned} \underset{e_{(m,n)}, f_{(m,n)}}{argmax} \sum_{m,n,m \neq n} & (\phi_{(m,n)} e_{(m,n)} + \\ & (1 - \phi_{(m,n)})(1 - e_{(m,n)})) \end{aligned}$$

subject to constraints:

$$\forall m,n \in F \cup R \cup NAF, e_{(m,n)} \in \{0,1\} \quad (1)$$

$$e_{(m,n)} = 0, \text{if } m \notin \mathcal{N} \text{or } n \notin \mathcal{N} \quad (2)$$

$$e_{(m,n)} = 0, \text{if } m \in F \text{ and } n \in F \quad (3)$$

$$e_{(m,n)} = 0, \text{if } m \in R \text{ and } n \in F \quad (4)$$

$$\forall m,n \in \mathcal{N}_{aug}, 0 \leq f_{(m,n)} \leq c_{(m,n)} \quad (5)$$

$$\forall n \in \mathcal{N}_{aug}, \sum_{m:(m,n) \in \mathcal{E}_{aug}} f_{(m,n)} = \sum_{o:(n,o) \in \mathcal{E}_{aug}} f_{(n,o)} \quad (6)$$

$$f_{(source,x)} = |\mathcal{N}| \quad (7)$$

$$\forall m,n \in \mathcal{N}_{aug}, e_{(m,n)} + e_{(n,m)} - (f_{(m,n)}/|\mathcal{N}|) \geq 0 \quad (8)$$

Note that $\mathcal{N}$, F, and R refer to the set of predicted nodes (from the model), the set of facts, and the set of rules, respectively. Equations 2, 3 and 4 ensure that edges are present only when the corresponding nodes are present and that there are no edges between two facts and from a rule to a fact. Next, to ensure proof connectivity, we first define the flow constraints in Equations 5 and 6 constrained by the flow variables $f_{(m,n)}$ for each edge $m \rightarrow n$. These maintain the capacity constraints (the flow at each edge should be less than its capacity) and the flow conservation constraints (the total flow through the incoming edges at a node is equal to the total flow through the outgoing edges). Equation 7 ensures connectivity in the proof graph, by enforcing the total flow to be $|\mathcal{N}|$. Finally, we ensure that the proof connectivity is checked on the valid edges only (which are part of the proof) through the last constraint, since a max-flow of $|\mathcal{N}|$ is achievable for any connected graph.

4 Experiments

Our experiments evaluate the effectiveness of PROVER (PR), our joint QA, and proof model against RuleTakers (RT). Details of our experimental setup are in the appendix.

4.1 Datasets and Evaluation Metrics

We conduct experiments on all the three sets of datasets introduced in Clark et al. (2020) and consisting of gold answers and proofs. Further details of the datasets can be found in the appendix.

DU0-DU5: The first set consists of five datasets, each containing 100k questions with theories in synthetic language and requiring reasoning paths up to depth D ($D = 0, 1, 2, 3, 5$). We refer to these datasets as DU0, DU1, DU2, DU3 and DU5, where DU stands for "Depth Upto".

Birds-Electricity: It consists of two test-only datasets of 5k samples used to evaluate the out-of-distribution performance of the models.

ParaRules: ParaRules consists of 40k questions against 2k theories expressed in paraphrased natural language, obtained through crowdsourcing.

We evaluate QA performance through accuracy. For proofs, we introduce three metrics: (1) **Node Accuracy (NA):** Fraction of examples where the predicted node set matches exactly with the gold node set, (2) **Edge Accuracy (EA):** Fraction of examples where the predicted edge set match exactly with the gold set, and (3) **Proof Accuracy (PA):** Fraction of examples where the generated proof matches exactly with the gold proof. For examples with multiple gold proofs, we give credit if the prediction matches exactly with any one of the proofs. We also evaluate **Full Accuracy (FA)**, denoting the fraction of samples where both the answer and the

D	Cnt	QA		NA	EA	PA	FA
		RT	PR				
0	6299	**100**	**100**	98.6	98.5	98.4	98.4
1	4434	98.4	**99.0**	93.3	95.1	93.2	93.1
2	2915	98.4	**98.8**	85.9	84.8	84.8	84.8
3	2396	98.8	**99.1**	82.3	80.5	80.5	80.5
4	2134	**99.2**	98.8	77.7	72.5	72.5	72.4
5	2003	**99.8**	99.3	76.0	65.1	65.1	65.1
All	20192	99.2	**99.3**	89.2	87.5	87.1	87.1

Table 1: QA comparison between RT and PR for varying depths along with node, edge, proof and full accuracy for PROVER on DU5. Cnt = Sample Count.

