Proof Extraction for Logical Neural Networks

Thabang Lebese
University of the Witwatersrand
thabang@aims.ac.za

Ndivhuwo Makondo
IBM Research - Africa
ndivhuwo.makondo@ibm.com

Cristina Cornelio
Samsung AI Cambridge
c.cornelio@samsung.com

Naweed Khan
IBM Research - Africa
naweed.khan@ibm.com

Abstract

Automated Theorem Provers (ATPs) are widely used for the verification of logical statements. Explainability is one of the key advantages of ATPs: providing an expert readable proof path which shows the inference steps taken to conclude correctness. Conversely, Neuro-Symbolic Networks (NSNs) that perform theorem proving, do not have this capability. We propose a proof-tracing and filtering algorithm to provide explainable reasoning in the case of Logical Neural Networks (LNNs), a special type of Neural-Theorem Prover (NTP).

1 Introduction

The ability to explain automated decision-making processes is fundamental to Artificial Intelligent (AI) systems. Recently, explainability has received a lot of attention: a representative example is the 2016 European Union General Data Protection Regulation (EU-GDPR) law. Under this law, AI systems are required to explain their reasoning processes in terms that people can understand. Concrete applications include automated online scoring of credit or mortgage applications, e-recruitment without human intervention, automated computation of insurance quotes, etc. To ensure the respect of ethical decisions and principles of fairness, AI systems deployed in regulated environments are therefore required to explain their reasoning process and justify any decisions reached.

Given their ability to provide an explicit explanation of the reasoning process, classic ATPs are often applied in these types of scenarios. They do so by iteratively applying inference rules to a set of logical axioms and, after a conclusion is reached, highlighting the proof steps that lead to it. Examples of state-of-the-art ATPs that provide this capability are Vampire prover [1], Beagle [2] and E theorem prover [3]. However, the task of extracting an interpretable proof from a reasoning engine, i.e. a proof trace, is non-trivial. For example, the extraction of only meaningful proof steps for an entailment in an OWL ontology has been extensively studied in past years [4].

More recently, the emergence of Neuro-Symbolic AI has resulted in several methods that perform approximate theorem proving using neural networks [5, 6, 7]. Despite being more computationally efficient to perform inference compared to classic ATPs, end-to-end differentiable and sparsely symbolically interpretable [8], these methods also lack an explainable and meaningful proof trace. Some attempts have been done in trying to extract symbolic knowledge from neural networks (e.g., Garcez, *et al.* [9]), however, these methods have not been applied to proof extraction for NTPs.

In this work we aim to fill this gap, providing graphical NSNs with the fundamental capability of producing interpretable explanations for a given query. In particular, we focus on the architecture of Logical Neural Networks (LNNs) [5], a real-valued NTP method that operates over a fragment of first-order logic (FOL) (see the paper introducing LNNs [5] for more details). We extend LNNs with a novel tracking and filtering method that allows for a graphical representation of the proof trace to be extracted; one that is both human interpretable and meaningful.

2 Related work

Classical state-of-the-art ATPs are capable of performing FOL inference while providing full explainability in the form of a proof: a sequence of proof steps that lead to a result. Examples of such theorem provers, resolution-based on FOL with equality, are Vampire [1], Beagle [2] and E-prover [3]. Similar approaches use different inference rules e.g., Tableaux [10], sequent calculus [11], etc. A common feature between these reasoners is that during inference they internally record the proof steps and the dependency structure between thems. At the end of a successful proof attempt, only the minimal set of axioms necessary to prove the input conjecture is stored, while the unnecessary exploration steps are discarded. Finally a linearized proof is returned, usually in the format used for the theorem prover input (e.g., TPTP-3 syntax). This final set of formulae, along with the dependency information stored within, can be used to reconstruct the proof graph.

The extraction of minimal justifications (or minimal set of proof steps) is a well studied research topic in logic, for different expressivity levels, e.g., description logic, OWL (Web Ontology Language), etc. As example, Horridge, *et al*. [4] extract minimal (consisting only of axioms that do not contain any superfluous "parts") justifications for the entailment task in an OWL ontology.

