Logical Neural Networks for Automated Theorem Proving

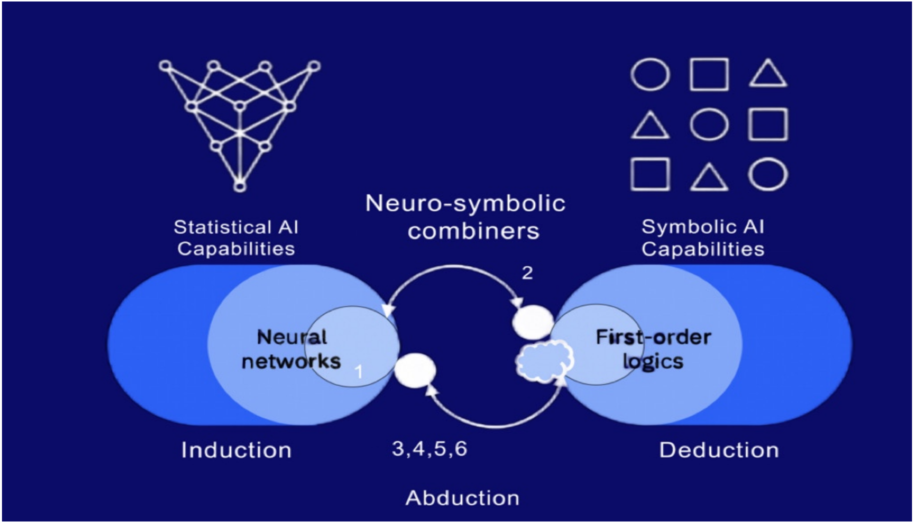
Hyunho James Choe [2], Zhifeng Guo [1], Xinliang He [1], Shouyi Lin [2], Zachary Tan [2], Ruzica Vuckovic [1], Stephanie Wang [1]

[1]Department of Mathematics
University of Rochester

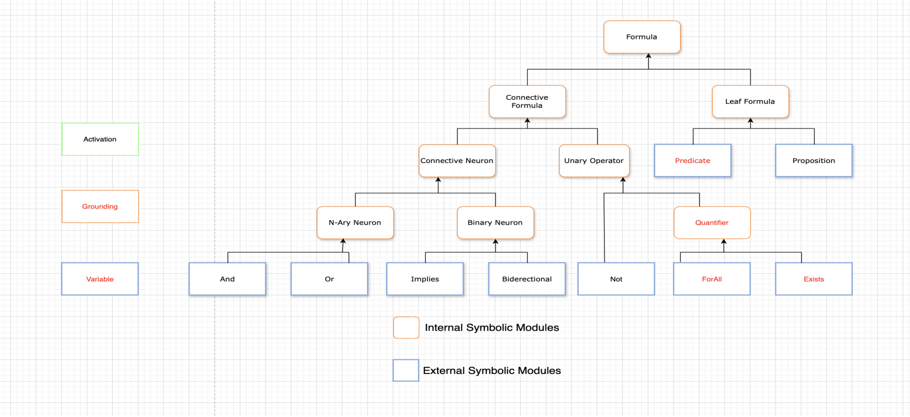
[2]Department of Computer Science
University of Rochester

STEMFORALL Final Presentation, August 2024

Introduction: What is Neuro-Symbolic AI



Overview: Logical Neural Network Structure

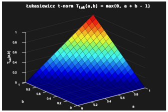


Theorem 1. Given monotonic $\neg$, $\oplus$, and f, Algorithm 3 converges to within ϵ in finite time.

Neurons, Activation Function, Fuzzy Logic

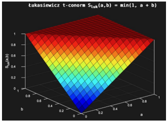
1. The n-ary weighted nonlinear conjunctions, used for logical AND gate:

$$\beta\left(\bigotimes_{i\in I} x_i^{\oplus w_i}\right) = f\left(\beta - \sum_{i\in I} w_i(1 - x_i)\right)$$



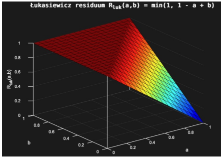
2. The n-ary weighted nonlinear disjunctions, used for logical OR:

$$\beta\left(\bigotimes_{i\in I} x_i^{\oplus w_i}\right) = f\left(1 - \beta + \sum_{i\in I} w_i x_i\right)$$



3. The weighted nonlinear residuum, used for logical IMPLICATION:

$$\beta\left(x^{\oplus w_x} \to y^{\oplus w_y}\right) = f\left(1 - \beta + w_x(1 - x) + w_y y\right)$$



Bidirectional Inference

Inference rules in weighted nonlinear logic are incorporated by setting the bounds for computations for each logical operation:

