# RULEBERT: Teaching Soft Rules to Pre-Trained Language Models

**Mohammed Saeed**[†] **Naser Ahmadi**[†] **Preslav Nakov**[‡] **Paolo Papotti**[†]
[†]Eurecom, France [‡]Qatar Computing Research Institute, HBKU, Qatar
{ *FirstName.LastName* }@eurecom.fr, pnakov@hbku.edu.qa

## Abstract

While pre-trained language models (PLMs) are the go-to solution to tackle many natural language processing problems, they are still very limited in their ability to capture and to use common-sense knowledge. In fact, even if information is available in the form of approximate (soft) logical rules, it is not clear how to transfer it to a PLM in order to improve its performance for deductive reasoning tasks. Here, we aim to bridge this gap by teaching PLMs how to reason with soft Horn rules. We introduce a classification task where, given facts and soft rules, the PLM should return a prediction with a probability for a given hypothesis. We release the first dataset for this task, and we propose a revised loss function that enables the PLM to learn how to predict precise probabilities for the task. Our evaluation results show that the resulting fine-tuned models achieve very high performance, even on logical rules that were unseen at training. Moreover, we demonstrate that logical notions expressed by the rules are transferred to the fine-tuned model, yielding state-of-the-art results on external datasets.

## 1 Introduction

Pre-trained language models (PLMs) based on transformers (Devlin et al., 2019; Liu et al., 2020) are established tools for capturing both linguistic and factual knowledge (Clark et al., 2019b; Rogers et al., 2020). However, even the largest models fail on basic reasoning tasks. If we consider common relations between entities, we see that such models are not aware of negation, inversion (e.g., *parent-child*), symmetry (e.g., *spouse*), implication, and composition. While these are obvious to a human, they are challenging to learn from text corpora as they go beyond linguistic and factual knowledge (Ribeiro et al., 2020; Kassner and Schütze, 2020). We claim that such reasoning primitives can be transferred to the PLMs by leveraging logical rules, such as those shown in Figure 1.

**Input facts**:
*Mike* is the parent of **Anne**. **Anne** lives with Mark.
**Anne** is the child of Laure. **Anne** lives with *Mike*.
**Input rules**:
($r_1$, .1) Two persons living together are married.
($r_2$, .7) Persons with a common child are married.
($r_3$, .9) Someone cannot be married to his/her child.
($r_4$, 1) Every person is the parent of his/her child.

**Test 1**: Laure and *Mike* are married.
**Answer**: *True* with probability 0.7 [$r_4, r_2$]

**Test 2**: **Anne** and Mark are married.
**Answer**: *False* with probability 0.9 [$r_1$]

**Test 3**: **Anne** and *Mike* are married.
**Answer**: *False* with probability 0.9 [$r_1, r_3, r_4$]

Figure 1: Examples of hypotheses that require reasoning using facts and possibly conflicting soft rules (rule id and confidence shown in brackets).

While there have been initial attempts to teach reasoning with rules to PLMs (Clark et al., 2020; Kassner et al., 2020), such approaches model only a subclass of logical rules. In fact, current solutions focus on exact rules, i.e., rules that hold in all cases. In reality, most of the rules are approximate, or soft, and thus have a certain confidence of being correct. For example, across the 7,015 logical rules defined on the DBpedia knowledge graph, only 11% have a confidence above 95%. In the example, rules $r_1$–$r_3$ are soft, i.e., cover knowledge that is not true in all circumstances. Consider rule $r_2$, stating that if two persons have a child in common, they are most likely married. As $r_2$ has a confidence of being correct of 0.7, this uncertainty is reflected in the probability of the prediction.

With the above considerations in mind, here we show how to reason over soft logical rules with PLMs. We provide facts and rules expressed in natural language, and we ask the PLM to come up with a logical conclusion for a hypothesis, together with the probability for it being true.

Unlike previous approaches (Clark et al., 2020), we enable deductive reasoning for a large class of soft rules with binary predicates and an unrestricted number of variables. Our model can even reason over settings with conflicting evidence, as shown in Test 3 in Figure 1. In the example, as Anne and Mike live together, they have a 0.1 probability of being married because of soft rule $r_1$. However, we can derive from exact rule $r_4$ that Anne is the child of Mike and therefore they cannot be married, according to soft rule $r_3$.

To model uncertainty, we pick one flavor of probabilistic logic programming languages, $\text{LP}^{\text{MLN}}$, for reasoning with soft rules (Lee and Wang, 2016). It assigns weights to stable models, similarly to how Markov Logic assigns weights to models. However, our method is independent of the logic programming approach at hand, and different models can be fine-tuned with different programming solutions. Our proposal makes use of synthetic examples that "teach" the desired formal behavior through fine-tuning. In particular, we express the uncertainty in the loss function used for fine-tuning by explicitly mimicking the results for the same problem modeled with $\text{LP}^{\text{MLN}}$.

Our contributions can be summarized as follows:

- We introduce the problem of teaching soft rules expressed in a synthetic language to PLMs through fine-tuning (modeled as binary classification).
- We create and release the first dataset for this task, which contains 3.2M examples derived from 161 rules describing real common-sense patterns with the target probability for the task obtained from a formal reasoner (Section 4).
- We introduce techniques to predict the correct probability of the reasoning output for the given soft rules and facts. Our solution relies on a revised loss function that effectively models the uncertainty of the rules (Section 5). Our approach handles multi-variable rules and nicely extends to examples that require reasoning over multiple input rules.
- We show that our approach enables fine-tuned models to yield prediction probability very close to that produced by a formal reasoner (Section 6). Our PLM fine-tuned on soft rules, RuleBERT, can effectively reason with facts and rules that it has not seen at training, even when fine-tuned with only 20 rules.
- We demonstrate that our fine-tuning approach effectively transfers knowledge about predicate negation and symmetry to the lower levels of the transformer, which benefits from the logical notions in the rules. In particular, RuleBERT achieves new state-of-the-art results on three external datasets.

The data, the code, and the fine-tuned model are available at `http://github.com/MhmdSaiid/RuleBert`.

## 2 Related Work

PLMs have been shown to have some reasoning capabilities (Talmor et al., 2020b), but fail on basic reasoning tasks (Talmor et al., 2020a) and are inconsistent (Elazar et al., 2021), especially when it comes to negation (Kassner and Schütze, 2020).

Our work focuses on deductive reasoning. Note that it is different from previous work, e.g., on measuring the factual knowledge of PLMs (Petroni et al., 2019), on probing the commonsense capabilities of PLMs at the token or at the sentence level (Zhou et al., 2020), or on testing the reasoning capabilities of PLMs on tasks such as age comparison and taxonomy conjunction (Talmor et al., 2020a). Our work relates to Task #15 in the bAbI dataset (Weston et al., 2016) and to RuleTakers (Clark et al., 2020). However, we differ (*i*) by using a larger subclass of first-order logic rules (with more variables and various forms), and (*ii*) by incorporating soft rules.

Our proposal is different from work on Question Answering (QA) with implicit reasoning based on common-sense knowledge (Clark et al., 2019a), as we rely purely on deductive logic from explicitly stated rules.

Our approach also differs from methods that semantically parse natural language into a formal representation on which a formal reasoner can be applied (Liang, 2016), as we directly reason with language. Yet, we are also different from Natural Language Inference (NLI) and textual entailment, which work with text directly, but cannot handle Horn rules (MacCartney and Manning, 2009; Dagan et al., 2013).

Unlike previous work (Yang et al., 2017; Hamilton et al., 2018; Minervini et al., 2020), we do not design a new, ad-hoc module for neural reasoning, but we rely solely on the transformer's capability to emulate algorithms (Wang et al., 2019b; Lample and Charton, 2020).

## 3 Background

**Language Models.** We focus on language models pre-trained with bidirectional transformer encoders using masked language modeling (Devlin et al., 2019). For fine-tuning, we create examples for sequence classification to teach the models how to emulate reasoning given facts and soft rules.

**Logical Rules.** We rely on existing corpora of declarative *Horn rules* mined from large RDF knowledge bases (KBs) (Galárraga et al., 2015; Ortona et al., 2018; Ahmadi et al., 2020). An RDF KB is a database representing information with triples (or *facts*) $p(s, o)$, where a *predicate* $p$ connects a *subject* $s$ and an *object* $o$. An atom in a rule is a predicate connecting two universally quantified variables. A Horn rule (or clause) has the form: $B \rightarrow h(x, y)$, where $h(x, y)$ is a single atom (head or conclusion of the rule) and $B$ (body or premise of the rule) is a conjunction of atoms. Positive rules identify relationships between entities, e.g., $r_1$, $r_2$, $r_4$ in Figure 1. Negative rules identify contradictions, e.g., $r_3$ in Figure 1. Rules can contain predicates comparing numerical values, such as $<$. For example, negative rule $r_5$: *birthYear(b,d)* $\wedge$ *foundYear(a,c)* $\wedge$ *<(c,d)* $\rightarrow$ *negfounder(a,b)* states that any person (variable $b$) with a birth year ($d$) higher than the founding year ($c$) of a company ($a$) cannot be its founder. A fact is derived from a rule if all the variables in the rule body are replaced with constants from facts. For $r_5$, facts "foundYear(Ford,1903), birthYear(E. Musk,1971), >(1971,1903)" trigger the rule that derives the fact *negFounder*(E. Musk, Ford).