There are several neural methods that attempt to approximate FOL ATPs, however, they do not have the ability to provide such a sequence of interpretable proof steps. Examples include: LNN, a weighted first order logic neural ATP [5]; NTP, a FOL neural ATP with soft unification [6]; Deep ProbLog, a probabilistic first-order logic neural reasoner [7]; Logic Tensor Networks (LTN), a method that takes advantage of the representational power of (multi-valued) FOL [12]; CORGI, a system that performs theorem proving using soft logical inference [13]; embedding approaches that alter NTP algorithms to increase exploration [14]; extending NTPs to construct Conditional Theorem Provers that learn an optimal rule selection strategy via gradient-based optimisation [15] and Topical NTPs, which address scalability using topic pruning strategies [16].

3 Proof extraction

3.1 LNNs overview

LNN [5] is a form of recurrent neural network that creates a 1-to-1 mapping between real-valued logical formulae and specialized neurons (i.e., `And`, `Or`, `Implies` gates) that constrain parameters to "behave logically". Successful computation iteratively tightens truth value bounds for each neuron, via upward and downward inference passes. Bounds tightening is monotonic, forcing inference to converge without oscillation. Similarly to a standard theorem prover, LNN takes a *query*, i.e. a logical formula to be proven, and a *logical theory* as input, i.e. a set of known facts and axioms. In contrast, LNN does not directly check if the *query* is entailed by a given set of formulae and known truth values. Instead, it computes truth values for all formulae in the model, including the *query*, based on the provided *logical theory*. The output is the updated truth value bounds for each neuron in the graph. See Appendix A for more details.

3.2 Proof extraction algorithm

We propose a data structure that keeps track of all updates in the groundings and bounds of all nodes for LNNs. Moreover, it keeps track of sources of updates and time stamps (iteration number). After filtering all idle computations (computations that do not change any bounds), this data is collected, sorted and stored as a proof graph. We present how the inference in LNNs is used by our proof extraction algorithm. The proof extraction process is divided into three main phases, as depicted in Figure 1: **Input phase:** the neural network input is pre-processed and the LNN graph is initialized; **Proof extraction core algorithm:** our algorithm tracks and stores, during inference, all the non-idle computation in a dedicated data structure; **Proof graph construction:** whereby the proof graph is constructed from the stored data and it is automatically converted into a static/temporal-dynamic graphical representation.

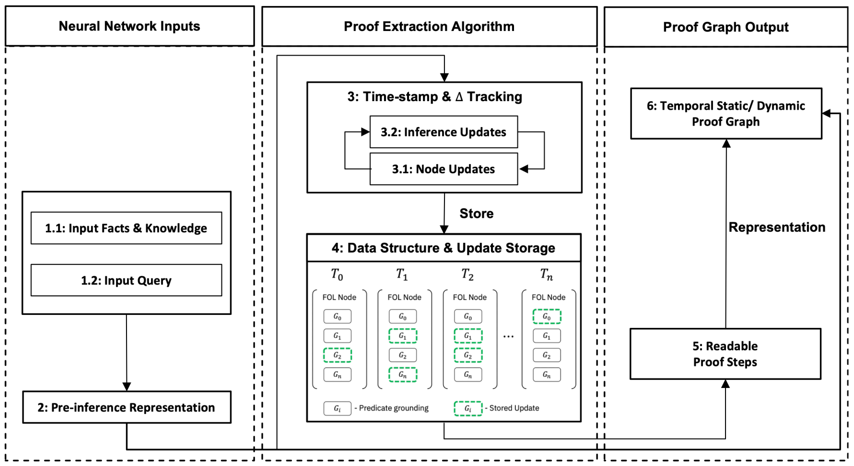

Neural Network Inputs
Proof Extraction Algorithm
Proof Graph Output
3: Time-stamp & Δ Tracking
3.2: Inference Updates
3.1: Node Updates
Store
4: Data Structure & Update Storage
T_0
T_1
T_2
T_n
FOL Node
G_0
G_1
G_2
G_n
G_i - Predicate grounding
G_i - Stored Update
1.1: Input Facts & Knowledge
1.2: Input Query
2: Pre-inference Representation
6: Temporal Static/ Dynamic Proof Graph
Representation
5: Readable Proof Steps

Figure 1: The three main phases for the proof graph extraction algorithm.

