Query Optimization of Backward-Chaining Reasoning with Learned Heuristics

Ben Schack[1], Yifan Zhang[1], James Hoffmeister[1] and Jeff Heflin[1]

[1]Lehigh University, CSE, 113 Research Drive, Bethlehem, PA 18015, USA

Abstract

Standard backward-chaining reasoners can perform well when operating on optimized knowledge bases but are often inefficient when used on unoptimized knowledge bases. We extend prior research that makes reasoners more efficient by learning a scoring function for pairs of goals and rules. This scoring function is then used to guide the search for a solution. We propose two improvements to the process: rather than ordering the search by the descending scores of goal/rule pairs, we find the maximum score for each goal, and then choose the goal with the lowest maximum. This prunes the search and encourages proving the most selective goal first. To support this search strategy, we modify the training regimen to include negative examples for goals that are normally proven by facts. We demonstrate the improved performance of these systems on several different knowledge bases including synthetically generated knowledge bases of varying sizes and a subset of the Lehigh University Benchmark (LUBM) translated into Datalog.

Keywords

Neuro-symbolic, Horn logic, Backward-chaining, Knowledge base, Deep learning

1. Introduction

The purpose of this research is to enhance reasoning on knowledge bases (KBs) that were not carefully designed by knowledge engineers. There are many cases where we would want to reason with less optimal KBs, such as when we are translating from a different knowledge representation language. Even if the original representation had an optimal structure, the translation can introduce flaws, especially since it is most likely not a one-to-one translation. Another case is when we are combining multiple KBs on the fly and there is not time to optimize the data.

Our current work focuses on Horn logic and backward-chaining reasoning, with more expressive logics and associated reasoning algorithms as the target of future research. We take a neuro-symbolic approach, where machine learning is used to evaluate the choices available to a symbolic reasoner. Our contribution consists mainly of three parts. 1) We have created a novel approach to selecting subgoals and matching rules during reasoning. Rather than select the highest scoring goal / rule pair as in the manner of Jia et al. [1], we first choose the subgoal expected to be the most selective and then the highest scoring rule for that subgoal. 2) We introduce an approach to create additional negative training examples to avoid bias in meta-reasoning, particularly when goals can only be proven through facts. 3) We have extensively tested the performance of this new method on both synthetic KBs and larger realistic KBs.

2. Background

Neuro-symbolic reasoning represents an advanced computational approach that seeks to integrate the strengths of both neural networks and symbolic artificial intelligence to facilitate more powerful and interpretable models [2]. This hybrid methodology aims to combine the robust learning capabilities of neural networks, which excel in handling unstructured data like images and text, with the precise

RuleML+RR'24: Companion Proceedings of the 8th International Joint Conference on Rules and Reasoning, September 16–22, 2024, Bucharest, Romania

bhschack@pm.me (B. Schack); yiz521@lehigh.edu (Y. Zhang); jah823@lehigh.edu (J. Hoffmeister); jeh3@lehigh.edu (J. Heflin)

and explainable reasoning processes inherent in symbolic AI, which operates with defined rules and structured data. This can include a broad range of topics from generating embeddings of knowledge graphs to training a neural network to predict whether one logical statement entails another.

One of the earliest attempts to combine logic rules and neural networks was KBANN [3]. KBANN takes propositional Horn rules and directly encodes them into the neural network. In the opposite direction, Xu et al. [4] have designed a semantic loss function for propositional logic, which can specify additional constraints on a neural network.

There are several works that integrate neural networks and first-order logic. Kijsirikul and Lerdlamnaochai [5] train a neural network that can perform inductive learning on first-order logic data. However, their architecture only allows the input of data representing a conjunction of atoms, and the output is a set of classes. There is no way to incorporate axioms into their reasoning. Rocktäschel and Riedel [6] trained a neural network to perform unification and apply a backward-chaining-like process. This network was used to predict missing atoms in a KB. TensorLog [7] uses a deep-learning architecture to implement a probabilistic deductive database. Like KBANN, it takes a KB as input and produces a neural net that computes inference. Unlike KBANN, TensorLog supports first-order logic, although it is restricted to unary and binary predicates.

Furthermore, innovations from domains like AlphaGo and NLP, which utilize reinforcement policy-based learning and attention mechanisms, are being integrated to refine problem-solving capabilities in AI [8]. Research has also extensively explored mathematical proofs, employing automated theorem provers to anticipate which established statements are required to substantiate a specific theorem. This approach enhances the efficiency of mathematical reasoning by leveraging proven statements to facilitate the proof process for new theorems[9]. Projects like NELLIE further illustrate this concept by emulating expert systems using neural natural language processing models[10]. Moreover, large language models (LLMs) facilitate the navigation of KBs composed in natural language, allowing for applying both forward-chaining and backward-chaining reasoning processes directly within human-readable texts.