	Cnt	QA		NA	EA	PA	FA
		RT	PR				
B1	40	**97.5**	95.0	92.5	92.5	92.5	92.5
B2	40	**100**	95.0	95.0	95.0	95.0	95.0
E1	162	96.9	**100**	95.1	96.3	95.1	95.1
E2	180	98.3	**100**	91.7	93.3	91.7	91.7
E3	624	**91.8**	89.7	72.3	73.1	72.3	71.8
E4	4224	76.7	**84.8**	81.4	81.3	80.6	80.6
All	5270	80.1	**86.5**	81.3	81.4	80.7	80.5

Table 2: Zero-shot performance comparison on the Birds-Electricity dataset after training on DU5.

proof are exactly correct.

4.2 QA and Proof Results for Varying Depths

We first train and evaluate PROVER on the train and test splits of the DU5 dataset, and compare its QA performance with RuleTakers for questions of varying depths (D). Table 1 shows these results and the proof-related metrics for PROVER. The corresponding validation set results can be found in the appendix. Overall, and at each depth, PROVER matches the QA performance of RuleTakers. PROVER is also able to generate exact proofs fairly accurately at 87%. Perhaps unsurprisingly, we find that edge prediction is a harder task than node prediction, and performance worsens with increasing depth due to an increasingly large number of edges to be labeled. The proof accuracy matches the edge accuracy at each depth, suggesting that proofs are almost always correct if the edges are correct. Similarly, the full accuracy matches the proof accuracy, showing that the predicted answer is almost always correct when the corresponding proof is correct. This points to an interesting observation – QA is easier than node prediction, which in turn is easier than edge prediction. All the datasets experimented with exhibit this behavior, as we also describe later. Proof generation becomes harder with increasing depth (and hence, more nodes and edges), as the exact proof generation accuracy drops to 65% for depth 5. On analyzing further, we find that on average, PROVER correctly predicts 6 out of 7 edges present in a depth 5 proof. Overall, PROVER is interpretable yet efficient, as it generates proofs fairly accurately without any loss in QA performance.

4.3 Zero-Shot Evaluation

Following previous work (Clark et al., 2020), we now test the out-of-distribution performance of

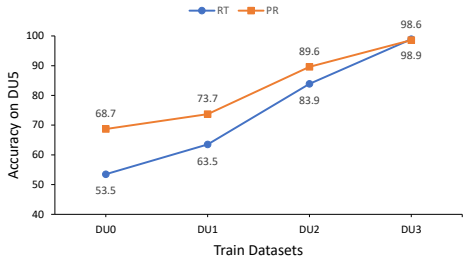


Figure 4: QA performance comparison between PROVER and RuleTakers with models trained on DU0, DU1, DU2 and DU3 and tested on DU5.

PROVER on the Birds-Electricity dataset (Table 2). The DU5-trained model is tested on six datasets, two from the birds domain (B1, B2) and another four from the electricity domain (E1, E2, E3, E4). Overall, our model achieves a 6% QA improvement over RuleTakers. More importantly, PROVER outperforms RuleTakers by 8% on the hardest and largest E4 subset of the data. The proof accuracy is also fairly high, demonstrating good proof generation ability of our model for out-of-distribution data as well. Similar to the test results on DU5, the full accuracy matches the proof accuracy, demonstrating proof consistency with the predicted answers. We show examples of proofs generated by PROVER in Figure 2 and in the appendix.

4.4 Generalization to Higher Depths

We evaluate the generalization ability of PROVER compared to RuleTakers by training models on the train splits of DU0, DU1, DU2 and DU3, and testing the QA performance on the overall test set for DU5, which includes questions with higher depth than seen during training. The corresponding validation set and proof-related results can be found in the appendix. As shown in Figure 4, PROVER, when trained on depth 0 examples only, performs significantly better than RuleTakers with an improvement of 15%. A similar trend is observed for DU1 and DU2, where PROVER improves by 10%

D	Cnt	QA		NA	EA	PA	FA
		RT	PR				
0	2968	**99.8**	99.7	99.5	99.9	99.5	99.4
1	2406	**99.3**	98.6	98.0	98.9	98.0	97.3
2	1443	**98.2**	98.2	89.2	88.9	88.9	88.7
3	1036	**96.7**	96.5	92.1	90.0	90.0	89.9
4	142	**90.1**	88.0	87.3	76.1	76.1	76.1
All	8008	**98.8**	98.4	96.0	95.8	95.4	95.1

Table 3: Comparison of models trained on DU3 and ParaRules training sets and tested on ParaRules test set.

and 6%, respectively. On DU3, both models show high and comparable performance. PROVER's superior generalization ability can be attributed to the extra training supervision incorporated in the form of proofs and an inductive bias for making proof-based predictions. While proof construction for supervised training is expensive, PROVER's superior QA results on out-of-distribution data (Table 2) and higher depth questions is a potential first step to showing that limited proof supervision can still lead to effective generalization.