Bidirectional Inference

Inference rules in weighted nonlinear logic are incorporated by setting the bounds for computations for each logical operation:
The bounds computations for $\neg$:

$$L_{\neg x} \geq \neg U_x = 1 - U_x, \quad U_{\neg x} \leq \neg L_x = 1 - L_x,$$

$$L_x \geq \neg U_{\neg x} = 1 - U_{\neg x}, \quad U_x \leq \neg L_{\neg x} = 1 - L_{\neg x}.$$

Bidirectional Inference

Inference rules in weighted nonlinear logic are incorporated by setting the bounds for computations for each logical operation:
The bounds computations for ¬:

$$L_{\neg x} \geq \neg U_x = 1 - U_x, \quad U_{\neg x} \leq \neg L_x = 1 - L_x,$$

$$L_x \geq \neg U_{\neg x} = 1 - U_{\neg x}, \quad U_x \leq \neg L_{\neg x} = 1 - L_{\neg x}.$$

Observing that, in weighted nonlinear logic:

$$\beta\left(x^{\bigotimes w_x} \to y^{\bigoplus w_y}\right) = \beta\left((1-x)^{\bigoplus w_x} \oplus y^{\bigoplus w_y}\right)$$

$$\text{and} \quad \beta\left(\bigotimes_{i \in I} x_i^{\bigoplus w_i}\right) = 1 - \beta\left(\bigoplus_{i \in I} (1 - x_i)^{\bigoplus w_i}\right)$$

Bidirectional Inference

Inference rules in weighted nonlinear logic are incorporated by setting the bounds for computations for each logical operation:
The bounds computations for $\neg$:

$$L_{\neg x} \geq \neg U_x = 1 - U_x, \quad U_{\neg x} \leq \neg L_x = 1 - L_x,$$

$$L_x \geq \neg U_{\neg x} = 1 - U_{\neg x}, \quad U_x \leq \neg L_{\neg x} = 1 - L_{\neg x}.$$

Observing that, in weighted nonlinear logic:

$$\beta\left(x^{\bigotimes w_x} \to y^{\bigoplus w_y}\right) = \beta\left((1-x)^{\bigoplus w_x} \oplus y^{\bigoplus w_y}\right)$$

$$\text{and} \quad \beta\left(\bigotimes_{i \in I} x_i^{\bigoplus w_i}\right) = 1 - \beta\left(\bigoplus_{i \in I}(1-x_i)^{\bigoplus w_i}\right)$$

It follows that:

Bidirectional Inference

Inference rules in weighted nonlinear logic are incorporated by setting the bounds for computations for each logical operation:
The bounds computations for $\neg$:

$$L_{\neg x} \geq \neg U_x = 1 - U_x, \quad U_{\neg x} \leq \neg L_x = 1 - L_x,$$

$$L_x \geq \neg U_{\neg x} = 1 - U_{\neg x}, \quad U_x \leq \neg L_{\neg x} = 1 - L_{\neg x}.$$

Observing that, in weighted nonlinear logic:

$$\beta\left(x^{\otimes w_x} \rightarrow y^{\oplus w_y}\right) = \beta\left((1-x)^{\oplus w_x} \oplus y^{\oplus w_y}\right)$$

$$\text{and} \quad \beta\left(\bigotimes_{i \in I} x_i^{\oplus w_i}\right) = 1 - \beta\left(\bigoplus_{i \in I} (1 - x_i)^{\oplus w_i}\right)$$

It follows that:
The upward bounds computations for $\oplus$:

$$L_{\oplus} x_i \geq \beta\left(\bigoplus_{i \in I} L_{\oplus x_i}^{\oplus w_i}\right), \quad U_{\oplus} x_i \leq \beta\left(\bigoplus_{i \in I} U_{\oplus x_i}^{\oplus w_i}\right)$$

Bidirectional Inference

Inference rules in weighted nonlinear logic are incorporated by setting the bounds for computations for each logical operation:
The bounds computations for $\neg$:

$$L_{\neg x} \geq \neg U_x = 1 - U_x, \quad U_{\neg x} \leq \neg L_x = 1 - L_x,$$

$$L_x \geq \neg U_{\neg x} = 1 - U_{\neg x}, \quad U_x \leq \neg L_{\neg x} = 1 - L_{\neg x}.$$

Observing that, in weighted nonlinear logic:

$$\beta\left(x^{\otimes w_x} \rightarrow y^{\oplus w_y}\right) = \beta\left((1-x)^{\oplus w_x} \oplus y^{\oplus w_y}\right)$$

$$\text{and} \quad \beta\left(\bigotimes_{i\in I} x_i^{\oplus w_i}\right) = 1 - \beta\left(\bigoplus_{i\in I}(1-x_i)^{\oplus w_i}\right)$$