Such LLM-based reasoners include AlphaGeometry, a model which generates proofs to solve Olympiad geometric problems [11]. AlphaGeometry uses forward inference to find conclusions from a starting premise. Notably, the AlphaGeometry model uses a forward search whereas our model utilizes backward reasoning to connect a query to a known KB. Additionally, the AlphaGeometry model deduces new statements exhaustively while our model implements inference that attempts to shorten the search of the proof space. ReProver, another LLM-based theorem prover for LeanDojo selects premises from large math libraries [12]. ReProver scores the retrieved premises using an LLM, much like our model scores possible rule-goal pairs. Our research aligns closely with these advancements as we develop methods to anticipate necessary rules and identify optimal paths through search trees for given queries.

Several researchers in theorem proving have looked at the problem of using machine learning for proof guidance. Wang et al. [13] proposed to use Graph Convolutional Networks to identify which mathematical statements were relevant to a given conjecture. Jakubův and Urban's ENIGMA is a learning-based method to train a classification model to identify "*useful* and *un-useful*" clauses for proof search [14]. Crouse et al. combine various embedding strategies, including from ENIGMA, with a deep reinforcement learning approach to proof search [8].

This paper extends the work of Jia et al. [1] which showed that learning an embedding of logical statements that respects unification is superior to other representations [14, 8]. Jia et al. also divided the problem of ML-based meta-reasoning into three subproblems: representation, training, and control. For representation, the general approach is to learn an embedding for logical atoms using triplet loss, where given an anchor a, the positive example p unifies with a and the negative example n does not unify with it. For training, it uses a supervised learning approach and a simple two-layer feed-forward neural network to learn a score for goal/rule pairs (g, r). The positive and negative (g, r) examples are generated by first deducing a set of candidate training queries using forward-chaining, and then using standard backward-chaining with randomized ordering of subgoals and rule bodies to build a search tree. For each node in the tree when a rule r eventually solves the chosen goal g, a positive (g, r) example is created; all other nodes lead to a negative (g, r) example. Finally, the control policy uses Arnold and Heflin's approach [15] of always choosing the highest scoring (g, r) pair. Although this

Table 1
A Sample Knowledge Base

```
mother(X,Y) :- female(X), parent(X,Y)
female(mary)
female(jane)
female(sophie)
female(rose)
female(sara)
parent(rose, jake)
```

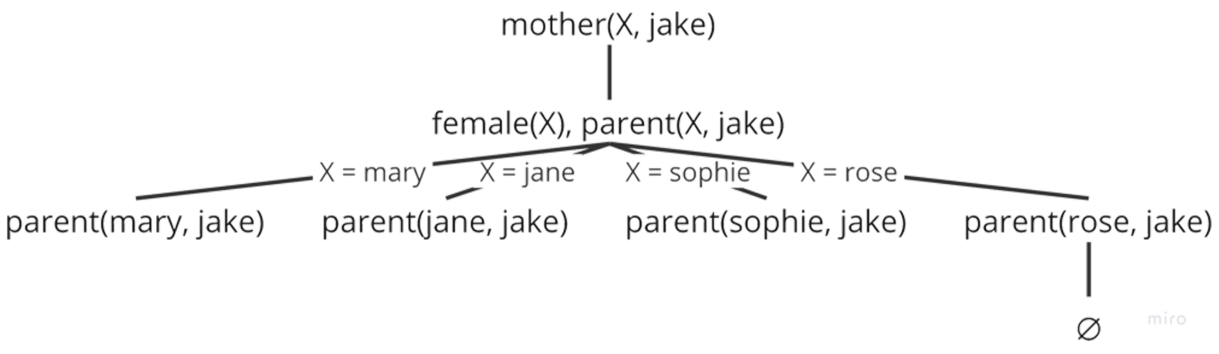

Figure 1: Non-Optimal Query Search Tree.

approach was generally superior to standard backward-chaining, it would sometimes have extreme outliers that were much worse. Jia et al. also proposed an alternate control strategy that would prune any option with a score below a given threshold. This removed the outliers, but would sometimes miss valid answers.

3. Choosing Selective Goals

A common approach to Horn logic reasoning is to use backward-chaining which performs a depth-first search by choosing a goal and then attempting to prove it by applying a rule. Such reasoners typically select the first (left-most) subgoal at each step, and apply rules from top to bottom. Knowledge engineers can optimize the design of KBs to take advantage of this process. However, this is not always ideal: it may not be possible to optimize for all usage patterns simply with statement ordering and there are several use cases where a knowledge engineer will not be available. In particular, we might want to reason with a KB that was produced via translation from some other knowledge representation language, or we might want to reason over a collection of ontologies and data sources collected from the Web.

Consider the (unoptimized) simple KB in Table 1. When running the query `mother(X, jake)` with a standard reasoner it will conduct an inefficient search. As demonstrated by the search tree in Fig. 1, it will first expand the query body to `female(X), parent(X, jake)`; then for every $female$ fact, it will check if there is a $parent$ fact that satisfies the clause. The more $female$ facts there are, the more back-tracking that will be necessary.