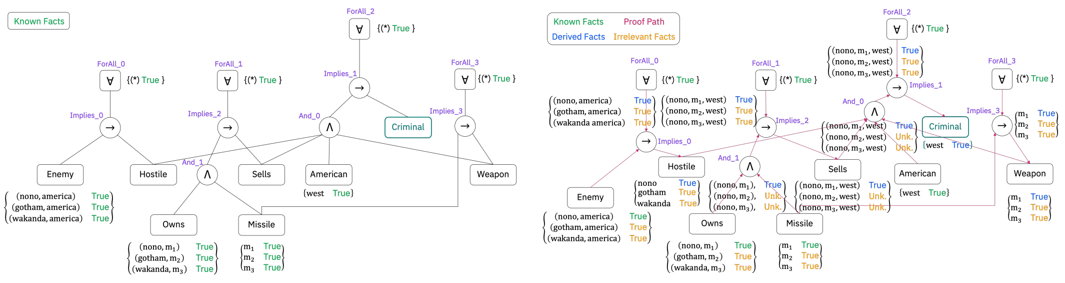
Neural network input - 1. The initial facts, background knowledge and the query are taken as inputs, represented as FOL formulae. **2.** The input is converted to an LNN graphical representation.

Proof extraction algorithm - 3. During inference, the algorithm tracks all sources and targets of computations that yield a bound update. Algorithm 1 demonstrates this in pseudo code. Lines $9-13$ track truth value bounds updates for each neuron per inference step, calculating the respective time stamps and highlighting the nodes that contributed to the bounds update. **3.1.-3.2.** are indicative of the core weighted-logic computation that occur inside LNN during the inference process. **4.** The updates are recorded and sorted, with **4.1.** maintaining an ordering in a *Data structure* that exists within each neuron in the LNN graph.

Proof graph finalization - 5. Provides a domain-expert readable output (i.e., analogous to the output proof of a classic ATP) consisting of the sequence of steps (with temporal information) obtained from the data structure. **6.** A final visual representation of the proof graph is generated, updating the NN-graph in **2** with the proof steps of **5**. The graph is obtained by a recursive search that extracts derived facts and sources of updates, starting at the query node. The search terminates when the remaining sources were provided directly from input facts. This is done as a depth-first search but any exhaustive search strategy is also applicable.

Algorithm 1 Time stamp and bounds tracking algorithm

```
Require: LNN model
Require: LNN query node(s)
 1: T ← 0
 2: for all nodes in LNN do
 3:     $\Delta$ Bounds = 0: for all groundings
 4:     if Bounds is True then
 5:         Source[None]: for all groundings with True bounds
 6: Add entry to database for T = 0
 7: while query is False do
 8:     T ← T + 1
 9:     for all nodes in LNN do
10:         Infer (upward (UI) or downward (DI) inference)
11:         if $\Delta$ Bounds $\neq 0$ for each node then
12:             Store source bounds update for all groundings: Source[source nodes]
13:             Store inference rule: UI or DI
14:     Add entry into database for T
```



(a) LNN at initialization (b) LNN at convergence with proof path

Figure 2: LNN for Example 1. Nodes are predicates (e.g., `Enemy`, `Hostile`, etc.), quantifiers (e.g., `ForAll_2`, etc.) or connectives (e.g., `And_0`, `Implies_1`, etc.). Axioms are subgraphs within the LNN model. Each node has a list of groundings (e.g., `(nono,m1)`) with corresponding truth values, additionally, each quantifier is `True` for all groundings.

4 Experiments

In this section we illustrate and evaluate the performance of our proof extraction algorithm using an example adapted from the book by Russell and Norvig [17]. A simpler example can be found in Appendix B. We show that LNN successfully solves this problem and we provide the extracted proof path to visualize the explanation.

Example 1. **Background knowledge:** The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, and other countries Gotham and Wakanda all have some missiles. Nono's missiles were sold to it by Colonel West, who is American. **Query:** Prove that Colonel West is a criminal.

Example 1 - FOL formulation.
Background knowledge:
$\forall x, y, z(\mathtt{American}(x) \wedge \mathtt{Weapon}(y) \wedge \mathtt{Sells}(x, y, z) \wedge \mathtt{Hostile}(z) \rightarrow \mathtt{Criminal}(x))$
$\forall x(\mathtt{Missile}(x) \wedge \mathtt{Owns}(\mathtt{nono}, x) \rightarrow \mathtt{Sells}(\mathtt{west}, x, \mathtt{nono}))$
$\forall x(\mathtt{Missile}(x) \rightarrow \mathtt{Weapon}(x))$
$\forall x(\mathtt{Enemy}(x, \mathtt{america}) \rightarrow \mathtt{Hostile}(x))$
$\mathtt{Owns}(\mathtt{nono}, \mathtt{m1})$
$\mathtt{Missile}(\mathtt{m1}), \mathtt{Missile}(\mathtt{m2}), \mathtt{Missile}(\mathtt{m3})$
$\mathtt{American}(\mathtt{west})$
$\mathtt{Enemy}(\mathtt{nono}, \mathtt{america}), \mathtt{Enemy}(\mathtt{wakanda}, \mathtt{america}), \mathtt{Enemy}(\mathtt{gotham}, \mathtt{america})$
Query:
$\mathtt{Criminal}(\mathtt{west})?$