It follows that:
The upward bounds computations for $\oplus$:

$$L_{\oplus} x_i \geq \beta\left(\bigoplus_{i\in I} L_{\oplus x_i}^{\oplus w_i}\right), \quad U_{\oplus} x_i \leq \beta\left(\bigoplus_{i\in I} U_{\oplus x_i}^{\oplus w_i}\right)$$

The downward bounds computations for $\oplus$:

$$L_{x_i} \geq \beta^{1/w_i}\left(\bigotimes_{j\neq i}\left(1 - U_{x_j}\right)^{\oplus w_j/w_i} \otimes L_{\oplus} x_i^{1/w_i}\right) \quad \textbf{if} \quad L_{\oplus} x_i > 1-\alpha, \quad \textbf{else} \quad 0$$

$$U_{x_i} \leq \beta^{1/w_i}\left(\bigotimes_{j\neq i}\left(1 - L_{x_j}\right)^{\oplus w_j/w_i} \otimes U_{\oplus} x_i^{1/w_i}\right) \quad \textbf{if} \quad U_{\oplus} x_i < \alpha, \quad \textbf{else} \quad 1$$

Note: α is a threshold determined by f to address potential divergent behavior at $L_{x_i} \leq 1-\alpha$ and $U_{x_i} \geq \alpha$.

Recurrent Inference

Algorithm 1 Recurrent inference procedure with recursive directional graph traversal (Algorithm 3)

```
function INFERENCE(formula z)
while ∑(|δL_z| + |δU_z|) > ϵ do
    for r ∈ roots(z) do
        UPWARDPASS(r) {leaves-to-root traversal}
        DOWNWARDPASS(r) {root-to-leaves traversal}
    end for
end while
end function =0
```

Upward Pass

Algorithm 2 Upward Pass (Algorithm 1): Infer formula truth value bounds from subformula bounds

function UPWARDPASS(formula z)
for operand z_i in z **do**
  if $z = \neg x$ **then**
    AGGREGATE(z_i, $(1 - U_x, 1 - L_x)$) {negation}
  else if $z = \bigoplus_{i \in I} x_i^{\oplus w_i}$ **then**
    AGGREGATE(z_i, $(\bigoplus_{i \in I} L_{\oplus x_i}^{\oplus w_i}, \bigoplus_{i \in I} U_{\oplus x_i}^{\oplus w_i})$) {multi-input disjunction}
  end if
end for
AGGREGATE(z, (L'_z, U'_z)) {tighten existing bounds}
end function =0

Downward Pass

Algorithm 3 Downward Pass (Algorithm 2): Infer subformula truth value bounds from formula bounds

function DOWNWARDPASS(formula z)
for operand x_j in z **do**
  if $z = \neg x$ **then**
    AGGREGATE$(x, (1 - U_z, 1 - L_z))$ {negation}
  else if $z = \beta\left(\bigoplus_{i \in I} L_{\oplus x_i}^{\oplus w_i}\right)$ **then**
    $L'_{x_j} := \beta^{1/w_j}\left(\bigotimes_{j \neq i}(1 - U_{x_j})^{\oplus w_j/w_i} \otimes L_{\oplus} {x_i}^{1/w_i}\right)$ {if $L_{\oplus} x_i > 1 - \alpha$, else 0}
    $U'_{x_j} := \beta^{1/w_j}\left(\bigotimes_{j \neq i}(1 - L_{x_j})^{\oplus w_j/w_i} \otimes U_{\oplus} {x_i}^{1/w_i}\right)$ {if $U_{\oplus} x_i < \alpha$, else 1}
  end if
end for
AGGREGATE$(x_j, (L'_{x_j}, U'_{x_j}))$ {propagate bounds downward to leaves}
end function =0

Learning

Loss function:

$$\min_{B,W} E(B, W) + \sum_{k \in N} \max\{0, L_{B,W,k} - U_{B,W,k}\}$$

$$\text{s.t.} \quad \forall k \in N,\ i \in I_k, \quad \alpha \cdot w_{ik} - \beta_k + 1 \geq \alpha, \quad w_{ik} \geq 0$$

$$\forall k \in N, \quad \sum_{i \in I_k} (1 - \alpha) \cdot w_{ik} - \beta_k + 1 \leq 1 - \alpha, \quad \beta_k \geq 0$$