However, if the algorithm chose to prove `parent(X,jake)` first, the search will involve no back-tracking, as shown in Fig. 2. The `parent(rose, jake)` fact can be used to prove the subogal by substituting $\{X/rose\}$. The new set of subgoals contains only `female(rose)`. The optimal ordering has the additional benefit of stopping without exploring every reachable datum for queries that are impossible to prove. The answer search for the query $mother(X, emily)$, for instance, will end significantly earlier than in the sub-optimal case. That is because the reasoner would see that there is no $parent$ for $emily$ and end the search. In the sub-optimal case, the reasoner will explore every $female$, substitute it into $parent$, and check if that $parent$ exists.

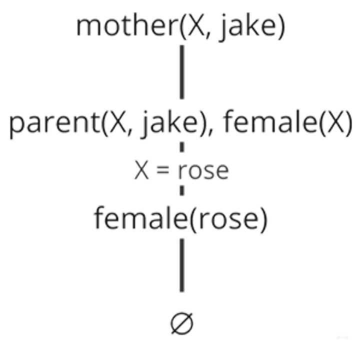

Figure 2: Optimal Query Search Tree.

Algorithm 1 BackwardChainGuided($goals$)

```
if $goals = \emptyset$ then
    return success
else if $depth > MaxDepth$ then
    return fail
else
    for all $g \in goals$ do
        $match_g \leftarrow \{(g, r) \mid \text{Unify}(g, \text{head}(r))\}$
        sort $match_g$ in descending order of score$(g, r)$
        $score_g \leftarrow \max_{(g,r) \in match_g} \text{score}(g, r)$
    end for
    $select \leftarrow match_g$ such that $\min_g score_g$
    for all $(g, r) \in select$ do
        $newbody \leftarrow \text{Subst}(body(r) \cup goals - \{g\})$
        $ans \leftarrow \text{BackwardChainGuided}(newbody)$
        if $ans \neq fail$ then
            return $ans$
        end if
    end for
end if
return $fail$
```

The goal of this paper is to use machine learning to predict which goal will lead to the most efficient search. In the simple example above, the difference is minimal, but even in small knowledge bases with ten rules and a few hundred facts, it is possible for an inefficient search to consider millions of nodes.

Any goal can have many possible rule substitutions and often multiple goals need to be proven. A standard reasoner will pick the first goal and the first matching rule for that goal before continuing the process. Jia et al. [1] showed that the control strategy of Arnold and Heflin [15] could have very redundant subtrees during search. The problem is that since it considers all goal/rule pairs (g, r) in descending order of the learned score, it considers every subgoal at every node. Even if it fails to prove one of the goals, it will continue searching with each of the other goals, and these subtrees will include attempts to prove the previously failed subgoal (perhaps with a substitution applied). If we have a query q with goals a, b, and c, we will choose the rule with the highest score among the substitutions for those goals which results in goals a, b, and d, having substituted d for c. If the proof fails, it will then iterate to the next highest substitution score in the original goal list. The result is that we repeat trying to solve goals that have already been disproven.

Our solution is to utilize the scores of all candidate (g, r) pairs to choose the goal that is most likely to lead to an efficient search. Here, we assume that we have learned a scoring function $score(g, r)$, where higher scores are indicative of greater success to proving the goal g via rule r. Given goals a_0, b_0, and c_0, we have a set of rules $r \in R$ where the head of r unifies with one of the goals. We can then further divide R into subsets $A, B, C \subset R$ such that the head of any rule $a \in A$ unifies with a_0, the the head of any rule $b \in B$ unifies with b_0, and the head of any rule $c \in C$ unifies with c_0. We calculate the maximum score for each subset and choose the subset with the lowest maximum score to iterate through. This effectively chooses the goal that we expect to be hardest to prove first. If we fail to prove this goal, no other goals will be considered. The process is shown in Algorithm 1, where each goal g has a set of matching rules $match_g$ (line 7). Line 9 assigns a value $score_g$ to g using the maximum score from $match_g$, and line 11 selects the goal with the lowest such score.

There are several motivations behind this implementation: a (g,r) match with a lower score is more likely to fail faster than a match with a higher score, and a disproven goal cannot be proven regardless of its depth in the search tree. A rule with many facts associated with it will have a high score since it often results in answers, while a rule with few facts associated with it will have a low score since it rarely results in answers. It is beneficial to disprove a goal quickly since we can explore different branches instead. For instance, goals a_0, b_0 with matching rules $a_0 \leftarrow a_1, b_0 \leftarrow b_1$. We might select b_0 first because the rule has a lower score. If we are able to disprove b_0, then we can avoid attempting to prove a_0.