**Rule Confidence.** *Exact* rules, such as $r_4$, apply in all cases, without exception. However, most rules are approximate, or soft, as they apply with a certain likelihood. For example, $r_3$ in Figure 1 is true in most cases, but there are historical exceptions in royal families. Rules are annotated with a measure of this likelihood, either manually or with a computed *confidence* (Galárraga et al., 2015).

**Probabilistic Answer Set Programming.** As we deal with soft rules, we adopt $\text{LP}^{\text{MLN}}$ (Lee and Wang, 2016) to create the dataset. $\text{LP}^{\text{MLN}}$ is a probabilistic extension of answer set programs (ASP) with the concept of weighted rules from Markov Logic (Baral, 2010). In ASP, search problems are reduced to computing *stable models* (answer sets), a set of beliefs described by the program. A weight (or confidence) is assigned to each rule, so that the more rules a stable model satisfies, the larger weight it gets, and the probability of the stable model is computed by normalizing its weight among all stable models. Given a set of soft rules and facts, we measure how much the hypothesis is supported by the stable model.

## 4 Dataset

We start by defining the reasoning task. We then discuss example generation methods for three scenarios: single rule as input, multiple (possibly conflicting) rules that require reasoning for the same conclusion, and multiple rules that require a sequence (chain) of reasoning steps. Examples of the data generation procedures are in the Appendix.

### 4.1 Reasoning Task

Each example is a triple *(context, hypothesis, confidence)*. *Context* is a combination of rule(s) and generated facts, such as "*If the first person lives together with the second person, then the first person is the spouse of the second person.*" and "*Anne lives with Mike.*" *Hypothesis* is the statement to be assessed based on the context, e.g., "*Laure is the spouse of Mike.*" *Confidence* is the probability that the hypothesis is valid given by the reasoner, e.g., 0.7. As we generate the examples, we know the confidence for each hypothesis.

### 4.2 Single-Rule Dataset Generation

Given a rule, we generate examples of different hypotheses to expose the model to various contexts. Each example contains the context $c$ and a hypothesis $h$ with its probability of being true as obtained for the $(c, h)$ pair from the $\text{LP}^{\text{MLN}}$ reasoner. The intuition is that the examples show the expected behavior of a formal reasoner for every combination of possible facts for a given rule. This process is not about teaching the model specific facts to recall later, but teaching it reasoning patterns.

Unlike previous work (Clark et al., 2020), our rules allow for multiple variables. This introduces additional complexity as examples must show how to deal with the symmetry of the predicate. For example, *child(Alice,Bob)* and *child(Bob,Alice)* are not equivalent since *child* is not symmetric, while *spouse(Alice,Bob)* and *spouse(Bob,Alice)* are equivalent as *spouse* is symmetric. We assume that metadata about the symmetry and the types is available from the KB for the predicates in the rules.

Given as input (*i*) a rule $r$, (*ii*) a desired number $n$ of examples, (*iii*) an integer $m$ to indicate the maximum number of facts given as a context, and (*iv*) a pool of values for each type involved in $r$'s predicate *pools*, Algorithm 1 outputs a dataset $D$ of generated examples.

**Algorithm 1:** Generate Synthetic Data

**Input:** rule $r$ ; `// child(a,b)→parent(b,a)`
$n$ ; `// # of examples`
$m$ ; `// max # of facts`
*pools* ; `// pools of names`
**Output:** Generated Dataset $D$

1 $D = \{\}, i = 1$ ; `// initialize`
2 **while** $i \leq ceiling(n/8)$ **do**
3 $F = GenFacts(r, m, pools)$ ; `// child(Eve,Bob),parent(Amy,Sam)`
4 $O = LPMLN(r, F)$ ; `// reasoner output`
5 $h_1 = f \in F$ ; `// child(Eve,Bob)`
6 $h_2 = Alter(f)$ ; `// negchild(Eve,Bob)`
7 $h_3 = r(F)$ ; `// parent(Bob,Eve)`
8 $h_4 = Alter(r(F))$ ; `// parent(Eve,Bob)`
9 $h_5 = \text{pos}.f_l \notin F$ ; `// child(Joe,Garry)`
10 $h_6 = \neg h_5$ ; `// negchild(Joe,Garry)`
11 $h_7 = f_r \notin O$ ; `// parent(Alice,Joe)`
12 $h_8 = \neg h_7$ ; `// negparent(Alice,Joe)`
13 $D.add(h_{1-8})$;
14 $i \leftarrow i + 1$;
15 **Function** `GenFacts(r,m,pools)` **:**
16 $F = GetRandomFacts(r, pools, m)$;
17 $F.add(GetRuleFacts(r, pools))$;
18 **return** F
19 **Function** `Alter(`$p(s,o)$`)` **:**
20 **if** *p is symmetric* **then return** $\neg p(s, o)$ ;
21 **if** *random()>0.5* **then return** $\neg p(s, o)$ ;
22 **else return** $p(o, s)$ ;

We start at line 3 by generating facts, such as *child(Eve,Bob)*, using the function *GenFacts* (lines 15–18), which takes as input a rule $r$, the maximum number of facts $m$ to generate, and the *pools*. A random integer less than $m$ sets the number of facts in the current context. The generated facts $F$ have predicates from the body of $r$, their polarity (true or negated atom) is assigned randomly, and variables are instantiated with values sampled from the pool (line 16). Facts are created randomly, as we are not interested in teaching the model specific facts to recall later, but instead we want to teach it how to reason with different combinations of rules and facts. We then ensure that the rule is triggered in every context, eventually adding more facts to $F$ using the function *GetRuleFacts* in line 17. After obtaining $F$, we feed rule $r$ along with facts $F$ to the $\text{LP}^{\text{MLN}}$ reasoner, and we obtain a set $O$ containing all satisfied facts and rule conclusions (line 4).

We generate different hypotheses, where each one leads to an example in dataset $D$. For each context, we add an example with different facts with respect to the given rule according to three dimensions. A fact can be (*i*) for a predicate in the premise or in the conclusion of a rule, could be (*ii*) satisfied or unsatisfied given the rule, and could have (*iii*) positive or negative polarity. This makes eight different possibilities, thus leading to the generation of eight different hypotheses (one for each context).

The first hypothesis $h_1$ is obtained by sampling a fact from the set $F$ (line 5). We then produce the counter hypothesis $h_2$ by altering the fact (line 6) using the function *Alter* (lines 19-22). Given a hypothesis $p(s, o)$ (line 19), we return its negated form if $p$ is symmetric (line 20). Otherwise, if $p$ is not symmetric, we produce a counter hypothesis either by negation (line 21), or by switching the subject and the object in the triple as the predicate is not symmetric (line 22). We rely on a dictionary to check whether a predicate is symmetric or not.

We then produce hypothesis $h_3$ (line 7), which is the outcome of triggering rule $r$ with the facts added in line 17. The counter hypothesis $h_4$ is generated by altering $h_3$ (line 8). Moreover, we generate hypothesis $h_5$ by considering any unsatisfied positive fact outside $F$. Following a closed-world assumption (CWA), we assume that positive triples are false if they cannot be proven, meaning that their negation is true. We sample a fact $f_l$ from the set of all possible positive facts that do not have the same predicate of the rule head (line 9). Thus, $h_5$ will never be in the output $O$ of the reasoner, as it cannot be derived. We then produce $h_6$ by negating $h_5$ in line 10. We further derive $h_7$ by sampling a fact $f_r$ that has the same predicate as that of the rule head, but does not belong to the output of the reasoner $O$ (line 11). For a positive (negative) rule, such a fact is labelled as False (True). $h_7$ is then negated to get the counter hypothesis $h_8$ (line 12). All generated hypotheses are added to $D$ (line 13), and the process repeats until we obtain $n$ examples.

Finally, we automatically convert the examples to natural language using predefined templates. A basic template for atom predicate $p$ (type $t_1$, type $t_2$) is "If the $1^{st}$ $t_1$ is $p$ of the $2^{nd}$ $t_2$." ("If the first person is spouse of ..."). For the single-rule scenario, we release a dataset for 161 rules with a total of 3.2M examples and a 80%:10%:10% split for training, validation, and testing.

### 4.3 Rules with Overlapping Conclusion

When multiple rules are in the context, there could be facts that trigger more than one rule for a given hypothesis. The triggered rules might be all of the same polarity (positive or negative), eventually accumulating their confidence, or could be a mix of positive and negative rules that oppose each other. While the data generation procedure in Section 4.2 can be extended to handle multiple rules, this raises an efficiency problem. Given a set of $R$ rules, it would generate $8^{|R|}$ examples for each *(facts,rule)* pair in order to cover all rule combinations. This is very expensive, e.g., for five rules, it would generate $8^5 = 32,768$ examples for a single context.