Learning

Loss function:

$$\min_{B,W} E(B, W) + \sum_{k \in N} \max\{0, L_{B,W,k} - U_{B,W,k}\}$$

$$\text{s.t.} \quad \forall k \in N,\ i \in I_k, \quad \alpha \cdot w_{ik} - \beta_k + 1 \geq \alpha, \quad w_{ik} \geq 0$$

$$\forall k \in N, \quad \sum_{i \in I_k} (1 - \alpha) \cdot w_{ik} - \beta_k + 1 \leq 1 - \alpha, \quad \beta_k \geq 0$$

Updated Loss Function:

$$\min_{B,W,S} E(B, W) + \sum_{k \in N} \max\{0, L_{B,W,k} - U_{B,W,k}\} + \sum_{k \in N} S_k \cdot W_k$$

$$\text{s.t.} \quad \forall k \in N,\ i \in I_k, \quad \alpha \cdot w_{ik} - s_{ik} - \beta_k + 1 \geq \alpha, \quad w_{ik}, s_{ik} \geq 0$$

$$\forall k \in N, \quad \sum_{i \in I_k} (1 - \alpha) \cdot w_{ik} - \beta_k + 1 \leq 1 - \alpha, \quad \beta_k \geq 0$$

]

Tailored Activation Function:

$$f_w(x) = \begin{cases} x \cdot \frac{(1-\alpha)}{x_F}, & \text{if } 0 \leq x \leq x_F, \\ (x - x_F) \cdot \frac{(2\alpha-1)}{(x_T - x_F)} + 1 - \alpha, & \text{if } x_F < x < x_T, \\ (x - x_T) \cdot \frac{(1-\alpha)}{(x_{\max} - x_T)} + \alpha, & \text{if } x_T \leq x \leq x_{\max}, \end{cases}$$

$$x_F = \sum_{i \in I} w_i \cdot (1 - \alpha), \quad x_T = w_{\max} \cdot \alpha, \quad x_{\max} = \sum_{i \in I} w_i$$

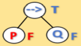
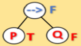
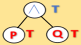
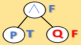
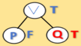
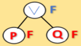
Gradient Transparent Clamping in Fuzzy Logic:

$$\bigotimes_{i\in I}^{\beta} x_i^{\otimes w_i} = \max\left(0, \min\left(1, \beta - \sum_{i\in I} w_i(1 - x_i)\right)\right), \tag{1}$$

$$\frac{\partial\left(\bigotimes_{i\in I}^{\beta} x_i^{\otimes w_i}\right)}{\partial\beta} = \begin{cases} 1 & \text{if } 0 \leq \bigotimes_{i\in I}^{\beta} x_i^{\otimes w_i} \leq 1, \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

$$\frac{\partial\left(\bigotimes_{i\in I}^{\beta} x_i^{\otimes w_i}\right)}{\partial w_i} = \begin{cases} (x_i - 1) & \text{if } 0 \leq \bigotimes_{i\in I}^{\beta} x_i^{\otimes w_i} \leq 1, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Empirical Evaluation

Rule	Logical Deduction	LNN Reasoning	Syntax Tree
Modus Pones	P P --> Q Q	If Operand is True and the operation --> is True	
Modus Tollens	¬Q P --> Q ¬P	If Operand is False and the operation --> is True	
Absorption	¬(P --> Q) P , ¬Q	If Operand is True and the operation --> is False	
Conjunctive Elimination	P ∧ Q P , Q	If Operand is True and the operation ∧ is True	
Modus Ponendo Tollens	P ¬(P ∧ Q) ¬Q	If Operand is False and the operation ∧ is False	
Disjunctive Syllogism	¬P P ∨ Q Q	If Operand is True and the operation ∨ is True	
De Morgan's	¬(P ∨ Q) ¬P , ¬Q	If Operand is False and the operation ∨ is False	

```
LawsOfInference = {
    (True, True, '→'): "Modus Ponens",
    (True, True, '∧'): "Conjunctive Elimination",
    (True, True, '∨'): "Disjunctive Syllogism",
    (False, False, '∨'): "De Morgan's Law",
    (False, False, '∧'): "Modus Ponendo Tollens",
    (False, True, '→'): "Modus Tollens",
    (True, False, '→'): "Absorption 1",
    (False, False, '→'): "Absorption 2"
}
```