This algorithm removes the fallback depth that was used in previous work [15, 1]. The fallback depth was a point in the search where the algorithm would switch back to a classical backward-chaining reasoner, and was an attempt to limit the amount of redundant exploration conducted. The algorithm presented in this paper never needs to resort to the standard reasoner.

4. Training with Negative Facts

Our second contribution is an insight into the proper generation of training data. In particular, in order to get good performance from the guided reasoner, we must make sure there is a representative set of negative training examples for facts.

Prior work [15, 1] generated training examples based on the tree of a randomized backward-chaining search. If the search attempted to prove goal g using clause c and succeeded, then a positive example (g,c,1) was created, even if the search eventually failed to prove an ancestor goal. However, if it failed, then a negative example (g,c,0) was created. However, this approach has a problem: when g can only be proven through facts, it will not generate any negative examples. The search only considers clauses that unify with g, and if a fact unifies with g, then g is automatically proven. Thus, g will only have positive examples. This will bias the learned score of all (g,r) pairs such that the more times g is proven, the higher the score will be, regardless of how hard it actually is to prove g.

We remedy this problem by adding useful negative training examples for such goals. Consider the search tree from Fig. 1. The first branch fails because we are unable to find any clauses that unify with parent(mary,jake). In this case, we can use the goal prior to the substitution $\{X/mary\}$ to create a negative example with goal $parent(X, jake)$ and rule $parent(mary, jake)$. We add the following logic to our generation of training examples:

```
if goal $g$ fails because there are no clauses that match with it
    if the parent search node $p$ proved its goal using a fact[1]
        let $g'$ be the second goal in $p$
        create a negative example consisting of goal $g'$ and rule $g$
```

For example in one genealogy knowledge base with 30 rules and 304 facts, when trying to prove $ancestor(lewis, X)$ the rule `ancestor(X,Y) :- parent(X,Y)` had a score of 0.017 while the rule `ancestor(X,Y) :- parent(X,Z),ancestor(Z,Y)` had a score of 0.12 when negative facts were

[1]This condition ensures that the original version of g is easy to find

not used. When negative facts were used in training, these scores became 0.785 and 0.503 respectively. This leads to choosing the simpler rule of checking for $lewis'$ parent first before trying transitivity of ancestry.

5. Experiments

5.1. Experimental Process

Firstly, we use triplet loss with unifying and non-unifying atoms to train a unification embedding model. All of our experiments use a target embedding size of 50.

This is followed by training our guided reasoning model, which scores potential (g, r) pairs. To train our guided reasoning model, we need to create the query training and test datasets. The creation of the query dataset begins with a list of derivable facts, which includes all possible facts given a knowledge base. For the synthetic knowledge base experiments, this fact list is generated by applying forward-chaining to the knowledge base. This fact list serves as the basis for creating our training and testing query datasets. For the LUBM knowledge base experiments, we use the same process to generate the training dataset but use the reference answers of LUBM-designed test queries to generate the test dataset. We generate queries from facts by randomly replacing one or more constants with a variable.

This method allows us to explore different facets of query formation, enhancing the richness of the dataset used for training and testing our models. The generated queries are shuffled to ensure variability and reduce order bias. The first 100 shuffled queries are designated for training, while the subsequent 100 are used for testing, facilitating a robust evaluation of the model's performance across various query scenarios. The training dataset is then used to create negative and positive example data using the process described in Section 4.

Unlike the approach in previous work by Jia et al., where the training phase was rigidly set to conclude after 1,000 epochs, we introduced a more adaptive stopping mechanism. This adjustment allows us to enhance the model's accuracy and potentially reduce training time by terminating the process when sufficient learning has been achieved, thus saving computational resources.

Given that our training loss exhibited significant variability, we implemented a double-smoothing technique. This can help us better understand the model's learning progress and identify when it has started to over-fit or under-fit. We first apply the Savitzky-Golay filter to the training loss curve. The Savitzky-Golay filter is a digital filter used to smooth data points while maintaining the signal's shape and features, particularly when the signal has sharp peaks or other fine structures. After using the Savitzky-Golay filter, the loss curve is still somewhat noisy, so we also use an averaging filter with a smoothing window of 5 loss points, which helped reduce noise and stabilize the loss curve for more reliable analysis.

The last step in our process is testing the different models on the testing dataset we created previously. We compare four systems. In the case of the neuro-symbolic models, it is assumed that $score(g, r)$ is produced by a two-layer neural network.

- **Standard** - A classicalbackward-chaining reasoner that chooses the first goal and matches rules in KB order.
- **All Goals** - Selects the (g, r) pair with the highest score without any other conditions. This is equivalent to the Unification system from Jia et al.
- **Min Goal** - Applies the goal selection approach from Section 3, where first we calculate the maximum score for each goal g, and then only consider (g, r) pairs that correspond to the g with lowest maximum score. This uses the same training regimen as Jia et al.
- **Min Goal (NF)** - Use the same control strategy as *Min Goal*, but applies the training regimen described in Section 4.