Figure 2a shows the LNN at initialization for solving Example 1. The green truth labels correspond to known facts or to the default `True` initialization for the quantifiers. In Figure 2b we show all LNN groundings after the reasoning process has converged. The blue fields correspond to deduced facts, whereas the orange fields indicate those that were irrelevant or did not offer meaningful contributions to the proof. The edges that have changed color from black to red identify the updates that contributed to proving the query/conjecture — all of the edges in this example. Moreover, these undirected edges have been transformed to directed ones, demonstrating the flow of information that was relevant to obtain the final proof.

An interpretation of the proof reads as follows (see **Example 1 - Proof path** and Figure 3): similar to the simple example in Appendix B, there are several facts and rules provided initially. The important facts that prove that Colonel West is a criminal are that he sold (`Sells(nono, m1, west)`, proved in step 12) a weapon (`Weapon(m1)`, proved in step 13) to a hostile (`Hostile(nono)`, proved in step 15) nation – represented by the big conjunction `And_0(nono, m1, west)` which is proved in step 16. Each of the facts contributing to `And_0(nono, m1, west)` were derived from other facts. `m1` deduced to be a weapon because it is a missile, `DI` on the `ForAll_3` quantifier (step 6 and 10). Nono is deduced to be hostile because we know for a fact that it is enemy to America, through `DI` on the `ForAll_0` quantifier (step 8 and 14) and the `Enemy` fact (step 3). `Sells(nono, m1, west)` was deduced (step 12) through `DI` on the `ForAll_1` quantifier (step 7 and 11) and the fact that Nono owns missile `m1` (step 4 and 9). Lastly, West is known to be American for a fact. `DI` on the `ForAll_2` quantifier and the derived facts in `And_0(nono, m1, west)` finally deduce that Colonel West must be a criminal (step 5, 17 and 18).

Example 1 - Proof path

1. T = 0 : American:((west), True) : Input : Source[None]
2. T = 0 : Missile:((m1), True) : Input : Source[None]
3. T = 0 : Enemy:((nono, america), True) : Input : Source[None]
4. T = 0 : Owns:((nono, m1), True) : Input : Source[None]
5. T = 0 : ForAll_2:((∗), True) : Input : Source[None]
6. T = 0 : ForAll_3:((∗), True) : Input : Source[None]
7. T = 0 : ForAll_1:((∗), True) : Input : Source[None]
8. T = 0 : ForAll_0:((∗), True) : Input : Source[None]
9. T = 1 : And_1:((nono, m1), True) : UI : Source[Owns(nono, m1), Missile(m1)]
10. T = 2 : Implies_3:((m1), True) : DI : Source[ForAll_3(∗)]
11. T = 2 : Implies_2:((nono, m1, west), True) : DI : Source[ForAll_1(∗)]
12. T = 2 : Sells:((nono, m1, west), True) : DI :
    Source[And_1(nono,m1), Implies_2(nono, m1, west)]
13. T = 2 : Weapon:((m1), True) : DI : Source[Missile(m1), Implies_3(m1)]
14. T = 2 : Implies_0:((nono, america), True) : DI : Source[ForAll_0(∗)]
15. T = 2 : Hostile:((nono), True) : DI :
    Source[Enemy(nono,america), Implies_0(nono, america)]
16. T = 3 : And_0:((nono, m1, west), True) : UI :
    Source[Hostile(nono), Sells(nono, m1, west), American(west), Weapon(m1)]
17. T = 4 : Implies_1:((nono, m1, west), True) : DI : Source[ForAll_2(∗)]
18. T = 4 : Criminal:((west), True) : DI :
    Source[And_0(nono, m1, west), Implies_1(nono, m1, west)]