Empirical Evaluation

```python
# Detect operator neuron type
operatorType = None                                      # Initialize operatorType
if _isinstance(self, "And"): operatorType = '∧'          # Check Conjunction (And) [ ∧ ]
elif _isinstance(self, "Or"): operatorType = '∨'         # Check Disjunction (Or) [ ∨ ]
elif _isinstance(self, "Implies"): operatorType = '→'    # Check Implication (Implies) [ → ]

# Add law of inference to solution steps in Printer if operator type is 'And', 'Or', or 'Implies'
if operatorType is not None:
    forTruth = op.state(to_bool=True)                                            # Set 'forTruth' as the operator neuron's truth value
    fromTruth = self.state(to_bool=True)                                         # Set 'fromTruth' as the target operand neuron's truth value
    # Find and add correct law of inference used for downward inference to solution steps in Printer if 'forTruth' and 'fromTruth' are not UNKNOWN
    if type(forTruth) == bool and type(fromTruth) == bool:
        rule = LawsOfInference[(forTruth, fromTruth, operatorType)]              # Check Laws of Inference dictionary for mapping to correct rule used
        addSolStep_Derivation((op.name, forTruth), rule, (self.name, fromTruth)) # Add correct rule to solution steps in Printer
    else:
        forTruth =  forTruth if type(forTruth) == bool else forTruth.name        # Convert 'forTruth' to its string if it is still an enumerator
        fromTruth = fromTruth if type(fromTruth) == bool else fromTruth.name     # Convert 'fromTruth' to its string if it is still an enumerator
        addSolStep_Derivation((op.name, forTruth), "CONTRADICTION", (self.name, fromTruth))  # Add CONTRADICTION to solution steps in Printer
```

Empirical Evaluation

```
# Initializing Global Variables
contradictionFound: bool = False        # Global variable to track if a contradiction is found during inference

def foundContradiction() -> None:
    """
    Sets global variable 'contradictionFound' to True if it is False
    ________________________________________________________________
    * Used to mark whether the model encountered a contradiction during inference
    """

    global contradictionFound       # Call global variable 'contradictionFound'
    contradictionFound = True       # Set g.v. 'contradictionFound' to True
```

Figure: foundContradiction()

Empirical Evaluation

```
def resetAndRunModel(model: Model, premises: List[Formula], query: Formula, queryTV, printB: bool, detailed: bool, file: bool) -> Tuple[Tuple[int, int], Any]:
    """
    Resets Model, sets query to a certain truth value, and runs inference on it

    ___________________________________________________________________________
    Parameters

    __________
    model:      {Model}             (REQUIRED)      -> The model to reset and run inference on                          \n
    premises:   {List[Formula]}     (REQUIRED)      -> List of premises in model                                        \n
    query:      {Formula}           (REQUIRED)      -> The query to add to the model                                    \n
    queryTV:    {enum: Fact}        (REQUIRED)      -> The truth value to set query to                                  \n
    printB:     {bool}              (DEFAULT: True) -> Boolean to determine whether to print or not                     \n
    detailed:   {bool}              (DEFAULT: True) -> Boolean to determine whether to show details or not              \n
    file:       {bool}              (DEFAULT: True) -> Boolean to determine whether to print to file or to Terminal instead    \n
    Returns

    _______
    (steps, facts_inferred)     {Tuple[Tuple[int, int], Any]}    -> Returns # of steps and facts inferred
    """

    model.flush()                                                               # Flush Model rests all neurons to Fact.UNKNOWN
    method = "Contradiction" if queryTV==Fact.FALSE else "Adonis"               # Adjust method name to reflect proof method
    model = Model(name=f"{model.name} With {method}")                           # Reinitialize Model

    for premise in premises:                                    # Iterate through list of premises
        model.add_knowledge(premise, world=World.AXIOM)         # Add each premise to the Model's knowledge base as an axiom

    query.add_data(queryTV)         # Set query to Fact.FALSE
    model.add_knowledge(query)      # Add query to Model's knowledge base

    # Print the Model BEFORE inference if 'printB' argument is True
    if printB: Printer.print_BeforeInfer(file=file, model=model, query=query, detailed=detailed)

    # Run upward and downward inference passes on Model until it converges on solution or no new knowledge is discovered
    steps, facts_inferred = model.infer()

    Printer.print_solution_steps()      # Print all solution steps to 'Proof.txt' file

    return steps, facts_inferred
```

Figure: resetAndRunModel()