Table 2
Performance comparison of reasoners on synthetic knowledge bases

KB Size	Reasoner	Median	Mean	Fails	Time
250	Standard	1,998.7	17,204.2	0.6	27.4
250	All Goals	3.4	12,808,046.4	12.6	20,555.3
250	Min Goal	3.4	360.9	0.0	3.1
250	Min Goal (NF)	3.4	981.8	0.0	7.2
375	Standard	2,429.2	33,459.5	2.8	77.8
375	All Goals	11.0	21,800,049.6	21.8	33,010.4
375	Min Goal	110,422.6	221,274.0	2.0	553.9
375	Min Goal (NF)	11.0	129,393.9	2.0	350.6
500	Standard	552,639.1	3,419,493.6	7.4	5,606.8
500	All Goals	2.8	26,517,593.2	26.4	35,554.2
500	Min Goal	3.2	7,990,467.8	6.6	10,356.3
500	Min Goal (NF)	2.8	8,481,922.1	7.0	10,863.0

5.2. Synthetic Experiments

5.2.1. Design

Our synthetic experiments consist of small knowledge bases of varying sizes and complexity. The varying complexity comes in several forms: different numbers of constants change the number of ground facts possible; the number of stated facts is different; and a larger number of rules leads to longer chains of rules. The symbolic data representation is in the form of predicates $p0, p1$, ..., constants $a0, a1$, ..., and variables $X0, X1$, ... We first generate a vocabulary with the desired number of predicates, constants, and variables. We then generate the knowledge base with the symbols from the vocabulary. Our current experiments restrict the predicates' arity to two, as this is the most common type in real-world knowledge bases. We conducted experiments on synthetic knowledge bases of sizes 250, 375, and 500, with approximately 20 percent of the entries consisting of rules and 80 percent of facts. All the knowledge base sizes have 20 predicates and 10 variables with the only difference being the number of constants which are 200, 300, and 400 respectively.

5.2.2. Results

We have run experiments on five distinct synthetic knowledge bases for each size. By averaging the data from each execution, we can see a clear performance difference between the reasoners. This is important because factors such as the degree to which the rules exhibit cycles can impact the complexity of the search. We abort any queries that fail to find an answer after exploring 100 million nodes. For each individual experiment, we collect several different types of data: the median number of nodes explored, the average number of nodes explored, the number of failed queries, and the total time to execute each test query in seconds. 100 test queries are used for each knowledge base.

First, the most important metric we are measuring is the median number of nodes visited. Since it is common for there to be a few queries that fail (i.e., they timed out after exploring 100 million nodes) which skew these data greatly. One of our main goals for this research was to lower the mean number of nodes explored, meaning that we needed to improve our worst-case scenario. As seen in Table 2, the reasoners using scoring usually have a similar median number of nodes explored. This is because most of the queries perform well, except for some outliers that skew the mean.

The *Min Goal* strategy (with or without negative facts) is clearly an improvement over *All Goals*. It explores anywhere from 3x fewer nodes to 10,000x fewer. This is typically done by failing on fewer queries (e.g., for size 500, *All Goals* fails an average of 26.4 times, while *Min Goal (NF)* only fails 7 times). However, the trend is that as that KB grows, the benefits narrow.

Our scoring strategies struggle with the larger sizes of synthetic knowledge bases. The median number of nodes explored does not increase, meaning that most of our queries are still being solved very efficiently. However, the mean number of nodes has increased to above the standard reasoner's.

The results comparing the Min Goal strategies with and without negative facts show mixed results, and the efficiency of these algorithms seems to be highly dependent on the knowledge base. There was an unexpected outcome during the Min Goal without additional negative facts experiment on the second knowledge base of size 375 which caused many of the queries to explore more nodes without failing. This is reflected in the median being much higher than would be expected while the mean is still relatively low compared to the All Goals reasoner. Further analysis will be required to determine the source of these outliers. Is it possible that 100 training queries insufficiently cover the search space for the larger KBs, or perhaps a deeper neural model is needed to learn a high-quality scoring function.

5.3. Lehigh University Benchmark Experiments

Lehigh University Benchmark (LUBM)[16] is a benchmark designed specifically for evaluating the performance of semantic web repositories with respect to use in large universities. The benchmark is intended to facilitate the evaluation of knowledge base systems in terms of scalability, performance, and correctness in handling real-world ontologies. We use the LUBM to evaluate how well our system performs on a larger, more realistic KB.

LUBM consists of a plausible university domain OWL ontology, customizable and repeatable synthetic RDF data, and a set of test queries. The ontology for LUBM describes typical entities in a university setting such as students, professors, courses, and departments. The synthetic data can be generated at various scales, allowing testers to create as large a dataset as needed to test the performance limits of their systems. The benchmark also includes a set of predefined queries that test various aspects of the system's reasoning and query answering abilities. These queries are designed to reflect common queries that might be performed in a university setting, ensuring that the benchmark provides a realistic set of performance challenges. In this paper, we will use the compact N3 syntax to describe RDF and OWL statements.