4.5 Varying Training Data Size

We explore varying the amount of training data from 10k to 30k to all the examples (70k) in DU5. As shown in Table 4, when trained with only 40% of the data, PROVER obtains a near-perfect QA accuracy of 97.8%. Thus, for QA, PROVER's joint training with proofs can compensate for the lack of training data. Proof generation, however, is much harder and with increased training data, the rate of increase in proof accuracy is much more gradual.

4.6 Evaluation on Complex Language

We also test PROVER's ability to generate proofs for more human-like natural language theories. More details on the ParaRules dataset can be found in the appendix. Following Clark et al. (2020), we train a model by combining the DU3 and ParaRules training partitions and test on the ParaRules test partition. Table 3 again shows that PROVER matches the QA performance of RuleTakers, and also generates proofs with a high accuracy of 95%. Following previous trends, the proof accuracy drops as the depth increases, and QA performance is higher than for node prediction, which in turn is higher than for edge prediction.

4.7 Ablation and Error Analysis

Table 5 analyzes the effectiveness of the individual components of PROVER through an ablation study. These ablated variants also provide natural baselines for our proof-related results. Specifically, we train and test the following models on DU5: (1) **QA+Node:** We train a model consisting of only the QA and Node modules; (2) **No NAF:** We train a model using random NAF embeddings; (3) **Unconstrained Train (UT) + No ILP:** We remove constraints both during training and inference; (4) **Unconstrained Train (UT) + ILP:** We remove constraints only during training; (5) **No Connectivity:** Finally, we train a model where we only remove the connectivity constraint during inference. More details about these models in appendix.

Count	QA	NA	EA	PA	FA
10k	87.1	48.1	44.7	44.0	42.7
30k	97.8	77.9	73.2	72.5	72.4
70k	99.3	89.2	87.5	87.1	87.1

Table 4: Comparison of PROVER models trained with varying amount of training data on DU5. Count = Number of training examples.

The QA accuracy is mostly unaffected in all our models and all but "No NAF" have similar node accuracy. The "No NAF" model does not learn a representation for NAF, leading to 5-6% drop in both node and edge accuracy. The 5-6% drop in edge and proof accuracy for the "Unconstrained Train + No ILP" model, compared to PROVER, shows that removing constraints results in a harder learning problem and the model fails to automatically learn all the constraints. The proof accuracy improves slightly when we add constraints only during inference ("Unconstrained Train + ILP"). The connectivity constraint provides only marginal improvement as our model mostly predicts connected proofs without any explicit supervision. Specifically, only 57 examples have disconnected proofs without this constraint. The overall PROVER model outperforms all variants in full accuracy.

To better understand the loss of accuracy for higher depth proofs, we perform error analysis of PROVER for the depth 5 subset of DU5. We find that our NAF learning module is highly accurate – PROVER correctly predicts NAF in a proof 95% of the time. Among all examples with incorrectly predicted node sets, 42% are such that the predicted set is a subset of the gold set while for 25% examples, it is a superset, demonstrating that our model tends to underestimate the number of essential rules and facts. PROVER almost perfectly identifies the direction of edges. We find only 1 example where the proof is incorrect solely due to the incorrect

	QA	NA	EA	PA	FA
QA+N+E (PR)	99.3	89.2	87.5	**87.1**	**87.1**
QA+N	99.4	88.9	-	-	-
QA (RT)	99.2	-	-	-	-
No NAF	**99.5**	83.1	82.3	81.7	81.7
UT + No ILP	99.4	**90.1**	83.0	81.9	81.9
UT + ILP	99.4	**90.1**	83.4	82.9	82.8
No Connectivity	99.3	89.2	**87.8**	87.0	87.0

Table 5: Ablation studies of PROVER showing the importance of each component and constraints.

identification of directionality. Further, 21% of the incorrectly predicted edges are subsets of the gold sets, while 35% are supersets.