Empirical Evaluation

```python
def prove(model: Model, premises: List[Formula], query: Formula, printB: bool=True, detailed: bool=True, file: bool=True, graph: bool=True) -> None:
    """
    Runs proof algorithm on Model
    ______________________________
    * Starts with Direct Proof, checks Principle of Explosion, tries Proof by Contradiction, tries Proof by Adonis, concludes inconclusive if all fails
    * Can print results to File or Terminal
    * Can generate visualization of Model

    Parameters
    __________
    model:      {Model}             (REQUIRED)      -> The model to run inference on                                         \n
    premises:   {List[Formula]}     (REQUIRED)      -> List of premises in model                                             \n
    query:      {Formula}           (REQUIRED)      -> The query to add to the model                                         \n
    printB:     {bool}              (DEFAULT: True) -> Boolean to determine whether to print or not                          \n
    detailed:   {bool}              (DEFAULT: True) -> Boolean to determine whether to show details or not                   \n
    file:       {bool}              (DEFAULT: True) -> Boolean to determine whether to print to file or to Terminal instead  \n
    graph:      {bool}              (DEFAULT: True) -> Boolean to determine whether to generate and show graph or not        \n
    """

    # <-------------------- INITIALIZATION ------------------->

    global contradictionFound       # Call global variable 'contradictionFound'
    contradictionFound = False      # Set g.v. 'contradictionFound' to False

    for premise in premises:                                    # Iterate through list of premises
        model.add_knowledge(premise, world=World.AXIOM)         # Add each premise to the Model's knowledge base as an axiom

    Printer.initPrinter(name=model.name)                                                # Initialize Printer with Model's name
    Printer.print_ModelInfo(name=model.name, premises=premises, query=query)            # Print Model's information to 'Proof.txt' file
```

Figure: prove()

Empirical Evaluation

```
# <------------------- DIRECT PROOF ------------------->

model.set_query(query)      # Set query in model as Fact.UNKNOWN

# Print the Model BEFORE inference if 'printB' argument is True
#      * Prints to file if 'file' arg. is True, otherwise prints to Terminal
#      * Prints detailed information if 'detailed' arg. is True
if printB: Printer.print_BeforeInfer(file=file, model=model, query=query, detailed=detailed)

# Run upward and downward inference passes on Model until it converges on solution or no new knowledge is discovered
#      * Receives number of steps and number of facts inferred
#      * Saves inference steps to Printer
#      * Updates g.v. 'contradictionFound' if contradiction encountered
steps, facts_inferred = model.infer()

Printer.print_solution_steps()      # Print all solution steps to 'Proof.txt' file

queryTV = query.state()     # Save query's truth value after inference to 'queryTV'
```

Figure: prove() continued...

Empirical Evaluation

```python
# <------------------- PRINCIPLE OF EXPLOSION CHECK ------------------->

# Conclude Principle of Explosion if contradiction found during Direct Proof
#       * Determines the premises are inconsistent and therefore query must be True
if contradictionFound:
    query.add_data(Fact.TRUE)                   # Set query to True
    Printer.principleOfExplosion(query.name)    # Print Principle of Explosion conclusion to 'Proof.txt'

# Conclude Direct Proof if Model converges on solution for query and no contradictions found
elif queryTV != Fact.UNKNOWN:
    Printer.concludeProof(True if queryTV==Fact.TRUE else False, query.name)

# <------------------- PROOF BY CONTRADICTION ------------------->

# Try Proof by Contradiction if Model does not prove query using Direct Proof or Principle of Explosion Check
elif queryTV == Fact.UNKNOWN:
    Printer.startContradiction(query)       # Setup Proof by Contradiction in 'Proof.txt'

    steps, facts_inferred = resetAndRunModel(model=model, premises=premises, query=query, queryTV=Fact.FALSE, printB=printB, detailed=detailed, file=file)    # Reset Model, set query to a False, and run inference

    # Conclude Proof by Contradiction if Model converges on solution for query
    if contradictionFound:
        Printer.concludeContradiction(query.name)

    # <------------------- PROOF BY GUO ------------------->

    # Try Proof by Guo if Model does not prove query using Direct Proof, Principle of Explosion Check, or Proof by Contradiction
    else:
        Printer.startGuo(query)      # Setup Proof by Guo in 'Proof.txt'

        steps, facts_inferred = resetAndRunModel(model=model, premises=premises, query=query, queryTV=Fact.TRUE, printB=printB, detailed=detailed, file=file)    # Reset Model, set query to a True, and run inference

        # Conclude Proof by Guo
        #       * Determines the premises are inconsistent if contradiction found during Proof by Guo
        #       * Determines inconclusive otherwise
        Printer.concludeGuo(contradictionFound, query.name)
```

Figure: prove() continued...

Empirical Evaluation

```python
# <------------------- FINALIZATION ------------------->

# Print the Model AFTER inference if 'printB' argument is True
#      * Prints to file if 'file' arg. is True, otherwise prints to Terminal
#      * Prints detailed information if 'detailed' arg. is True
if printB:
    Printer.print_AfterInfer(file=file, model=model, query=query, detailed=detailed, steps=steps, facts_inferred=facts_inferred)

# Generate and show a visual plot of the Model (Directed Acyclic Graph) if 'graph' is True
if graph:
    model.plot_graph()

print(f"\nCompleted Inference on {model.name} Model!")
```

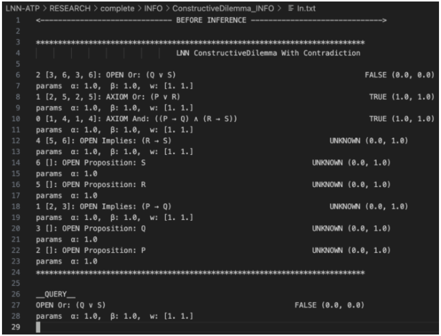
Figure: prove() continued...