Given this challenge, we follow a different approach. We first generate data for each rule individually using Algorithm 1. We then generate more examples only for combinations of two or more rules having *all their rule conclusions* as hypotheses. For every input context, we produce rule-conclusion hypotheses (positive and negative) while varying the rules being fired. Thus, we generate $2 * \sum_{x=2}^{|R|} \binom{|R|}{x}$ examples with at least two rules triggered. Adding the single-rule data, we generate $8 * |R| + 2 * \sum_{x=2}^{|R|} \binom{|R|}{x}$ for every *(facts,rules)* pair, which is considerably smaller than $8^r$ for $|R| \geq 2$, according to the binomial theorem. For example, for $|R|$=5, we generate 92 examples per context. For the overlapping rules scenario, we release a dataset for 5 rules with a total of 300K examples, and a 70%:10%:20% split for training, validation, and testing.

### 4.4 Chaining of Rule Executions

For certain hypotheses, an answer may be obtained by executing rules in a sequence, i.e., one on the result of the other, or in a *chain*. To be able to evaluate a model in this scenario, we generate hypotheses that can be tested only by chaining a number of rules (an example is shown in Appendix H). Given a pool of rules over different relations and a depth $D$, we sample a chain of rules with length $D$. We then generate hypotheses that would require a depth varying between 0 and $D$. We generate a rule-conclusion hypothesis ($h_3$) and its alteration ($h_4$) for each depth $d \leq D$. A depth of 0 means that the hypothesis can be verified using the facts alone without triggering any rule. We also generate counter-hypotheses by altering the hypotheses at a given depth, and we further include hypotheses that are unsatisfied given the input.

For the chaining rules scenario, we start with a pool of 64 soft rules, and we generate hypotheses that would need at most five chained rules to verify them. The dataset for $d \leq 5$ contains a total of 70K examples, and a 70%:10%:20% split for training, validation, and testing.

## 5 Teaching PLMs to Reason

In this section, we explain how we teach a PLM to reason with one or more soft rules. Note that uncertainty stems from the rule confidence. One approach to teach how to estimate the probability of a prediction is to treat each confidence value (or bucket of confidence values) as a class and to model the problem as a $k$-way classification instance (or regression), but this is intractable when multiple rules are considered. Instead, we keep the problem as a two-class one by altering how the information is propagated in the model to incorporate uncertainty from the rule confidence.

Let $D = \{(x_i, y_i)\}_{i=1}^{m}$ be our generated dataset, where $x_i$ is one example of the form *(context,hypothesis,confidence)* and $y_i$ is a label indicating whether the hypothesis is validated or not by the context (facts and rules in English), and $m$ is the size of the training set. A classifier $f$ is a function that maps the input to one of the labels in the label space. Let $h(x, y)$ be a classification loss function. The empirical risk of the classifier $f$ is

$$R_h(f) = \mathbb{E}_D(h(x, y)) = -\frac{1}{m}\sum_{i=1}^{m} h(x_i, y_i)$$

We want to introduce uncertainty in our loss function, using the weights computed by the $\text{LP}^{\text{MLN}}$ solver as a proxy to represent the probability of predicting the hypothesis as being true. To do so, we apply a revised empirical risk:

$$R'_h(f) = \mathbb{E}_D(h(x, y)) =$$
$$-\frac{1}{m}\sum_{i=1}^{m}(w(x_i) * h(x_i, 1) + (1 - w(x_i)) * h(x_i, 0))$$

where $w(x_i)$ is the probability of $x_i$ being True.

We now state that each example is considered as a combination both of a weighted positive example with a weight $w(x_i)$ provided by the $\text{LP}^{\text{MLN}}$ solver and a weighted negative example with a weight $1 - w(x_i)$.

When trained to minimize this risk, the model learns to assign the weights to each output class, thus predicting the confidence for the true class when given the satisfied rule head as a hypothesis.

| Dataset | Total | Train | Dev | Test |
|---|---|---|---|---|
| Single Rule (Section 6.2) | 20K | 16K | 2K | 2K |
| Overlap (Section 6.3) | 300K | 210K | 30K | 60K |
| Chaining (Depth=5) (Section 6.4) | 70K | 56K | 4.6K | 9.4K |
| RULEBERT (Section 7) | 3.2M | 2.56M | .32M | .32M |

Table 1: Datasets for the experiments and their splits.

## 6 Experiments

We first describe the experimental setup (Section 6.1). We then evaluate the model on single (Section 6.2) and on multiple rules (Sections 6.3 and 6.4). We show that a PLM fine-tuned on soft rules, namely RULEBERT, makes accurate predictions for unseen rules (Section 6.5), and it is more consistent than existing models on three external datasets (Section 7). We report the values of the hyper-parameters, as well as the results for some ablation experiments in the Appendix. The datasets for all experiments are summarized in Table 1.

### 6.1 Experimental Setup

**Rules.** We use a corpus of 161 soft rules mined from DBpedia. We chose a pool of distinct rules with varying number of variables, number of predicates, rule conclusions, and confidences.

**Reasoner.** We use the official implementation[1] of the $\text{LP}^{\text{MLN}}$ reasoner. We set the reasoner to compute the exact probabilities for the triples.

**PLM.** We use the HuggingFace pre-trained *RoBERTa*$_{LARGE}$ (Liu et al., 2020) model as our base model, as it is trained on more data compared to *BERT* (Devlin et al., 2019), and is better at learning positional embeddings (Wang and Chen, 2020). We fine-tune the PLM[2] with the weighted binary cross-entropy (wBCE) loss from Section 5. More details can be found in Appendix C.

**Evaluations Measures.** For the examples in the test set, we use accuracy (Acc) and F1 score (F1) for balanced and unbalanced settings, respectively. As these measures do not take into account the uncertainty of the prediction *probability*, we further introduce Confidence Accuracy@k (CA@k), which measures the proportion of examples whose absolute error between the predicted and the actual probabilities is less than a threshold $k$:

$$CA@k = \frac{\#\{x_i, |w_i - \hat{w}_i| < k\}}{\#\{x_i\}}$$

where $x_i$ is the i$^{th}$ example of dataset, $w_i$ is the actual confidence of the associated hypothesis given by the $\text{LP}^{\text{MLN}}$ reasoner, $\hat{w}_i$ is the predicted confidence by the model, and $k$ is a chosen threshold.

The measure can be seen as the ordinary accuracy measure, but true positives and negatives are counted only if the condition is satisfied, where lower values for $k$ indicate stricter evaluation.

### 6.2 Single Soft Rule

We fine-tune 16 models for 16 different positive and negative rules (one model per rule) using 16k training samples per rule. We compare the accuracy of each model (*i*) without teaching uncertainty using binary cross-entropy (RoBERTa), and (*ii*) with teaching soft rules using wBCE.

**Results.** Every row in Table 2 shows a rule with its confidence, followed by accuracy and CA@$k$ (for $k = 0.1$ and $k = 0.01$) for both loss functions. We see that models fine-tuned using *RoBERTa-wBCE* perform better on CA@k. In terms of *accuracy*, both models perform well, with *RoBERTa-wBCE* performing better for all rules. Interestingly, the best performing rules are two rules that involve comparison of numerical values (birth years against death and founding years), which suggests that our method can handle comparison predicates.

### 6.3 Rules Overlapping on Conclusion

The dataset contains five soft rules with *spouse* or *negspouse* in the rule conclusion, and a confidence between 0.30 and 0.87 (shown in Figure 2). We train a model on the dataset and test it (*i*) on a test set for each of the five rules separately, (*ii*) on test sets with $U$ triggered rules, where $U \in \{2, 3, 4, 5\}$.

($r_1$, .87) child(a,c) $\wedge$ parent(c,b) $\rightarrow$ spouse(A,B)
($r_2$, .64) child(a,b) $\rightarrow$ negspouse(a,b)
($r_3$, .3) relative(a,b) $\rightarrow$ spouse(a,b)
($r_4$, ,78) child(a,c) $\wedge$ child(b,c) $\rightarrow$ spouse(a,b)
($r_5$, .67) predecessor(a,b) $\rightarrow$ negspouse(a,b)

Figure 2: The five overlapping soft rules.

**Results.** Table 3 shows that the model achieves high scores both on the single test sets (top five rows) and on the sets with interacting rules. The test sets with $U = 2$ and $U = 3$ are most challenging, as they contain $\binom{5}{2} = 10$ and $\binom{5}{3} = 10$ combinations of rules, respectively, while the one with $U = 5$ has only one possible rule combination. The high scores indicate that PLMs can actually learn the interaction between multiple soft rules.