5 Discussion and Future Work

Graph-based Explanations: While we have presented PROVER as a model that can emulate formal reasoning, it has further potential use as an explanation generation system. PROVER generates compositional explanations in the form of graphs and QA systems, in general, can potentially benefit from generating such graphical explanations. For example, in multi-hop QA tasks, the node module can choose all the relevant sentences in the context and the edge module can identify the flow of information between these to arrive at the answer (in the presence of task-specific constraints). Graphical explanations, in contrast to natural language ones, are more structured and can allow explicit modeling of causality (and are easier to evaluate, as opposed to free-form natural language generation). We hope that PROVER will encourage further work towards developing interpretable NLP models with structured explanations.

QA and Proof Consistency: Currently, PROVER predicts the answer and generates the proof by jointly optimizing the QA, node and edge modules using a shared RoBERTa model. Another modeling choice could explicitly condition the QA module on the node and edge modules so that the answer is predicted from the proof. We empirically verify the consistency between the predicted answer and the generated proof by showing that the full accuracy matches the proof accuracy. However, in scenarios where questions have open-ended answers, generating answer from a 'proof' in a consistent manner needs more exploration. PROVER's constraints like ensuring connectivity are necessary constraints for generating valid proofs for any graph-based explanation generation system. However, other tasks may require imposing additional constraints to ensure valid explanations.PROVER's inference mechanism can be extended to incorporate these.

Broader Implications in Formal Logic: PROVER's framework is not conceptually constrained to a particular logic fragment. PROVER uses the idea that applying a rule to fact(s) can produce new fact(s). All logic fragments from formal logic fit this idea and may only differ in the nature of the graphs generated. For a fact "Robin is a bird" and a rule with universal quantification "All birds can fly", PROVER's graph will have an edge from the fact to the rule to generate "Robin can fly". We experiment with datasets which already contain negations in facts. While these datasets currently do not contain disjunctions, our graphical representations of proofs allow an easy extension in such scenarios. E.g., if there is a disjunction rule "If X or Y then Z" instead of a conjunction rule "If X and Y then Z", only the shape of the graph changes. In the former, Z is proved by either an edge from X or from Y to the rule, while in the latter, both edges have to be necessarily present. Inferences over modals like "might" and disjunction rules like "If X then Y or Z" will mean that both the answer and the proof will be probabilistic. In such scenarios, PROVER's unweighted proof graphs can be extended to weighted ones to represent this probabilistic nature.

6 Conclusion

We introduce PROVER, an interpretable joint model that answers binary questions over natural language rule-bases and generates corresponding proofs. The proofs are generated through the node and edge modules of the model in the presence of multiple global constraints during training and ILP inference. Our model improves state-of-the-art QA accuracy in the zero-shot scenario by 6% and generates proofs accurately. PROVER also generalizes much better to higher depth questions with up to 15% absolute improvement in QA performance over RuleTakers. PROVER's modeling is relatively generic, and similar proof generation methods can be explored in traditional multi-hop QA tasks. PROVER can also be a helpful aid to formal reasoners in scenarios where rules are fuzzy and creating rule-bases in a formal language is tedious or infeasible.

Acknowledgements

We thank the reviewers for their helpful feedback. This work was supported by DARPA MCS Grant N66001-19-2-4031, NSF-CAREER Award 1846185, DARPA YFA17-D17AP00022, ONR Grant N00014-18-1-2871, Microsoft Investigator Fellowship, and Munroe & Rebecca Cobey Fellowship. The views in this article are those of the authors and not the funding agency.

References

Ralph Abboud, Ismail Ilkan Ceylan, and Thomas Lukasiewicz. 2020. Learning to reason: Leveraging neural networks for approximate DNF counting. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*.

Ibrahim Abdelaziz, Veronika Thost, Maxwell Crouse, and Achille Fokoue. 2020. An experimental study of formula embeddings for automated theorem proving in first-order logic. *arXiv preprint arXiv:2002.00423*.

Mohit Bansal, Claire Cardie, and Lillian Lee. 2008. The power of negative thinking: Exploiting label disagreement in the min-cut classification framework. In *COLING 2008: Companion Volume: Posters*, pages 15–18.

Regina Barzilay and Mirella Lapata. 2005. Collective content selection for concept-to-text generation. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 331–338. Association for Computational Linguistics.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1533–1544.

Jonathan Berant and Percy Liang. 2014. Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1415–1425.

Oana-Maria Camburu, Tim Rocktäschel, Thomas Lukasiewicz, and Phil Blunsom. 2018. e-SNLI: Natural language inference with natural language explanations. In *Advances in Neural Information Processing Systems*, pages 9539–9549.