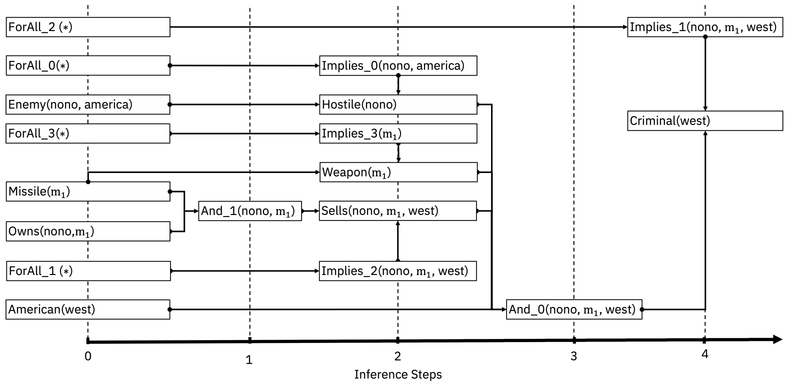


Figure 3: Temporal proof graph for Example 1

5 Conclusion

In this paper we proposed a proof extraction method which provides explainability to LNNs. This is performed by a tracing algorithm that employs a novel data structure for management and storage of non-idle computations. A future direction is to study and understand the quality, rationality and the length of the extracted proof graph, and conduct explainability experiments for larger real-world problems understanding how users interpret the machine reasoning process. Moreover, another interesting direction is the extension of our method to other neuro-symbolic reasoners.

Acknowledgments

The work of Thabang Lebese was conducted during an internship at IBM Research - Africa. The work of Cristina Cornelio was partially conducted while at IBM Research.

References

[1] Laura Kovács and Andrei Voronkov. First-order theorem proving and vampire. In *International Conference on Computer Aided Verification*, pages 1–35. Springer, 2013.

[2] Roni Khardon and Dan Roth. Learning to reason. *Journal of the ACM (JACM)*, 44(5):697–725, 1997.

[3] Peter Baumgartner, Joshua Bax, and Uwe Waldmann. Beagle–a hierarchic superposition theorem prover. In *International Conference on Automated Deduction*, pages 367–377. Springer, 2015.

[4] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Laconic and precise justifications in owl. In *International semantic web conference*, pages 323–338. Springer, 2008.

[5] Ryan Riegel, Alexander Gray, Francois Luus, Naweed Khan, Ndivhuwo Makondo, Ismail Yunus Akhalwaya, Haifeng Qian, Ronald Fagin, Francisco Barahona, Udit Sharma, et al. Logical neural networks. *arXiv preprint arXiv:2006.13155*, 2020.

[6] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. *arXiv preprint arXiv:1705.11040*, 2017.

[7] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *Advances in Neural Information Processing Systems*, 31:3749–3759, 2018.

[8] A.S.A. Garcez, A.S.D.A.G.K.B.D.M. Gabbay, K. Broda, D.M. Gabbay, and A. de Morgan Professor of Logic Dov M Gabbay. *Neural-Symbolic Learning Systems: Foundations and Applications*. Perspectives in Neural Computing. Springer London, 2002.

[9] AS d'Avila Garcez, Krysia Broda, and Dov M Gabbay. Symbolic knowledge extraction from trained neural networks: A sound approach. *Artificial Intelligence*, 125(1-2):155–207, 2001.

[10] Marcelo Finger. Analytic methods for the logic of proofs. *Journal of logic and computation*, 20(1):167–188, 2010.

[11] G. Gentzen. Untersuchungen über das logische schließen i. *Mathematische Zeitschrift*, 39:176–210, 1935.

[12] Luciano Serafini and Artur d'Avila Garcez. Logic tensor networks: Deep learning and logical reasoning from data and knowledge. *arXiv preprint arXiv:1606.04422*, 2016.

[13] Forough Arabshahi, Jennifer Lee, Mikayla Gawarecki, Kathryn Mazaitis, Amos Azaria, and Tom Mitchell. Conversational neuro-symbolic commonsense reasoning. *arXiv preprint arXiv:2006.10022*, 2020.

[14] Michiel de Jong and Fei Sha. Neural theorem provers do not learn rules without exploration. *arXiv preprint arXiv:1906.06805*, 2019.

[15] Pasquale Minervini, Sebastian Riedel, Pontus Stenetorp, Edward Grefenstette, and Tim Rocktäschel. Learning reasoning strategies in end-to-end differentiable proving. In *International Conference on Machine Learning*, pages 6938–6949. PMLR, 2020.