[1] http://github.com/azreasoners/lpmln
[2] The prompt is <s>*context*</s></s>*hypothesis*</s>.

| Rule | Conf. | *RoBERTa-wBCE* Acc. | CA@k .10 | .01 | *RoBERTa* Acc. | CA@k .10 | .01 |
|---|---|---|---|---|---|---|---|
| birthYear(a,c) ∧ deathYear(b,d) ∧ >(c,d)→negspouse(a,b) | .990 | .995 | .993 | .993 | .970 | .490 | .486 |
| birthYear(b,d) ∧ foundYear(a,c) ∧ <(c,d)→negfounder(a,b) | .990 | .928 | .927 | .927 | .908 | .486 | .456 |
| spouse(c,a) ∧ parent(b,c) → negspouse(a,b) | .923 | .974 | .963 | .747 | .875 | .491 | .279 |
| relative(a,c) ∧ spouse(b,c) ∧ child(b,a) → relative(a,b) | .860 | .922 | .844 | .801 | .866 | .342 | .146 |
| parent(c,a) ∧ child(b,c) → spouse(a,b) | .825 | .944 | .828 | .444 | .842 | .342 | .146 |
| publisher(c,b) ∧ subsequentWork(c,a) → publisher(a,b) | .721 | .909 | .834 | .765 | .905 | .358 | .219 |
| successor(b,a) → negspouse(a,b) | .718 | .972 | .896 | .693 | .949 | .369 | .313 |
| child(c,b) ∧ relative(c,a) → negchild(a,b) | .644 | .935 | .880 | .693 | .905 | .310 | .303 |
| child(c,b) ∧ spouse(a,c) → negrelative(a,b) | .562 | .920 | .907 | .608 | .915 | .255 | .250 |
| relation(a,b) → negchild(a,b) | .549 | .904 | .886 | .737 | .902 | .371 | .366 |
| child(c,b) ∧ spouse(c,a) → child(a,b) | .492 | .901 | .827 | .422 | .658 | .223 | .107 |
| knownFor(b,a) → founder(a,b) | .387 | .882 | .601 | .477 | .839 | .372 | .215 |
| founder(c,b) ∧ publisher(c,a) → negfounder(a,b) | .246 | .886 | .795 | .665 | .802 | .311 | .297 |
| publisher(a,c) ∧ parentCompany(b,c) → negpublisher(a,b) | .235 | .812 | .748 | .643 | .811 | .313 | .271 |
| successor(c,a) ∧ spouse(c,d) ∧ successor(d,b)→spouse(a,b) | .221 | .927 | .738 | .628 | .761 | .248 | .215 |
| relative(a,c) ∧ parent(c,b) → child(a,b) | .135 | .841 | .704 | .552 | .727 | .227 | .182 |

Table 2: Evaluation results for single-rule models.

| Test | Size | F1 | CA@.15 | CA@.1 | CA@.05 |
|---|---|---|---|---|---|
| $r_1$ | 1.6k | .990 | .987 | .986 | .954 |
| $r_2$ | 1.6k | .999 | .997 | .996 | .946 |
| $r_3$ | 1.6k | .995 | .994 | .994 | .992 |
| $r_4$ | 1.6k | .990 | .989 | .988 | .935 |
| $r_5$ | 1.6k | 1 | .999 | .998 | .979 |
| U=2 | 20k | .985 | .997 | .993 | .968 |
| U=3 | 20k | .925 | 1 | .998 | .949 |
| U=4 | 10k | .956 | 1 | 1 | .988 |
| U=5 | 2k | 1 | 1 | 1 | .980 |

Table 3: Results for a model trained on five rules sharing the same predicate, and tested on multiple test sets.

| Data | Mod0 | Mod1 | Mod2 | Mod3 | Mod4 | Mod5 |
|---|---|---|---|---|---|---|
| Test | .996 | .926 | .883 | .852 | .856 | .831 |
| All | .589 | .743 | .772 | .811 | .831 | .831 |
| Dep0 | .993 | .974 | .973 | .982 | .978 | .973 |
| Dep1 | .264 | .860 | .884 | .887 | .889 | .889 |
| Dep2 | .396 | .655 | .730 | .751 | .750 | .720 |
| Dep3 | .438 | .581 | .636 | .684 | .690 | .656 |
| Dep4 | .538 | .468 | .547 | .626 | .666 | .627 |
| Dep5 | .552 | .356 | .496 | .703 | .785 | .744 |

Table 4: F1 scores for models trained on varying depths and tested on six datasets. The boxed area indicates models tested on unseen chaining depths.

### 6.4 Rule Chaining

Here, we assess models fine-tuned on various chaining depths. We construct six datasets for this scenario with increasing depths ($D = 0$, $D \leq 1$, $D \leq 2$, $D \leq 3$, $D \leq 4$, $D \leq 5$), i.e., dataset $D \leq x$ contains hypotheses that need at most $x$ chained rules. We thus train six models (one per dataset), and we test them (*i*) on their own test dataset (Test), (*ii*) on the test set with $D \leq 5$ that contains all examples up to depth 5 (All), and (*iii*) on test sets with a chaining of depth $x$ (Dep$x$).

**Results.** The results are shown in Table 4. We can see that the models achieve high F1 scores on the respective test sets for Depth 0. The red borderline indicates F1 scores for models tested on chaining depths higher than the ones they have been trained on. We see that Mod3 and Mod4 do fairly well on Depth 5. However, there is a decrease for higher depths, possibly due to the need for more training examples in order to learn such depths.

Moreover, since we sample a chain of rules each time, it is likely that every model has been trained on certain chains of rules. This yields lower scores in the constant-depth test sets as the models are being tested on unseen rule chains.

Note that Mod0 shows a counter-intuitive increase in the F1 score for higher unseen depths. Chaining soft rules may lead to a low probability for the associated hypothesis, and thus eventually to a *False* label. However, Mod0 is not trained on chaining and sees a hypothesis that requires chaining as an unsatisfied fact, thus eventually labelling it as *False*, while in fact it is the chaining of the soft rules that is the cause for this label. This is never the case with hard rules, as the actual label there would be *True*.

### 6.5 Testing RuleBERT on Unseen Rules

We have seen that a PLM can be successfully fine-tuned with rules. We now study the performance on the PLM after it has been fine-tuned on 161 (single) rules. We call this fine-tuned model RuleBERT.

| | Rule | *FT-PLM* | RULEBERT$_{20}$ | *FT-*RULEBERT$_{20}$ |
|---|---|---|---|---|
| Known predicates | child(a,b) → parent(b,a) | .719 | .869 | .989 |
| | relative(a,b) → negspouse(b,a) | .885 | .885 | .963 |
| | child(a,b) ∧ child(b,c) → negchild(a,c) | .835 | .888 | .918 |
| | parent(a,b) ∧ parent(a,c) → spouse(b,c) | .754 | .757 | .814 |
| | parent(a,b) → negchild(a,b) | .923 | .933 | .963 |
| Unknown predicates | knownFor(b,a) → founder(a,b) | .817 | .795 | .971 |
| | worksFor(b,a) → negfounder(a,b) | .951 | .915 | .952 |
| | occupation(a, b) → negalmaMater(a, b) | .939 | .917 | .972 |
| | author(c,b) ∧ series(a,c) → author(a,b) | .965 | .937 | .989 |
| | city(a,b) → negstate(a,b) | .923 | .912 | .971 |

Table 5: Evaluation on unseen rules (accuracy). The first group contains rules with predicates seen by RULEBERT among the 20 rules used for fine-tuning, while the second group has rules with unseen predicates.

child(a,b) → negparent(a,b)
child(a,b) → nespouse(a,b)
child(a,b) → negchild(b,a)
child(a,b) → negrelation(b,a)
parent(a,b) → negparent(b,a)
parent(a,b) → nespouse(a,b)
spouse(a,b) → relative(b,a)
successor(a,b) → predecessor(b,a)
predecessor(a,b) → negsuccessor(a,b)
successor(a,b) → negspouse(a,b)
predecessor(a,b) → negspouse(a,b)
child(a,c) ∧ parent(c,b) → spouse(a,b)
child(b,a) ∧ child(c,a) → spouse(b,c)
parent(a,b) ∧ parent(b,c) → negparent(a,c)
parent(a,b) ∧ child(c,a) → spouse(b,c)
spouse(a,b) ∧ parent(c,a) → negspouse(b,c)
spouse(a,b) ∧ child(a,c) → negspouse(b,c)
successor(a,c) ∧ successor(b,c) → negspouse(a,b)
publisher(c,b) ∧ subsequentwork(c,a) → publisher(a,b)
publisher(c,b) ∧ previouswork(c,a) → publisher(a,b)

Figure 3: The 20 random rules used for RULEBERT$_{20}$.

We first evaluate RULEBERT on unseen rules. We fine-tune it with only twenty randomly selected rules (shown in Figure 3) and call it RULEBERT$_{20}$. We then select ten new rules divided into two groups: (*i*) five rules containing predicates that were used in the rules for fine-tuning RULEBERT$_{20}$, and (*ii*) five rules that share no predicates with the fine-tuning rules. For each rule in the test sets, we run a model fine-tuned (with 4k examples) only for that rule (FT-PLM), the model fine-tuned on the twenty original rules (RULEBERT$_{20}$), and the same model fine-tuned again for the rule at hand (FT-RULEBERT$_{20}$).