Peter Clark, Oyvind Tafjord, and Kyle Richardson. 2020. Transformers as soft reasoners over language. In *International Joint Conference on Artificial Intelligence*.

Ido Dagan, Dan Roth, Mark Sammons, and Fabio Massimo Zanzotto. 2013. Recognizing textual entailment: Models and applications. *Synthesis Lectures on Human Language Technologies*, 6(4):1–220.

Finale Doshi-Velez and Been Kim. 2017. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*.

Shimon Even and R Endre Tarjan. 1975. Network flow and testing graph connectivity. *SIAM journal on computing*, 4(4):507–518.

Peter Hase and Mohit Bansal. 2020. Evaluating explainable AI: Which algorithmic explanations help users predict model behavior? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

Han He and Jinho Choi. 2020. Establishing strong baselines for the new decade: Sequence tagging, syntactic and semantic parsing with bert. In *The Thirty-Third International Flairs Conference*.

Shalmali Joshi, Oluwasanmi Koyejo, Been Kim, and Joydeep Ghosh. 2018. xGEMs: Generating examplars to explain black-box models. *arXiv preprint arXiv:1806.08867*.

Aishwarya Kamath and Rajarshi Das. 2019. A survey on semantic parsing. In *Automated Knowledge Base Construction (AKBC)*.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. 2017. RACE: Large-scale reading comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 785–794.

Guillaume Lample and François Charton. 2020. Deep learning for symbolic mathematics. In *8th International Conference on Learning Representations, ICLR 2020*. OpenReview.net.

Tom Leighton and Satish Rao. 1999. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM (JACM)*, 46(6):787–832.

Kevin Lin, Oyvind Tafjord, Peter Clark, and Matt Gardner. 2019. Reasoning over paragraph effects in situations. In *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*, pages 58–62, Hong Kong, China. Association for Computational Linguistics.

Hui Liu, Qingyu Yin, and William Yang Wang. 2019a. Towards explainable NLP: A generative explanation framework for text classification. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5570–5581.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019b.

RoBERTa: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Bill MacCartney and Christopher D Manning. 2014. Natural logic and natural language inference. In *Computing meaning*, pages 129–147. Springer.

Mark A Musen and Johan Van Der Lei. 1988. Of brittleness and bottlenecks: Challenges in the creation of pattern-recognition and expert-system models. In *Machine Intelligence and Pattern Recognition*, volume 7, pages 335–352. Elsevier.

Arvind Neelakantan, Quoc V. Le, and Ilya Sutskever. 2016. Neural programmer: Inducing latent programs with gradient descent. In *4th International Conference on Learning Representations, ICLR 2016, Conference Track Proceedings*.

Allen Newell and Herbert Simon. 1956. The logic theory machine–a complex information processing system. *IRE Transactions on information theory*, 2(3):61–79.

Bo Pang and Lillian Lee. 2004. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*, page 271. Association for Computational Linguistics.

Nazneen Fatema Rajani, Bryan McCann, Caiming Xiong, and Richard Socher. 2019. Explain yourself! leveraging language models for commonsense reasoning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4932–4942.

Scott E. Reed and Nando de Freitas. 2016. Neural programmer-interpreters. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Anchors: High-precision model-agnostic explanations. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Kyle Richardson, Hai Hu, Lawrence S Moss, and Ashish Sabharwal. 2020. Probing natural language inference models through semantic fragments. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*.

Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215.

Pouya Samangouei, Ardavan Saeedi, Liam Nakagawa, and Nathan Silberman. 2018. ExplainGAN: Model explanation via decision boundary crossing transformations. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 666–681.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. 2019. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations*.

Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. 2019. Learning a SAT solver from single-bit supervision. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Oyvind Tafjord, Matt Gardner, Kevin Lin, and Peter Clark. 2019. QuaRTz: An open-domain dataset of qualitative relationship questions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5941–5946, Hong Kong, China. Association for Computational Linguistics.

Alon Talmor, Yanai Elazar, Yoav Goldberg, and Jonathan Berant. 2019. oLMpics–on what language model pre-training captures. *arXiv preprint arXiv:1912.13283*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. 2019. Learning deep transformer models for machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1810–1822, Florence, Italy. Association for Computational Linguistics.

Leon Weber, Pasquale Minervini, Jannes Münchmeyer, Ulf Leser, and Tim Rocktäschel. 2019. NLProlog: Reasoning with weak unification for question answering in natural language. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6151–6161, Florence, Italy. Association for Computational Linguistics.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. 2015. Towards AI-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, page 658–666. AUAI Press.