[16] Shuang Xia, Krysia Broda, and Alessandra Russo. Topical neural theorem prover that induces rules. In *GCAI*, pages 107–120, 2020.

[17] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*, chapter 9. Inference in First-Order Logic, pages 330–333. Prentice Hall Press, One Lake Street Upper Saddle River, NJ United States, third edition, 2009.

A LNN overview - extended

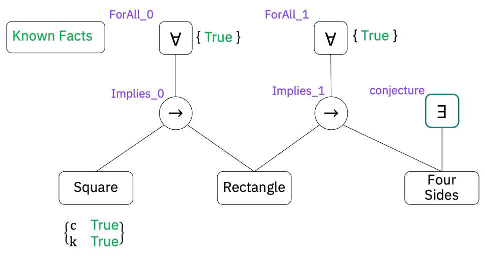
In general, LNNs have three types of nodes: predicates, neural-connectives ($\wedge, \vee, \rightarrow, \neg$) and quantifiers ($\forall, \exists$). Each node has a table of groundings, G, with each grounding $g \in G$ represented as a tuple of constants that are mapped to truth value bounds, $B = [\texttt{Lower}, \texttt{Upper}]$. When operated in a classical system, these bounds offer symbolic interpretability such that $\texttt{True} = [1, 1]$, $\texttt{False} = [0, 0]$, $\texttt{Unknown} = [0, 1]$ and $\texttt{Contradiction} = [1, 0]$. At initialization all known facts and clauses are initialized as `True`, whereas nodes without facts may allow groundings to follow either the open-world (i.e. `Unknown`) or closed-world (i.e. `False`) initialisation assumption. In this paper, we follow the open-world assumption.

Inference utilises a message passing algorithm whereby groundings are propagated, followed by successive 'upward' and 'downward' computations for each node in the graph. 'Upward' refers to the evaluation of operator truth bounds given the truth of all operands and 'downward' refers to the evaluation of a single operand's truth bounds given the truth of the operator and the rest of the operands. The evaluation of truths additionally follows a monotonic tightening of bounds, storing only new information. Each neuron can be individually inspected, thereby allowing a 'query' node to be symbolically interpreted upon convergence of the of the inference strategy. LNNs however, only provide truth value bound outputs and do not provide a proof — the sequence of logical steps taken to reach a converged solution.

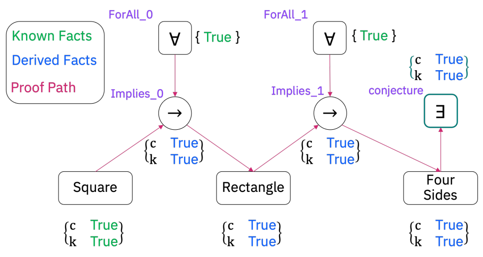
B Example 2: A simple case

Example 2. **Background knowledge:** All square objects are rectangular. All rectangular objects are four-sided. The object c is a square. The object k is a square. **Query:** Is there any object with four sides?

Example 2 - FOL formulation.
Background knowledge:
$\texttt{Square}(\texttt{c})$
$\texttt{Square}(\texttt{k})$
$\forall x(\texttt{Square}(x) \rightarrow \texttt{Rectangle}(x))$
$\forall x(\texttt{Rectangle}(x) \rightarrow \texttt{Foursides}(x))$
Query:
$\exists x \texttt{Foursides}(x)?$



(a) Square-rectangle visualisation at initialization



(b) Square-rectangle visualisation at convergence

Figure 4: LNN for Example 2

This section illustrates our proof extraction algorithm with a simple example. The problem is described in natural language in **Example 2** and in First-order Logic in **Example 2 - FOL formulation**. Figure 4a shows the LNN representation of the problem at initialization. There are two facts known for predicate `Square`, shown by two groundings (`c` and `k`) with corresponding truth value of `True`. In addition, all quantifiers are set to default bounds of `True` for future groundings. These facts known at initialization are labeled in green. During inference computation flows through the edges in both directions (upward inference (UI) and downward inference (DI)), propagating groundings appropriately and computing bounds at each node.

Figure 4b shows the LNN after inference has converged. New facts have been derived (shown in blue) – all relevant to the query in this example – and edges have changed to red to indicate edges traversed by the LNN during inference. The direction of arrows on the edges represents information flow relevant to computing the truth value bounds of the query and therefore forms part of the proof path.