**Results.** Table 5 shows that RULEBERT$_{20}$ outperforms the fine-tuned model (FT-PLM) on the first group. Even though fine-tuned on 20 rules, it learned enough about (*i*) symmetric/transitive predicates and (*ii*) rule confidence to predict correctly, even better than rule-specific models.

For the second rule group, the accuracy of RULEBERT$_{20}$ is high, but FT-PLM performs better. Applying the same fine-tuning on RULEBERT$_{20}$ yields the best results in all scenarios.

## 7 RULEBERT on External Datasets

As our fine-tuning propagates information in the layers of the encoder, we hypothesize that RULEBERT effectively "learns" logical properties of the concepts represented in the rules, such as negation and symmetry, and thus it could perform better on tasks testing such properties of PLMs. To study the negation of predicates, we use the *Negated LAMA datasets*, which test how PLMs distinguish a Cloze question and its negation (Kassner and Schütze, 2020). In most cases, PLMs make the same prediction both for a positive statement ("*Relativity was developed by Einstein.*") and for its negation ("*Relativity was not developed by Einstein.*"). To test the symmetry relationship between predicates, we use the SRL test in *CheckList* (Ribeiro et al., 2020), which focuses on behavioral testing of NLP models; we use its test set for the duplicate-question detection task (QQP) (Wang et al., 2019a). Finally, we test deductive reasoning on the *bAbI* dataset and its Task #15 (Weston et al., 2016).

### 7.1 Negated LAMA Experiments

For Negated LAMA, we do not fine-tune RULEBERT for the task; instead, we replace its original classification layer by an MLM head with weights identical to those of RoBERTa (not fine-tuned). Note that this configuration is biased in favor of RoBERTa, as the parameters of the MLM head and of the RoBERTa encoder have been trained in conjunction and thus good values have been found for this combination, which is not the case for our RULEBERT.

| | Fine-Tuned | RoBERTa | RuleBERT |
|---|---|---|---|
| bAbI | 1 epoch | .401 | .477 |
| | 2 epochs | .676 | **.863** |
| | 3 epochs | .827 | .825 |
| Neg. LAMA | - | .684 | **.852** |
| CheckList QQP | 1 epoch | .000 | **.422** |
| | 3 epochs | .000 | .000 |

Table 6: Evaluation on external datasets (accuracy).

| | | Facts | RoBERTa | | RuleBERT | |
|---|---|---|---|---|---|---|
| | | | $\rho$ | % | $\rho$ | % |
| GR | birthplace | 2,404 | 90.99 | 18.51 | 71.72 | 4.20 |
| | birthdate | 1,565 | 82.87 | 1.40 | 63.55 | 0.13 |
| | deathplace | 649 | 86.44 | 0.31 | 71.13 | 0.00 |
| T-REx | 1-1 | 973 | 78.95 | 61.38 | 51.21 | 32.96 |
| | N-1 | 20,006 | 87.56 | 43.80 | 67.63 | 11.48 |
| | N-M | 13,096 | 89.39 | 50.78 | 72.59 | 28.90 |
| ConceptNet | — | 2,996 | 42.61 | 9.00 | 37.43 | 4.83 |
| SQ | — | 286 | 89.71 | 44.76 | 75.05 | 26.32 |

Table 7: Negated LAMA: Mean Spearman rank correlation ($\rho$) and mean percentage of overlap in the first ranked predictions (%) for original vs. negated queries.

**Results** Yet, even in this arguably unfair setting, RuleBERT outperforms RoBERTa on all datasets of Negated LAMA, as shown in Table 7. We can see that RuleBERT performs better on both evaluation measures used in (Kassner and Schütze, 2020). It achieves a lower mean Spearman rank correlation ($\rho$) and a much smaller percentage of positive and negated answers overlap (%).

### 7.2 CheckList QQP Experiments

The CheckList tests (Ribeiro et al., 2020) have shown that PLMs fail in many basic cases. We hypothesize that RuleBERT can perform better on tasks and examples that deal with symmetric and asymmetric predicates, if such predicates have been shown to it during pre-fine-tuning. We experiment with the QQP dataset, which asks to detect whether two questions are duplicates. We identify a few rules that can teach a model about symmetric predicates, and we pre-fine-tune RuleBERT on them; then, we fine-tune it on the QQP dataset.

**Results** Table 6 shows the results on the challenging CheckList QQP test set: we can see that RuleBERT achieves accuracy of 0.422 after one epoch, while RoBERTa is at 0.0. However, after three epochs RuleBERT is also at 0.0,[3] i.e., it started to unlearn what it had learned at pre-fine-tuning (Kirkpatrick et al., 2017; Kemker et al., 2018; Biesialska et al., 2020). Learning a new task often leads to such catastrophic forgetting (Ke et al., 2021). While there are ways to alleviate this (Ke et al., 2021), this is beyond the scope of this paper.

### 7.3 bAbI Task #15 Experiments

Finally, we experiment with task #15 of the bAbI dataset, where the goal is to assess whether a model can perform deductive reasoning. However, as mentioned in the original bAbI paper (Weston et al., 2016), it is not only desirable to perform well on the task, but also to use the fewest examples.

[3]On the much easier QQP test set, RuleBERT achieved 0.89 accuracy after one epoch, and 0.91 after three epochs.

Thus, we use the smallest dataset consisting of about 2,000 data points. We hypothesize that under the same conditions and hyper-parameters, RuleBERT should be able to generalize faster and to learn in fewer epochs. As PLMs produce varying scores when fine-tuned on small datasets, we repeat the experiment ten times and we report the average scores. We then compare to RoBERTa. Both models contain two classification layers to predict start and end spans of the input context.

**Results** We can see in Table 6 that RuleBERT achieves accuracy of 0.863 in two epochs, while RoBERTa achieves 0.676. On the third epoch, RoBERTa catches up with accuracy of 0.827, while RuleBERT starts to overfit (goes down to 0.825), indicating that fewer epochs should be used, probably due to catastrophic forgetting.

## 8 Conclusion and Future Work

We studied whether PLMs could reason with soft rules over natural language. We experimented with one flavor of probabilistic answer set programming ($\text{LP}^{\text{MLN}}$), but other semantics can be also used with the proposed methodology. We further explored the inference capabilities of Transformer-based PLMs, focusing on positive and negative textual entailment.

We leave non-entailment for future work. We also leave open the development of explainable models. Some approaches use occlusion that removes parts of the input and checks the impact on the output (Clark et al., 2020) or build proof iteratively using 1-hop inference (Tafjord et al., 2021).

## Acknowledgments

This work is partially supported by a Google Faculty Research Award and the ANR JCJC Grant *InfClean*.

## Ethics and Broader Impact

**Data Collection** While we generated the facts in our examples, the logical rules have been mined from the data in the DBpedia knowledge graph, which in turn has been generated from Wikipedia.

**Biases** We are aware of (*i*) the biases and abusive language patterns (Sheng et al., 2019; Zhang et al., 2020; Bender et al., 2021; Liang et al., 2021) that PLMs impose, and (*ii*) the imperfectness and the biases of our rules as data from Wikipedia has been used to mine the rules and compute their confidences (Janowicz et al., 2018; Demartini, 2019). However, our goal is to study PLM's capability of deductive soft reasoning. For (*i*), there has been some work on debiasing PLMs (Liang et al., 2020), while for (*ii*), we used mined rules to have more variety, but could resort to user-specified rules validated by consensus to relieve the bias.

**Environmental Impact** The use of large-scale Transformers requires a lot of computations and GPUs/TPUs for training, which contributes to global warming (Strubell et al., 2019; Schwartz et al., 2020). This is a smaller issue in our case, as we do not train such models from scratch; rather, we fine-tune them on relatively small datasets. Moreover, running on a CPU for inference, once the model is fine-tuned, is less problematic as CPUs have a much lower environmental impact.

## References


Naser Ahmadi, Thi-Thuy-Duyen Truong, Le-Hong-Mai Dao, Stefano Ortona, and Paolo Papotti. 2020. RuleHub: A public corpus of rules for knowledge graphs. *J. Data and Information Quality*, 12(4).

Chitta Baral. 2010. *Knowledge Representation, Reasoning and Declarative Problem Solving*, 1st edition. Cambridge University Press, USA.

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, page 610–623, Virtual Event, Canada. Association for Computing Machinery.

Magdalena Biesialska, Katarzyna Biesialska, and Marta R. Costa-jussà. 2020. Continual lifelong learning in natural language processing: A survey. In *Proceedings of the 28th International Conference on Computational Linguistics*, COLING '20, pages 6523–6541, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Ekaba Bisong. 2019. Google colaboratory. In *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*, pages 59–64. Apress.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019a. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, NAACL-HLT '19, pages 2924–2936, Minneapolis, Minnesota, USA. Association for Computational Linguistics.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019b. What does BERT look at? An analysis of BERT's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, BlackboxNLP '19, pages 276–286, Florence, Italy. Association for Computational Linguistics.