5.3.1. Design

We performed a manual conversion of the LUBM ontology's rules and facts from the OWL format to the Datalog format suppported by our model. We note that there are generally two ways to represent categories in first-order logic languages; Russell and Norvig [17, Chapter 12] refer to these as the predicate and object representations. We evaluated our system on both types of translations.

1) Category Predicates: These unary predicates take a single argument and represent class membership. In RDF/OWL category membership is expressed by the triple 'I rdf:type C' stating that I is a member of category C; this can be in Datalog by the atom $c(i)$. Subclass triples such as 'C rdfs:subClassOf D' are translated to rules `d(X) :- c(X)`.

2) Predicates as Objects: Treating categories as objects allows queries about the class hierarchy. Special binary predicates are required to express class membership and subclassing. We translate the triple 'I rdf:type C' to `type(i, c)` in Datalog. Similarly, 'A rdfs:subClassOf B' is translated to `subClassOf(a, b)` in Datalog.

OWL is a description logic, and neither description logcs nor Datalog (more generally Horn logic) is more expressive than other. Instead, there is a common intersection. Grosof et al.'s work [18] define this intersection and describe a translation. Building on these mappings, we extract rules from the ontology of LUBM in the OWL version and translate them using predicate categories and category objects in Datalog. Table 3 illustrates the translation of RDFS statements from LUBM into Datalog. Using the Data Generator (UBA) provided by LUBM, we generate OWL data based on the LUBM ontology for a single university department. We then obtain facts from the generated data and apply the same mapping method described in Table 3 to translate these facts from OWL into Datalog. Importantly, we were only able to provide a partial translation of OWL intersections that involved a property restriction.

The original LUBM OWL file was translated into 77 Datalog rules. However, since our reasoning model does not currently support literals, we removed all statements that involved datatype properties

Table 3
Correspondences between RDFS/OWL, Description Logic, and Datalog

RDFS/OWL	Description Logic	Category Predicate
X rdf:type a	X : a	a(X)
X p Y	$\langle X, Y \rangle : p$	p(X,Y)
a rdfs:subClassOf b	a $\sqsubseteq$ b	b(X) :- a(X)
p rdfs:subPropertyOf q	p $\sqsubseteq$ q	q(X,Y) :- p(X,Y)
p rdfs:domain b	$\top \sqsubseteq \forall p^{-1}.b$	b(X) :- p(X,Y)
p rdfs:range b	$\top \sqsubseteq \forall p.b$	b(Y) :- p(X,Y)

p owl:inverseOf q	p $\equiv$ q$^-$	p(X,Y) :- q(Y,X)
		q(X,Y) :- p(Y,X)

p rdf:type owl:TransitiveProperty	p$^+$ $\sqsubseteq$ p	p(X,Z) :- p(X,Y), p(Y,Z)
c owl:intersectionOf $\langle$c1,c2$\rangle$	$c \equiv c1 \sqcap c2$	c(X) :- c1(X), c2(X) c1(X) :- c(X) * c2(X) :- c(X) *
r owl:onProperty p r someValuesFrom c	$\exists p.c$	p(X, Y) :- c(Y)

* Classes representing the intersection of a class and a restriction on an object property may be interpreted as `c(X) :- c1(X), p(X,Y) c2(Y)`. However, the inferred rule `p(X,Y), c2(Y) :- c(X)` cannot be used because the head of the rule must be a single atomic predicate to be consistent with Horn logic.

(except those involving the $name$ property), resulting in an ontology of 48 Datalog rules. The generated data by UBA was translated into 7081 facts. These facts and rules were then combined into a single KB.

To better evaluate the ability of our system to generalize to unseen queries, we creates test queries for the LUBM from a different population than the randomized training queries. Instead, we used the benchmark queries from LUBM as the basis for creating test queries.

LUBM provides a set of 14 test queries in SPARQL 1.0 syntax.

We created a Datalog rule for each query, where the head consisted of the predicate q followed by the query number. The terms of this predicate corresponded to the SELECT clause of the SPARQL query. The body of the rule was formed by translating the WHERE clause and FILTER conditions. Any clauses that relied on excluded Datatype properties were omitted. For example, one of the Datalog rules is

```
q7(X, Y) :- student(X), course(Y), takesCourse(X, Y),
     teacherOf(d0U0_AssociateProfessor0 , Y).
```

We create two versions: one using predicate categories and one using categories as objects. We excluded two queries that had an arity greater than 2, so that our final KB would still only consist of unary and binary predicates. This resulted in a total of 12 queries.