Hongming Zhang, Xinran Zhao, and Yangqiu Song. 2020. WinoWhy: A deep diagnosis of essential commonsense knowledge for answering Winograd schema challenge. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*.

A Appendix

A.1 Experimental Setup

We build our model on top of the Hugging Face Transformers library (Wolf et al., 2019).[5] All hyperparamters are chosen based on the best validation set performance (Full Accuracy) of the corresponding dataset. We use RoBERTa-large (Liu et al., 2019b) as the pre-trained Language Model and all our models are trained using a batch size of 8 and a maximum sequence length of 300. We train the models for a maximum of 5 epochs using an initial learning rate of 10^{-5}, with linear decay and a weight decay of 0.1 . The dropout probability is chosen to be 0.1. The random seed used in all the experiments is 42. Each epoch of PROVER takes 2.5 hours to run on one V100 Volta GPU. The total number of parameters of PROVER is similar to that of RoBERTa-large ($355M$). Batch size and learning rate are manually tuned in the range {8,16} and $\{10^{-5}, 2*10^{-5}\}$ respectively. The ILP is modeled using PuLP.[6] Proofs in the datasets are represented as bracketed strings, which are pre-processed into graph representations consisting of unique nodes and edges. The maximum number of facts and rules corresponding to a context is 25.[7]

A.2 Dataset Details

Below we briefly describe the three sets of datasets we conduct experiments on.[8] Each dataset has a train, validation and test split, except for the zero-shot test-only one. Further details about these can be found in Clark et al. (2020).

DU0-DU5: The first set consists of five datasets, each containing 100k questions with theories in synthetic language and requiring reasoning paths up to depth D ($D = 0, 1, 2, 3, 5$). For example, $D = 0$ means the true facts can be proved by simple lookup in the context. The samples are randomly split 70/10/20 into train/dev/test partitions such that there is no overlap of theories between the partitions.

Birds-Electricity: The second set consists of two test-only datasets used to evaluate robustness and out-of-distribution performance of the models. The contexts are about birds and an electric circuit, and consist of 5k samples in total. The vocabulary of entities, attributes and predicates, apart from `is()` are all new at test time.

ParaRules: The final dataset, ParaRules consists of 40k questions against 2k theories expressed in paraphrased natural language, obtained through crowdsourcing. While the previous datasets contain synthetic language, ParaRules tests the models' ability to reason over more human-like paraphrased language.

A.3 QA and Proof Results for Varying Depths

Table 7 shows the DU5 validation set performance of PROVER trained on the training split of DU5. PROVER obtains a near perfect QA accuracy and a proof accuracy of 88%. While the QA accuracy remains equally high at all depths, the proof accuracy drops with increasing depth. Full accuracy matches the proof accuracy, demonstrating consistency between the predicted answers and generated proofs.

A.4 Generalization to Higher Depths

In Table 6, we provide detailed results of PROVER's generalization ability to higher depth questions. Specifically, we evaluate four models, trained on the training splits of DU0, DU1, DU2 and DU3 and tested on the validation and test splits of DU5. We have shown previously that PROVER does significantly better than RuleTakers (Clark et al., 2020) on QA generalization. The proofs, however, do not generalize that well. Note that depth 0 proofs are rather simple (consisting of a

[5] https://github.com/huggingface/transformers

[6] https://pypi.org/project/PuLP/

[7] Further details of our best hyperparameters can be found in the attached code as part of the supplementary material.

[8] https://rule-reasoning.apps.allenai.org/

	QA		NA		EA		PA		FA	
	Dev	Test	Dev	Test	Dev	Test	Dev	Test	Dev	Test
DU0	68.3	68.7	45.3	46.0	49.3	49.5	43.8	44.4	42.3	42.8
DU1	73.2	73.7	66.4	66.3	64.5	64.3	63.9	63.8	61.8	61.9
DU2	89.3	89.6	76.6	76.4	73.1	73.1	72.6	72.6	72.3	72.3
DU3	98.3	98.6	85.5	85.0	79.9	79.5	79.4	79.1	79.4	79.1

Table 6: Performance of PROVER trained on the training splits of DU0, DU1, DU2 and DU3 and tested on the validation and test splits of DU5.