Peter Clark, Oyvind Tafjord, and Kyle Richardson. 2020. Transformers as soft reasoners over language. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence,*, IJCAI '20, pages 3882–3890, Online. International Joint Conferences on Artificial Intelligence Organization.

Ido Dagan, Dan Roth, Mark Sammons, and Fabio Massimo Zanzotto. 2013. *Recognizing Textual Entailment: Models and Applications*. Synthesis Lectures on Human Language Technologies. Morgan and Claypool publishers.

Gianluca Demartini. 2019. Implicit bias in crowdsourced knowledge graphs. In *Proceedings of the 2019 World Wide Web Conference: Companion Volume*, WWW '19, page 624–630, San Francisco, California, USA. Association for Computing Machinery.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, NAACL-HLT '19, pages 4171–4186, Minneapolis, Minnesota, USA. Association for Computational Linguistics.

Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah A. Smith. 2020. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv:2002.06305*.

Yanai Elazar, Nora Kassner, Shauli Ravfogel, Abhilasha Ravichander, Eduard H. Hovy, Hinrich Schütze, and Yoav Goldberg. 2021. Measuring and

improving consistency in pretrained language models. *arXiv:2102.01017*.

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

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. 2014. Clingo = ASP + control: Preliminary report. *arXiv:1405.3694*.

William L. Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. 2018. Embedding logical queries on knowledge graphs. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 2030–2041, Montréal, Canada. Curran Associates Inc.

Krzysztof Janowicz, Bo Yan, Blake Regalia, Rui Zhu, and Gengchen Mai. 2018. Debiasing knowledge graphs: Why female presidents are not like female popes. In *Proceedings of the International Semantic Web Conference*, ISWC '18, Monterey, California, USA.

Nora Kassner, Benno Krojer, and Hinrich Schütze. 2020. Are pretrained language models symbolic reasoners over knowledge? In *Proceedings of the 24th Conference on Computational Natural Language Learning*, CoNLL '20, pages 552–564, Online. Association for Computational Linguistics.

Nora Kassner and Hinrich Schütze. 2020. Negated and misprimed probes for pretrained language models: Birds can talk, but cannot fly. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, ACL '20, pages 7811–7818, Online. Association for Computational Linguistics.

Zixuan Ke, Hu Xu, and Bing Liu. 2021. Adapting BERT for continual learning of a sequence of aspect sentiment classification tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL-HLT '21, pages 4746–4755, Online. Association for Computational Linguistics.

Ronald Kemker, Marc McClure, Angelina Abitino, Tyler L. Hayes, and Christopher Kanan. 2018. Measuring catastrophic forgetting in neural networks. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, AAAI '18, pages 3390–3398, New Orleans, Louisiana, USA. AAAI Press.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526.

Guillaume Lample and François Charton. 2020. Deep learning for symbolic mathematics. In *Proceedings of the 8th International Conference on Learning Representations*, ICLR '20, Addis Ababa, Ethiopia. OpenReview.net.

Joohyung Lee, Samidh Talsania, and Y. Wang. 2017. Computing LPMLN using ASP and MLN solvers. *Theory and Practice of Logic Programming*, 17(5–6):942–960.

Joohyung Lee and Yi Wang. 2016. Weighted rules under the stable model semantics. In *Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning*, KR '16, page 145–154, Cape Town, South Africa. AAAI Press.

Paul Pu Liang, Irene Mengze Li, Emily Zheng, Yao Chong Lim, Ruslan Salakhutdinov, and Louis-Philippe Morency. 2020. Towards debiasing sentence representations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, ACL '20, pages 5502–5515, Online. Association for Computational Linguistics.

Paul Pu Liang, Chiyu Wu, Louis-Philippe Morency, and Ruslan Salakhutdinov. 2021. Towards understanding and mitigating social biases in language models. In *Proceedings of the International Conference on Machine Learning*, ICML '21, pages 6565–6576, Online. PMLR.

Percy Liang. 2016. Learning executable semantic parsers for natural language understanding. *Commun. ACM*, 59(9):68–76.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin. 2020. RoBERTa: A robustly optimized BERT pretraining approach. In *Proceedings of the 8th International Conference on Learning Representations*, ICLR '20, Addis Ababa, Ethiopia. OpenReview.net.

Bill MacCartney and Christopher D. Manning. 2009. An extended model of natural logic. In *Proceedings of the Eight International Conference on Computational Semantics*, IWCS-WS '09, pages 140–156, Tilburg, The Netherlands. Association for Computational Linguistics.

Pasquale Minervini, Matko Bošnjak, Tim Rocktäschel, Sebastian Riedel, and Edward Grefenstette. 2020. Differentiable reasoning on large knowledge bases and natural language. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):5182–5190.

Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. 2021. On the stability of fine-tuning bert: Misconceptions, explanations, and strong baselines. In *Proceedings of the 9th International Conference on Learning Representations*, ICLR '21, Virtual Event, Austria. OpenReview.net.

Stefano Ortona, Venkata Vamsikrishna Meduri, and Paolo Papotti. 2018. Robust discovery of positive and negative rules in knowledge bases. In *Proceedings of the 2018 IEEE 34th International Conference on Data Engineering*, ICDE '18, pages 1168–1179, Paris, France. IEEE.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language models as knowledge bases? 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 '19, pages 2463–2473, Hong Kong, China. Association for Computational Linguistics.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, ACL '20, pages 4902–4912, Online. Association for Computational Linguistics.

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2020. A primer in BERTology: What we know about how BERT works. *Transactions of the Association for Computational Linguistics*, 8:842–866.

Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. 2020. Green AI. *Commun. ACM*, 63(12):54–63.

Emily Sheng, Kai-Wei Chang, Premkumar Natarajan, and Nanyun Peng. 2019. The woman worked as a babysitter: On biases in language generation. 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 '19, pages 3407–3412, Hong Kong, China. Association for Computational Linguistics.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, ACL '19, pages 3645–3650, Florence, Italy. Association for Computational Linguistics.

Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. 2021. ProofWriter: Generating implications, proofs, and abductive statements over natural language. In *Findings of the Association for Computational Linguistics*, ACL-IJCNLP '21, pages 3621–3634, Online. Association for Computational Linguistics.

Alon Talmor, Yanai Elazar, Yoav Goldberg, and Jonathan Berant. 2020a. oLMpics-on what language model pre-training captures. *Transactions of the Association for Computational Linguistics*, 8:743–758.

Alon Talmor, Oyvind Tafjord, Peter Clark, Yoav Goldberg, and Jonathan Berant. 2020b. Leap-of-thought: Teaching pre-trained models to systematically reason over implicit knowledge. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020*, volume 33 of *NeurIPS '20*, pages 20227–20237, Online.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019a. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 7th International Conference on Learning Representations*, ICLR '19, New Orleans, Louisiana, USA. OpenReview.net.

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. 2019b. Learning deep transformer models for machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, ACL '19, pages 1810–1822, Florence, Italy. Association for Computational Linguistics.

Yu-An Wang and Yun-Nung Chen. 2020. What do position embeddings learn? An empirical study of pre-trained language model positional encoding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, EMNLP '20, pages 6840–6849, Online. Association for Computational Linguistics.

Jason Weston, Antoine Bordes, Sumit Chopra, and Tomás Mikolov. 2016. Towards AI-complete question answering: A set of prerequisite toy tasks. In *Proceedings of the 4th International Conference on Learning Representations*, ICLR '16, San Juan, Puerto Rico.

Fan Yang, Zhilin Yang, and William W. Cohen. 2017. Differentiable learning of logical rules for knowledge base reasoning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, California, USA*, NeurIPS '17, pages 2319–2328.

Haoran Zhang, Amy X. Lu, Mohamed Abdalla, Matthew McDermott, and Marzyeh Ghassemi. 2020. Hurtful words: Quantifying biases in clinical contextual word embeddings. In *Proceedings of the ACM Conference on Health, Inference, and Learning*, CHIL '20, page 110–120, Toronto, Ontario, Canada. Association for Computing Machinery.

Xuhui Zhou, Yue Zhang, Leyang Cui, and Dandan Huang. 2020. Evaluating commonsense in pre-trained language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34 of *AAAI '20*, pages 9733–9740, Online. AAAI Press.

## A More on Reasoning with Soft Rules

Let $\sigma$ be a signature as in first-order logic. An LP$^{\text{MLN}}$ program $\Pi$ is a finite set of weighted rules of the form:

$$w : A \leftarrow B \tag{1}$$

where $A$ is a disjunction of atoms of $\sigma$, $B$ is a conjunction of literals (atoms and negated atoms) of $\sigma$, and $w$ is a real number or the symbol $\alpha$.