Next, we translate the reference query answers for department 0 into facts in Datalog format and then create the test query dataset based on these facts. We create one hundred test queries by randomly selecting facts and replacing some constants with variables. This approach ensures that the test queries are entirely novel and do not originate from the same population as the training queries, thereby preventing any leakage of training information into the test set. Consequently, we can better evaluate the performance of the model on the unseen dataset.

5.3.2. Results

For each version of the LUBM we produced two sets of training data, one with negative facts and one without. The category predicate representation of LUBM had 115,668 and 195,026 training examples, while the category object representation had 484,116 and 208,202. This difference in example numbers is due to the randomization of the search for answers to training queries during the generation of goal/rule pairs. The category object representation had training data for 16 of the 19 predicates (missing only *name*, *researchInterest* and*teacherOf*). The category predicate representation had training data for 35 of the 39 predicates (it was missing data for the same three predicates as the other representation,

Table 4
Performance comparison of reasoners on LUBM

Representation	Reasoner	Median	Mean	Fails	Time
Category predicates	Standard	101	1,187,537	4	804.6
	All Goals	7	611,991	6	18.2
	Min Goal	7	215,974	2	3.5
	Min Goal (NF)	8	779,224	7	11.9
Category objects	Standard	8	601,586	6	288.2
	All Goals	12	1,900,629	19	17.4
	Min Goal	12	11,260	0	5.4
	Min Goal (NF)	8	4,212	0	12.8

plus the category *publication*). In all cases the missing predicates neither appear in the head nor body of any rule.

The distribution of positive and negative examples by predicate varies. For the category object version, the following predicates only had positive examples: *advisor, doctoralDegreeFrom, mastersDegreeFrom, publicationAuthor* and *undergraduateDegreeFrom.* This was true for both the training sets with and without negative facts. For the category predicate version, the same predicates only had positive examples, and there were an additional 11 categories that only had positive examples.

Training the reasoning models took 12-18 hours each. Our adaptive stopping criteria took between 3000 and 3500 epochs and reached training losses between 0.170 and 0.205.

Table 4 displays the median and mean nodes explored, the number of times the query failed due the node limit, and the average query time in seconds for various reasoners on both translations of the LUBM.

For the category objects representation of LUBM the results show that *Min Goal (NF)* has the best median nodes explored. The time for *Min Goal* was less, but given that it explored nearly 3x as many nodes, we believe this is an aberration due to competing process on the test machine. *All Goals* was clearly the worst, even worse than the standard backward-chaining reasoner. The mean nodes was significantly higher than any other system, mostly because it required over 10,000,000 nodes for 19 queries. However, unlike the results of Jia et al. [1], it also has a worse median (12) than the standard reasoner (8).

We were surprised by the results for the category predicate representation. First, we were expecting this to be the easier knowledge base of the two, since the category object representation added an additional transtive rule for subClassOf and collapsed many category facts into a single type predicate. However, that was not the case, all systems performed better on the category object representation. Further, *Min Goal* performed better than *Min Goal (NF)* with a median nodes of 2 (vs. 8) and mean nodes of 215,964 (vs. 779,224).

5.3.3. Discussion

Closer investigation of *Min Goal (NF)*'s poor performance on the category predicates version revealed that all 7 of the worst performing queries were of the form $q5(a)$ where a is a constant representing a faculty member. The rule for this query is

```
q5(X) :- person(X), memberOf(X, department0University0).
```

Traces of the algorithm show the following sequence of choices:

- It attempts to prove `person(X)` by first using the rule
  `person(X) :- advisor(X,Y)`. This assumes that `X` is the student in the relationship. Note, this goal will eventually fail.
- It then switches to prove `memberOf` using a rule for inverse properties:
  `memberOf(X) :- member(X)`
- It then uses the opposite rule for the same inverse property
  `member(X) :- memberOf(X)`

- It continues repeating the two inverse property rules until it reaches the pre-defined depth limit (15)
- It then backtracks, and eventually proves `memberOf` using `worksFor`, but then fails to prove `advisor(a,Y}`. It repeats this for each attempt to prove `memberOf` in the path.
- After much exploration, it chooses `person(X) :- student(X)` as the way to try to prove a is a person. This repeats a similar process to above.
- Later it encounters another cycle trying to prove a is a student, by proving that a is a person that takes a course. Similar problems repeat through the exploration of 10 million nodes.

This example illustrates that if the learned goal/rule scoring function is incorrect, then our algorithm can still get caught in cycles. For this reason, we believe that future approaches should incorporate some notion of context when training and evaluating goal/rule pairs. Some examples of context might be the depth of the current node or the rule applied.

Another lesson learned is that the negative facts approach from Section 4 does not appear to have much benefit in this use case. Our approach depends on having a conjunction of goals where matching one provides bindings to another. Since all of our training queries are single atoms, any predicate that does not appear as part of a conjunction in the body of a rule has no opportunity for negative facts to be generated. Only nine (of 39) predicates in the category predicate LUBM KB met this condition. A future training regimen should generate plausible conjunctive queries to resolve this limitation.