D	Cnt	QA	NA	EA	PA	FA
0	3116	100	98.7	98.6	98.5	98.5
1	2304	98.8	92.5	94.9	92.2	92.2
2	1436	99.2	86.1	85.6	85.6	85.6
3	1165	98.7	85.1	82.8	82.8	82.8
4	1041	98.8	81.2	76.9	76.9	76.9
5	990	99.3	78.3	67.4	67.4	67.4
All	10068	99.3	90.0	88.6	88.0	88.0

Table 7: Performance of PROVER trained on the training split of DU5 and tested on the validation split of DU5.

D	Cnt	QA	NA	EA	PA	FA
0	1485	99.9	99.6	99.7	99.5	99.5
1	1180	99.7	99.3	99.5	99.3	99.3
2	727	99.4	91.5	91.5	91.5	91.3
3	524	98.5	92.0	90.3	90.3	90.3
4	81	100	87.6	72.8	72.8	72.8
5	7	100	100	0	0	0
All	4004	99.6	96.8	96.2	96.1	96.0

Table 8: PROVER results on the ParaRules validation set after training on DU3+ParaRules training splits.

single fact) and a model trained on only such proofs, unsuprisingly, fails to generate proofs for higher depth questions. However, the proof results start improving as the model gets trained on more complex proofs and reaches an accuracy of 79%, when trained on DU3 questions.

A.5 Evaluation on Complex Language

In Table 8, we report the ParaRules validation set results of PROVER trained on the combination of DU3 and ParaRules training splits (following previous work (Clark et al., 2020)). ParaRules is created by first separating the fact groups (a fact group is the set of all facts in the theory concerning a particular person) and the rules from a theory and then asking crowdworkers to paraphrase these in their own words. For example, a fact group "Alan is blue. Alan is rough. Alan is young.", may be reworded into "Alan is on the young side, but rough. He often feels rather blue.". Thus, unlike the previous datasets where the proof graphs are composed of facts and rules, ParaRules proofs are composed of fact groups and rules.[9] PROVER obtains high QA and proof accuracy on complex human-parapharsed rule-bases, showing good generalization on such language. However, the proof accuracy again drops as the depth of the questions increases.

A.6 Ablation Models and Simpler Baselines

We provide brief descriptions of our ablation models. These are (1) **QA+Node:** We model PROVER consisting of only the QA and Node modules. Since there is no edge module, this model does not require any constrained training or inference; (2) **No NAF:** We train a model using random NAF embeddings with no learning. This helps us understand the effectiveness of our NAF learning; (3) **Unconstrained Train + No ILP:** Through this model, we study the effectiveness of our global constraints. Specifically, no edges are masked for training and during inference, the edge labels are predicted based on the model's probability scores only; (4) **Unconstrained Train + ILP:** Here the constraints are employed only during inference. Note that the reverse configuration, constrained training without ILP inference, is not included as the edge logits for the masked out labels would be random (since they are not learned). (5) **No Connectivity:** Finally, we train a model where we only remove the connectivity constraint during ILP optimization, keeping everything else same.

We also experiment with simpler baselines for edge prediction like training a Random Forest with lexical features (BLEU scores, length difference,

[9]The original ParaRules dataset released by Clark et al. (2020) has proofs for the unparaphrased theories (consisting of facts and rules). Using the mapping from a fact to the corresponding fact group, we replace the fact nodes in the proof graph with the corresponding fact group nodes. Note that this is done to report proof accuracy for this dataset as well.

word overlap, etc.) and this obtains a much lower edge accuracy of 47%. This fails primarily because (1) proof graphs can contain NAF which account for 9% of the data and edges from it cannot be learned without learning a latent representation; (2) overlap features are mostly symmetric and hence are not enough for learning directionality; (3) there is lack of overall context information.

A.7 Critical Sentence Identification

Clark et al. (2020) provide an initial solution towards generating explanations for the predicted answers by using a post-hoc method – they remove each fact or rule from the theory and check if the predicted answer changes with the new theory. They define all such rules and facts which flip the answer as critical sentences. If an example has multiple gold proofs, a critical sentence is one which is present in all the proofs. We argue that this leave-one-out analysis is not ideal for multiple reasons - (1) This does not work if the theory has negations, (2) This only predicts the presence or absence of rules and facts, and does not look at the entire chain of reasoning, which our model achieves through the edge module. In our final experiment, we still apply the leave-one-out-strategy on the no-negation subset of the DU5 test set for a direct comparison with RuleTakers. As shown in Table 9, our model identifies the exact critical sentences in an example in 78% of the cases, a 4% improvement over RuleTakers.