When $A$ is $\bot$ (the empty disjunction), the rule asserts that $B$ should be false in the stable model. An LP$^{\text{MLN}}$ rule (1) is called *soft* if $w$ is a real number or *hard* if $w$ is $\alpha$. An LP$^{\text{MLN}}$ program is *ground* if its rules contain no variables. An LP$^{\text{MLN}}$ program $\Pi$ that contains variables is identified with a ground LP$^{\text{MLN}}$ program $gr_\sigma[\Pi]$, which is obtained from $\Pi$ by replacing every variable with every ground term of $\sigma$. The weight of a ground rule in $gr_\sigma[\Pi]$ is the same as the weight of the corresponding rule in $\Pi$. By $\overline{\Pi}$ we denote the unweighted logic program obtained from $\Pi$, i.e., $\overline{\Pi} = \{R \mid w : R \in \Pi\}$.

For a ground LP$^{\text{MLN}}$ program $\Pi$, $\Pi_I$ denotes the set of rules $w : R$ in $\Pi$ such that $I$ satisfies $R$ (denoted $I \models R$) and SM$[\Pi]$ denotes the set $\{I \mid I$ is a (deterministic) stable model of $\overline{\Pi_I}\}$. The (unnormalized) weight of $I$ under $\Pi$ is defined as follows:

$$W_\Pi(I) = \begin{cases} exp(\sum\limits_{w:R\in\Pi_I} w) & \text{if } I \in \text{SM}[\Pi]; \\ 0 & \text{otherwise.} \end{cases}$$

The probability of $I$ under $\Pi$ is the normalized weight defined as follows:

$$P_\Pi(I) = \lim_{\alpha\to\infty} \frac{W_\Pi(I)}{\sum_{J\in\text{SM}[\Pi]} W_\Pi(J)}.$$

In Answer Set programming (ASP), search problems are reduced to computing *stable models* (a.k.a. answer sets), a set of beliefs described by the program. In the case of a Horn program, the stable models coincide with the minimal models. LP$^{\text{MLN}}$ programs are transformed to meet the needs of an ASP solver (Gebser et al., 2014; Lee et al., 2017).

## B Rule Support

We designed an experiment to show the impact of increasing the number of overlapping rules on the same target predicate. The goal is to measure how often multiple rules are triggered for the same target triple.

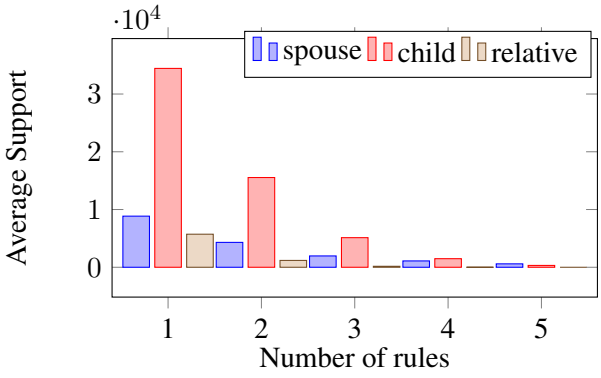


Figure 4: Support of the overlapping rules.

We measure this with the support of a rule, i.e., the number of triples in the knowledge base that satisfy all the atoms in the rule.

To compute the support for more than one rule, we combine the premises of the rules. In this experiment, we picked three predicates (*spouse*, *child* and *relative*), and for each one we selected ten rules randomly. Next, we used DBpedia online endpoint[4] to compute the support for each combination of $n$ ($n$=1,2,...,5) rules for each predicate. The results in Figure 4 show that by increasing the number of rules, the support decreases for all predicates. For combinations with more than three rules, the support is very small.

## C More Experimental Details

For fine-tuning our models, we use Google Colaboratory (Bisong, 2019), which assigns random GPU clusters of various types. The number of parameters of our models is about 355M. We select the values of our hyper-parameters (shown in Table 8) on the development sets, by maximizing accuracy.

The execution times vary largely depending on the GPU at hand and on the scenario, with fine-tuning on a Tesla V100 taking from one hour for a single rule to a few hours for all the chaining experiments. The training/validation/testing splits are shown in Table 1. Table 9 shows the sizes of the used test datasets.

## D Ablation

### D.1 Impact of the Data Size

**Setting.** We report the impact of the size of the fine-tuning data on the model performance. As shown in Table 2, the accuracy of the fine-tuned model is higher for rules with higher confidence.

[4] http://dbpedia.org/sparql

| Hyper-Parameter | Value |
|---|---|
| Learning Rate | 1e-6 |
| Weight Decay | 0.1 |
| Number of Epochs | 3 |
| Batch Size | 16 |
| Learning Rate Decay | Linear |
| Warmup Ratio | 0.06 |

Table 8: Hyper-parameters for fine-tuning our model.

| Dataset | Size |
|---|---|
| Mod0 Test(own) | 2,667 |
| Mod1 Test(own) | 4,000 |
| Mod2 Test(own) | 5,334 |
| Mod3 Test(own) | 6,667 |
| Mod4 Test(own) | 8,000 |
| Mod5 Test(own) | 9,334 |
| Test(D≤5) | 9,334 |
| Depth=0 | 16,057 |
| Depth=1 | 6,608 |
| Depth=2 | 5,389 |
| Depth=3 | 3,993 |
| Depth=4 | 2,619 |
| Depth=5 | 1,336 |

Table 9: Number of examples in each of the test datasets for the chaining experiment.

We therefore divide the rules in three categories: *High* contains rules with confidence greater than 0.8, *Medium* has rules with confidence between 0.4 and 0.8, and *Low* is for the rest. There are six rules in the *Medium* category and the other two categories have five rules each. For each rule, we fine-tune seven models with 1k, 2k, 5k, 10k, 15k, 20k, and 30k examples.

**Results.** Figure 5 shows that having more training data improves the accuracy in all scenarios. For all categories, there is a sizable increase going from 10k to 15k examples; the impact is smaller for higher values. The highest increase is for rules with high confidence, and rules with medium confidence demonstrate larger increase than low confidence.

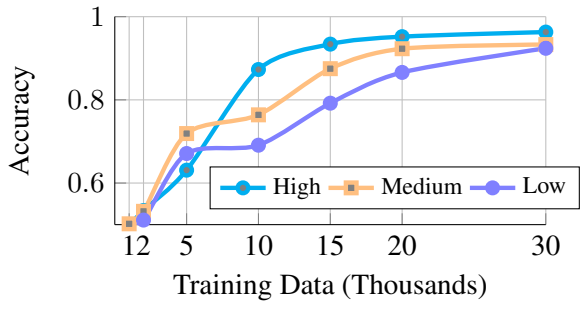


Figure 5: Impact of the training data size.

### D.2 Role of the Example Format

**Setting.** When we teach rules to PLMs, we rely on examples with real names from a fixed pool. However, our goal is to teach PLMs the semantics of the soft rule, not the facts in our examples. Thus, we further design an experiment to assess the impact of the format used in the example facts on the behavior of the model. We distinguish two formats for the generated facts: (*i*) real names such as *Alice* and *IBM*, and (*ii*) letters such as *A* and *B*. We first use each format in fine-tuning and we then test both formats. We end up with two test/train scenarios: one with the same format and one with different formats. For this study, we use just one rule: $child(a,c) \wedge parent(c,b) \rightarrow spouse(a,b)$, with 30K examples for fine-tuning, and 2k for testing.

| | Train Letter | Train Name |
|---|---|---|
| Test Letter | .981 | .932 |
| Test Name | .977 | .985 |

Table 10: Impact of the example format on accuracy.

**Results.** The results in Table 10 show that the model performance does not depend heavily on using the same fact format for training and testing. With examples using letters in training, the results are slightly better in the case with two formats. We ultimately use names for testing and training in our default configuration as it yields better results.

## E Impact of the Random Seed

Pre-trained transformers often suffer from instability of the results across multiple reruns with different random seeds. This usually happens with small training datasets (Dodge et al., 2020; Mosbach et al., 2021). In such cases, typically multiple reruns are performed, and the average value over these reruns is reported.

However, the numbers for the main experiments we report in this paper are not averaged over multiple reruns as our datasets are considerably large and the models did not suffer from instability due to random seeds. For example, when we reran RULE-BERT on a single-rule experiment three times, we obtained accuracy of 0.98959, 0.99551, 0.99636 with a standard deviation of only 0.003.

Yet, for the small dataset *bAbI*, we observed a much higher standard deviation of 0.17. Thus, in this case we report results that are averaged over ten reruns.