6. Conclusion

We have continued the exploration of how machine learning can learn a scoring function that can be used by symbolic reasoners. We introduced a new control strategy that uses the scores of candidate goal/rule pairs (g, r) to choose the most selective goal, and show that this is generally better than simply exploring all pairs in descending order of score. We have introduced a modification to the training regimen to get more representative scores by using failed bindings to generate negative facts. However, our experiments demonstrated mixed results on the effectiveness of this approach.

Future work includes exploring strategies to reduce outliers as KB's grow, and looking at mechanisms to introduce more context into the training of the scoring function. The lessons we have learned so far will also be useful in extending this idea to more expressive languages (and associated more complex reasoners) and to determining a useful reinforcement learning approach.

References

[1] Y.-B. Jia, G. Johnson, A. Arnold, J. Heflin, An evaluation of strategies to train more efficient backward-chaining reasoners, in: Proceedings of the 12th Knowledge Capture Conference 2023, 2023, pp. 206–213.

[2] D. Yu, B. Yang, D. Liu, H. Wang, S. Pan, A survey on neural-symbolic learning systems, Neural Networks 166 (2023) 105–126.

[3] G. G. Towell, J. W. Shavlik, Knowledge-based artificial neural networks, Artificial intelligence 70 (1994) 119–165.

[4] J. Xu, Z. Zhang, T. Friedman, Y. Liang, G. Broeck, A semantic loss function for deep learning with symbolic knowledge, in: International Conference on Machine Learning, 2018, pp. 5502–5511.

[5] B. Kijsirikul, T. Lerdlamnaochai, First-Order Logical Neural Networks, International Journal of Hybrid Intelligent Systems 2 (2016) 253–267.

[6] T. Rocktäschel, S. Riedel, End-to-end differentiable proving, in: Advances in Neural Information Processing Systems, 2017, pp. 3788–3800.

[7] W. Cohen, F. Yang, K. R. Mazaitis, TensorLog: A probabilistic database implemented using deep-learning infrastructure, Journal of Artificial Intelligence Research 67 (2020) 285–325.

[8] M. Crouse, I. Abdelaziz, B. Makni, S. Whitehead, C. Cornelio, P. Kapanipathi, K. Srinivas, V. Thost, M. Witbrock, A. Fokoue, A Deep Reinforcement Learning Approach to First-Order Logic Theorem Proving, in: 35th AAAI Conference on Artificial Intelligence, AAAI 2021, volume 7, 2021, pp. 6279–6287. arXiv:1911.02065.

[9] C. Kaliszyk, J. Urban, J. Vyskocil, Efficient semantic features for automated reasoning over large theories, in: Twenty-fourth International Joint Conference on Artificial Intelligence, 2015.

[10] N. Weir, B. Van Durme, Dynamic generation of grounded logical explanations in a neuro-symbolic expert system, arXiv preprint arXiv:2209.07662 (2022).

[11] T. H. Trinh, Y. Wu, Q. V. Le, H. He, T. Luong, Solving olympiad geometry without human demonstrations, Nature (2024) 476–482. doi:10.1038/s41586-023-06747-5.

[12] K. Yang, A. M. Swope, A. Gu, R. Chalamala, P. Song, S. Yu, S. Godil, R. Prenger, A. Anandkumar, Leandojo: Theorem proving with retrieval-augmented language models, 2023. arXiv:2306.15626.

[13] M. Wang, Y. Tang, J. Wang, J. Deng, Premise selection for theorem proving by deep graph embedding, in: Proc. of the 31st Intn'l Conf. on Neural Info. Processing Systems, NIPS'17, Curran Associates Inc., Red Hook, NY, USA, 2017, p. 2783–2793.

[14] J. Jakubův, J. Urban, Enigma: Efficient learning-based inference guiding machine, in: H. Geuvers, M. England, O. Hasan, F. Rabe, O. Teschke (Eds.), Intelligent Computer Mathematics, Springer Intn'l Publishing, Cham, 2017, pp. 292–302.

[15] A. Arnold, J. Heflin, Learning a more efficient backward-chaining reasoner, in: Tenth Annual Conference on Advances in Cognitive Systems (ACS-2022), Cognitive Systems Foundation, Arlington, VA, 2022.

[16] Y. Guo, Z. Pan, J. Heflin, LUBM: A benchmark for OWL knowledge base systems, Journal of Web Semantics 3 (2005) 158–182.

[17] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, third ed., Prentice Hall, Upper Saddle River, NJ, 2010.

[18] B. N. Grosof, I. Horrocks, R. Volz, S. Decker, Description logic programs: Combining logic programs with description logic, in: Proceedings of the 12th international conference on World Wide Web, 2003, pp. 48–57.