We can extract a proof path for each grounding of the query. **Example 2 - Proof path** shows the proof path for grounding c and Figure 5 shows the proof as a temporal graph. The proof for k can be similarly extracted. Each proof step consists of four components. Firstly, `T` is the number of inference steps it took to compute bounds. The second component is the node being updated (e.g., `Square` in Line 1). The third component is the relevant grounding and corresponding bounds. The fourth component is the type of inference rule: `Input`, `UI`, or `DI`. The application of a single rule on the LNN model is considered a single inference step. The final component is `Source[node1,node2]`, where `node1` and `node2` are nodes from which groundings and bounds for a particular node were computed. `Source[Node]` indicates that no nodes were responsible for bound computation, usually due to the bounds provided as facts.

Example 2 - Proof path

1. $\texttt{T = 0} : \texttt{Square:}((\texttt{c}), \texttt{True}) : \texttt{Input} : \texttt{Source}[\texttt{None}]$
2. $\texttt{T = 0} : \texttt{Forall0:}((*), \texttt{True}) : \texttt{Input} : \texttt{Source}[\texttt{None}]$
3. $\texttt{T = 0} : \texttt{Forall1:}((*), \texttt{True}) : \texttt{Input} : \texttt{Source}[\texttt{None}]$
4. $\texttt{T = 2} : \texttt{Implies0:}((\texttt{c}), \texttt{True}) : \texttt{DI} : \texttt{Source}[\texttt{Forall0}(*)]$
5. $\texttt{T = 2} : \texttt{Implies1:}((\texttt{c}), \texttt{True}) : \texttt{DI} : \texttt{Source}[\texttt{Forall1}(*)]$
6. $\texttt{T = 2} : \texttt{Rectangle:}((\texttt{c}), \texttt{True}) : \texttt{DI} : \texttt{Source}[\texttt{Implies0}(\texttt{c}), \texttt{Square}(\texttt{c})]$
7. $\texttt{T = 4} : \texttt{Foursides:}((\texttt{c}), \texttt{True}) : \texttt{DI} : \texttt{Source}[\texttt{Implies1}(\texttt{c}), \texttt{Rectangle}(\texttt{c})]$
8. $\texttt{T = 5} : \texttt{conjecture:}((\texttt{c}), \texttt{True}) : \texttt{UI} : \texttt{Source}[\texttt{Foursides}(\texttt{c})]$

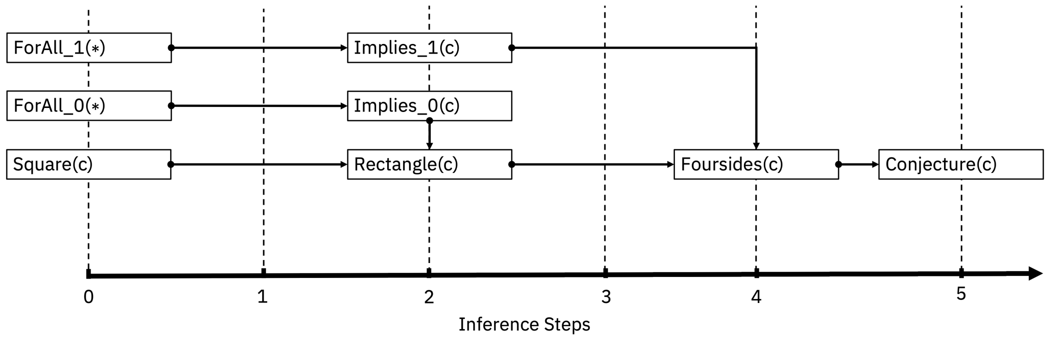


Figure 5: Temporal proof graph for Example 2

An interpretation of the proof reads as follows: At `T = 0` we are given as facts that object `c` is square, and that any square object is rectangular and any rectangular object has four sides. `UI` at `T = 1` does not deduce any new facts. Then at `T = 2` we deduce that the two logic rules apply for object `c` through `DI`, and also that `c` must be a square through `DI` on the first logic rule[1]. At `T = 3` no bounds are updated, thus no new facts are deduced. Then at `T = 4` we deduce that `c` has four sides through `DI` on the second logic rule, concluding the conjecture through `UI` at `T = 5`.

[1]This corresponds to modus ponens in classical logical rules