Constructive Dilemma: Code

LNN-ATP > RESEARCH > complete > ConstructiveDilemma.py > ...

```python
from lnn import *
from helper import Executor

def ConstructiveDilemma():
    # Initialize Model
    model = Model(name="ConstructiveDilemma")

    # Initialize Propositions
    P,Q,R,S = Propositions('P','Q','R','S')

    # Define Premises
    premise1 = And(
        Implies(P, Q),
        Implies(R, S)
    )
    premise2 = Or(P, R)

    # Add Premises to List
    premises = [premise1, premise2]

    # Define Query
    query = Or(Q, S)

    # Run Proof Algorithm on Model
    Executor.prove(model=model, premises=premises, query=query)

if __name__ == "__main__":
    ConstructiveDilemma()
```

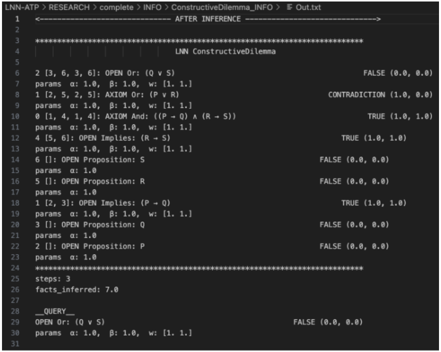
Constructive Dilemma: Output



LNN-ATP > RESEARCH > complete > INFO > ConstructiveDilemma_INFO > In.txt

```
<----------------------------- BEFORE INFERENCE ----------------------------->

******************************************************************************
                                 LNN ConstructiveDilemma With Contradiction

2 [3, 6, 3, 6]: OPEN Or: (Q ∨ S)                                    FALSE (0.0, 0.0)
params  α: 1.0,  β: 1.0,  w: [1. 1.]
1 [2, 5, 2, 5]: AXIOM Or: (P ∨ R)                                    TRUE (1.0, 1.0)
params  α: 1.0,  β: 1.0,  w: [1. 1.]
0 [1, 4, 1, 4]: AXIOM And: ((P → Q) ∧ (R → S))                       TRUE (1.0, 1.0)
params  α: 1.0,  β: 1.0,  w: [1. 1.]
4 [5, 6]: OPEN Implies: (R → S)                             UNKNOWN (0.0, 1.0)
params  α: 1.0,  β: 1.0,  w: [1. 1.]
6 []: OPEN Proposition: S                               UNKNOWN (0.0, 1.0)
params  α: 1.0
5 []: OPEN Proposition: R                               UNKNOWN (0.0, 1.0)
params  α: 1.0
1 [2, 3]: OPEN Implies: (P → Q)                             UNKNOWN (0.0, 1.0)
params  α: 1.0,  β: 1.0,  w: [1. 1.]
3 []: OPEN Proposition: Q                               UNKNOWN (0.0, 1.0)
params  α: 1.0
2 []: OPEN Proposition: P                               UNKNOWN (0.0, 1.0)
params  α: 1.0
******************************************************************************

__QUERY__
OPEN Or: (Q ∨ S)                                  FALSE (0.0, 0.0)
params  α: 1.0,  β: 1.0,  w: [1. 1.]
```

Figure: In.png



LNN-ATP > RESEARCH > complete > INFO > ConstructiveDilemma_INFO > Out.txt

```
<----------------------------- AFTER INFERENCE ----------------------------->

******************************************************************************
                                 LNN ConstructiveDilemma

2 [3, 6, 3, 6]: OPEN Or: (Q ∨ S)                                    FALSE (0.0, 0.0)
params  α: 1.0,  β: 1.0,  w: [1. 1.]
1 [2, 5, 2, 5]: AXIOM Or: (P ∨ R)                            CONTRADICTION (1.0, 0.0)
params  α: 1.0,  β: 1.0,  w: [1. 1.]
0 [1, 4, 1, 4]: AXIOM And: ((P → Q) ∧ (R → S))                       TRUE (1.0, 1.0)
params  α: 1.0,  β: 1.0,  w: [1. 1.]
4 [5, 6]: OPEN Implies: (R → S)                                 TRUE (1.0, 1.0)
params  α: 1.0,  β: 1.0,  w: [1. 1.]
6 []: OPEN Proposition: S                                  FALSE (0.0, 0.0)
params  α: 1.0
5 []: OPEN Proposition: R                                  FALSE (0.0, 0.0)
params  α: 1.0
1 [2, 3]: OPEN Implies: (P → Q)                                 TRUE (1.0, 1.0)
params  α: 1.0,  β: 1.0,  w: [1. 1.]
3 []: OPEN Proposition: Q                                  FALSE (0.0, 0.0)
params  α: 1.0
2 []: OPEN Proposition: P                                  FALSE (0.0, 0.0)
params  α: 1.0
******************************************************************************
steps: 3
facts_inferred: 7.0

__QUERY__
OPEN Or: (Q ∨ S)                                  FALSE (0.0, 0.0)
params  α: 1.0,  β: 1.0,  w: [1. 1.]
```