	Accuracy	Precision	Recall	F1
RuleTakers	74.5	**98.7**	86.9	92.4
PROVER	**78.1**	**98.7**	**87.2**	**92.6**

Table 9: Comparison of critical sentence identification on the No Negation subset of DU5 test set.

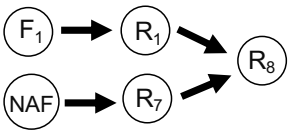
A.8 Proofs Generated by PROVER

In Figure 5, we show two rule-bases, one about electric circuits and another about birds from the Birds-Electricity dataset. PROVER not only answers the questions correctly but also generates the proofs accurately. These proofs are complex because of the presence of NAF and also the long chains of reasoning needed in the inference process. Figure 6 shows three more accurate proofs generated by PROVER for three questions from the DU5 datset.

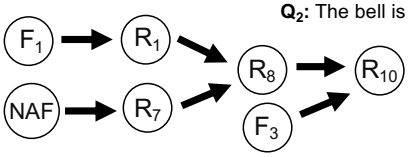
Facts :
F_1: The circuit has the battery. F_2: The switch is on.
F_3: The circuit has the bell.

Rules :
R_1: If the circuit has the battery then the circuit is powered.
R_2: If the circuit does not have the battery then the circuit is dead.
R_3: If the circuit is dead then the bell is not ringing.
R_4: If the circuit is dead then the radio is not playing.
R_5: If the circuit is dead then the light bulb is not glowing.
R_6: If the circuit has the switch and the switch is on then the circuit is complete.
R_7: If the circuit does not have the switch then the circuit is complete.
R_8: If the circuit is powered and the circuit is complete then the current runs through the circuit.
R_9: If the current runs through the circuit and the circuit has the light bulb then the light bulb is glowing.
R_{10}: If the current runs through the circuit and the circuit has the bell then the bell is ringing.
R_{11}: If the current runs through the circuit and the circuit has the radio then the radio is playing.

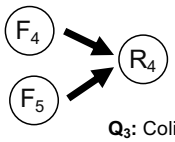
Q_1: The current runs through the circuit. **[Answer : T]**



Q_2: The bell is ringing. **[Answer : T]**

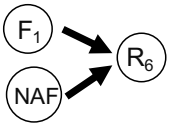


Facts :
F_1: Arthur is a bird. F_2: Arthur is not wounded.
F_3: Bill is an ostrich. F_4: Colin is a bird.
F_5: Colin is wounded. F_6: Dave is not an ostrich.
F_7: Dave is wounded.



Q_3: Colin is not abnormal. **[Answer : F]**

Rules :
R_1: If someone is an ostrich then they are a bird.
R_2: If someone is an ostrich then they are abnormal.
R_3: If someone is an ostrich then they cannot fly.
R_4: If someone is a bird and wounded then they are abnormal.
R_5: If someone is wounded then they cannot fly.
R_6: If someone is a bird and not abnormal then they can fly.



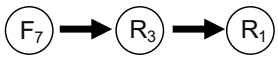
Q_4: Arthur can fly. **[Answer : T]**

Figure 5: Examples of proofs generated by PROVER for four questions on two rule-bases about electric circuits and birds from the Birds-Electricity dataset. PROVER not only answers the questions correctly but also accurately predicts the long reasoning chains with multiple branches.

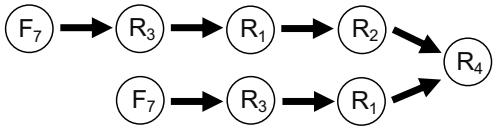
Facts :
F_1: The bear visits the tiger. F_2: The cat is kind.
F_3: The mouse is green. F_4: The mouse is kind.
F_5: The mouse sees the tiger. F_6: The tiger is rough.
F_7: The tiger visits the cat.

Rules :
R_1: If something visits the bear then it sees the bear.
R_2: If something sees the bear then the bear likes the cat.
R_3: If something visits the cat then the cat visits the bear.
R_4: If something sees the bear and the bear likes the cat then it is cold.
R_5: Cold things are rough.
R_6: If something is green and it likes the tiger then the tiger visits the mouse.

Q_1: The cat sees the bear. **[Answer : T]**



Q_2: The cat is not cold. **[Answer : F]**



Q_3: The tiger is cold. **[Answer : F]**

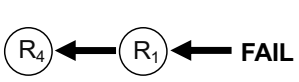


Figure 6: Examples of proofs generated by PROVER for three questions on a rule-base from the DU5 dataset. The proof corresponding to the last question is a failed case.