## F Data Generation Example

We show an example of data generation for Algorithm 1. For simplicity, here we show an example of a hard rule, i.e., one whose confidence is implicitly set to one.[5] We begin by setting the values of the input parameters:

**Algorithm 1 Input:**

- $r$ = `child(A,C) ∧ parent(C,B) → spouse(A,B)`
- $n = 8$
- $m = 5$
- $pools$ = `{Alice,Bob,Carl,David,Eve}`

We set $n = 8$ to generate all the eight hypotheses. We start by generating a set of facts F (line 3), having predicates from the body of the rule with random polarity. We ensure that there are facts that trigger the rule. Their number should not exceed $m$. Here is an example of generated facts $F$:

**Generated Facts F:**

- $f_1$: `negparent(Eve,Carl)`
- $f_2$: `child(Eve,David)`
- $f_3$: `parent(Carl,Bob)`
- $f_4$: `child(Alice,Carl)`

Four facts are generated in total. Facts $f_3$ and $f_4$ trigger rule $r$. We then feed the rule $r$ and facts $F$ into the LP$^{\text{MLN}}$ reasoner (line 4). The output $O$ is then:

LP$^{\text{MLN}}$ **Reasoner Output O:**

- $o_1$: `child(Eve,David)`
- $o_2$: `child(Alice,Carl)`
- $o_3$: `parent(Carl,Bob)`
- $o_4$: `spouse(Alice,Bob)`
- $o_5$: `negchild(Eve,Carl)`

We start generating the hypotheses:

**Generated Hypotheses H:**

- $h_1$: `child(Eve,David)`
- $h_2$: `child(David,Eve)`
- $h_3$: `spouse(Alice,Bob)`
- $h_4$: `negspouse(Alice,Bob)`
- $h_5$: `child(David,Carl)`
- $h_6$: `negchild(David,Carl)`
- $h_7$: `spouse(Bob,Eve)`
- $h_8$: `negspouse(Bob,Eve)`

[5] We show an example of a soft rule in Section G below.

Hypothesis $h_1$ is obtained by sampling from $F$ (line 5), and thus it is a valid hypothesis. Then, the hypothesis $h_2$ is generated by altering $h_1$ with the function $Alter$ (line 19-22). In this example, since $child$ is not symmetric, $h_2$ is produced using a switch of the subject and the object of $h_1$ to generate a false hypothesis (line 6).

Hypothesis $h_3$ is the outcome of rule $r$ being triggered by facts $f_3$ and $f_4$ (line 7). In a similar fashion to $h_2$, we produce $h_4$ (line 8).

Hypothesis $h_5$ is sampled from the universe of all unsatisfied positive facts having a different predicate than that of the rule body (line 9), which makes it an invalid hypothesis, as it is not found in the $O$. Hypothesis $h_6$ is the negation of $h_5$, and, following CWA, it is a valid hypothesis (line 10).

Finally, hypothesis $h_7$ is sampled from the universe of unsatisfied rule-head atoms (line 11), and it is negated to produce hypothesis $h_8$.

Overall, we obtain eight different examples represented in symbolic knowledge, where each example contains the set of generated facts $F$, the rule $r$, and a single hypothesis $h_i$. The following is one example in symbolic knowledge:

**Example #1 (Symbolic):**

- $Rule\ r$ = `child(A,C) ∧ parent(C,B) → spouse(A,B)`
- $Facts\ F$ :
  - $f_1$: `negparent(Eve,Carl)`
  - $f_2$: `child(Eve,David)`
  - $f_3$: `parent(Carl,Bob)`
  - $f_4$: `child(Alice,Carl)`
- $Hypothesis\ h_3$ : `spouse(Alice,Bob)`

We then convert each example to synthetic English using a set of pre-defined templates for the facts and for the rules. Here is the above Example #1, but now rewritten in synthetic English:

**Example #1 (Synthetic English):**

- $Rule\ r =$ If the child of the first person is the third person, and the parent of the third person is the second person, then the first person is the spouse of the second person.
- $Facts\ F$ :
  - $f_1$**:** The parent of Eve is not Carl.
  - $f_2$**:** The child of Eve is David.
  - $f_3$**:** The parent of Carl is Bob.
  - $f_4$**:** The child of Alice is Carl.
- $Hypothesis\ h_3$ : The spouse of Alice is Bob.

The $Context$ is defined as the combined set of facts and rule(s). Both $Context$ and $Hypothesis$ are fed as an input to the model.

**Example #1 (Model Input):**

- $Context$ : The parent of Eve is not Carl. The child of Eve is David. If the child of the first person is the third person, and the parent of the third person is the second person, then the first person is the spouse of the second person. The parent of Carl is Bob. The child of Alice is Carl.
- $Hypothesis$ : The spouse of Alice is Bob.

## G Rule Overlap Example

After generating the data for every rule in Figure 2, we generate additional examples using combinations of rules. Below, we show how to handle the interaction of two rules: $r_2$ and $r_3$. We follow the procedure in Algorithm 1 by generating facts that trigger the rules, but we only take into consideration hypotheses that deal with rule conclusions.

For example, consider the following facts:

**Generated Facts**

- $f_1$**:** `negparent(Eve,Carl)`
- $f_2$**:** `child(Eve,David)`
- $f_3$**:** `relative(Eve,David)`
- $f_4$**:** `predecessor(Eve,David)`

We can generate an example that triggers two rules: $f_2$ triggers $r_2$, and $f_3$ triggers $r_3$. Feeding the above facts and rules $r_2$ and $r_3$ to the reasoner, we obtain the following output (the numbers in parentheses indicate the likelihood of the triple):

**$\mathrm{LP}^{\mathrm{MLN}}$ Reasoner Output O:**

- $o_1$**:** `relative(Eve,David) (1.0)`
- $o_2$**:** `child(Eve,David) (1.0)`
- $o_3$**:** `negparent(Eve,Carl) (1.0)`
- $o_4$**:** `spouse(Eve,David) (0.134)`
- $o_5$**:** `negspouse(Eve,David) (0.55)`
- $o_6$**:** `predecessor(Eve,David) (1.0)`

We produce hypotheses that trigger both rules together. For example, here we generate two hypotheses coming from $o_4$ and $o_5$. The confidence (weight) of a hypothesis is given by the $\mathrm{LP}^{\mathrm{MLN}}$ reasoner. Taking $o_5$ as a hypothesis, we feed the following example to the model:

**Example #2 (Model Input):**

- $Context$ : The parent of Eve is not Carl. The child of Eve is David. If the child of the first person is the second person, then the first person is not the spouse of the second person. The relative of Eve is David. If the relative of the first person is the second person, then the first person is the spouse of the second person. The predecessor of Eve is David.
- $Hypothesis$ : The spouse of Eve is not David.
- $Weight$ : 0.55

We also generate an example, where three rules are triggered: In addition to $r_2$ and $r_3$, $r_5$ is triggered by $f_4$. We then repeat the same procedure to generate the following example:

**Example #3 (Model Input):**

- $Context$ : The parent of Eve is not Carl. The child of Eve is David. If the child of the first person is the second person, then the first person is not the spouse of the second person. The relative of Eve is David. If the relative of the first person is the second person, then the first person is the spouse of the second person. The predecessor of Eve is David. If the predecessor of the first person is the second person, then the first person is not the spouse of the second person.
- $Hypothesis$ : The spouse of Eve is not David.
- $Weight$ : 0.6

This procedure is repeated for all combinations of two or more rules. In case when all rules have the same head polarity, we generate a false example by altering the hypothesis and finding the complement of the initial (1-$Weight$) weight. For example, $r_2$ and $r_5$ can occur together and both have the same rule head, and thus no conflict occurs. The generated valid example would be as follows:

**Example #4 (Model Input):**

- $Context$ : The parent of Eve is not Carl. The child of Eve is David. If the child of the first person is the second person, then the first person is not the spouse of the second person. The relative of Eve is David. The predecessor of Eve is David. If the predecessor of the first person is the second person, then the first person is not the spouse of the second person.
- $Hypothesis$ : The spouse of Eve is not David.
- $Weight$ : 0.64

An invalid example is generated from the valid example by altering the hypothesis. Here is an invalid example:

**Example #4 (Model Input):**

- $Context$ : The parent of Eve is not Carl. The child of Eve is David. If the child of the first person is the second person, then the first person is not the spouse of the second person. The relative of Eve is David. The predecessor of Eve is David. If the predecessor of the first person is the second person, then the first person is not the spouse of the second person.
- $Hypothesis$ : The spouse of Eve is ~~not~~ David.
- $Weight$ : 0.36 (1-0.64)

## H Rule Chaining Example

Here is an example that illustrates rule chaining:

**Example #5 (Symbolic):**

- $Rules\ R =$
  - $r_1$: `child(A,C) ∧ parent(C,B) → spouse(A,B)`
  - $r_2$: `child(B,A) → parent(A,B)`
- $Facts\ F$ :
  - $f_1$: `negparent(Eve,Carl)`
  - $f_2$: `child(Bob,Carl)`
  - $f_3$: `child(Alice,Carl)`
- $Hypothesis\ h$ : `spouse(Alice,Bob)`

$f_2$ triggers $r_2$ which produces $t =$ `child(Bob,Carl).`

$t$ and $f_3$ trigger $r_1$ to validate the hypothesis $h$. $r_1$ and $r_2$ have been chained to validate the hypothesis. Since we used two rules to validate the hypothesis, we say that this is a chain of depth $= 2$.