Figure: Out.png

Constructive Dilemma: Proof

LNN-ATP > RESEARCH > complete > LNN_INFO.log

```
INFO:root:********************** ConstructiveDilemma 2024-08-09 11:38:20.415140 **********************
INFO:root:----------------------
INFO:root:REASONING STEP:0
INFO:root:↓ BOUNDS UPDATED TIGHTENED:1.0 FOR:'(P → Q)' FROM:'((P → Q) ∧ (R → S))' FORMULA:1 PARENT:0
INFO:root:↓ BOUNDS UPDATED TIGHTENED:1.0 FOR:'(R → S)' FROM:'((P → Q) ∧ (R → S))' FORMULA:4 PARENT:0
INFO:root:DOWNWARD INFERENCE RESULT:2.0
INFO:root:----------------------
INFO:root:REASONING STEP:1
INFO:root:NO UPDATES AVAILABLE, TRYING A NEW AXIOM
INFO:root:======================
INFO:root:INFERENCE CONVERGED WITH 2.0 BOUNDS UPDATES IN 2 REASONING STEPS
INFO:root:******************************************************************************
INFO:root:********************** ConstructiveDilemma With Contradiction 2024-08-09 11:38:20.446316 **********************
INFO:root:----------------------
INFO:root:REASONING STEP:0
INFO:root:↓ BOUNDS UPDATED TIGHTENED:1.0 FOR:'Q' FROM:'(Q ∨ S)' FORMULA:3 PARENT:2
INFO:root:↓ BOUNDS UPDATED TIGHTENED:1.0 FOR:'S' FROM:'(Q ∨ S)' FORMULA:6 PARENT:2
INFO:root:↓ BOUNDS UPDATED TIGHTENED:1.0 FOR:'(P → Q)' FROM:'((P → Q) ∧ (R → S))' FORMULA:1 PARENT:0
INFO:root:↓ BOUNDS UPDATED TIGHTENED:1.0 FOR:'(R → S)' FROM:'((P → Q) ∧ (R → S))' FORMULA:4 PARENT:0
INFO:root:↓ BOUNDS UPDATED TIGHTENED:1.0 FOR:'R' FROM:'(R → S)' FORMULA:5 PARENT:4
INFO:root:↓ BOUNDS UPDATED TIGHTENED:1.0 FOR:'P' FROM:'(P → Q)' FORMULA:2 PARENT:1
INFO:root:DOWNWARD INFERENCE RESULT:6.0
INFO:root:----------------------
INFO:root:REASONING STEP:1
INFO:root:↑ BOUNDS UPDATED TIGHTENED:1.0 FOR:'(P ∨ R)' FORMULA:1
INFO:root:↑ CONTRADICTION FOR:'(P ∨ R)' FORMULA:1
INFO:root:UPWARD INFERENCE RESULT:1.0
INFO:root:----------------------
INFO:root:REASONING STEP:2
INFO:root:NO UPDATES AVAILABLE, TRYING A NEW AXIOM
INFO:root:======================
INFO:root:INFERENCE CONVERGED WITH 7.0 BOUNDS UPDATES IN 3 REASONING STEPS
INFO:root:******************************************************************************
```

Figure: IBM's Proof Trace

Constructive Dilemma: Proof

LNN-ATP > RESEARCH > complete > INFO > ConstructiveDilemma_INFO > Proof.txt

```
Model:
    ConstructiveDilemma

Premises:
    ((P → Q) ∧ (R → S)) : True
    (P ∨ R) : True

Query:
    (Q ∨ S) : Fact.UNKNOWN

Derivation Solution:
    [ (P → Q) : True ]     Conjunctive Elimination     from [ ((P → Q) ∧ (R → S)) : True ]
    [ (R → S) : True ]     Conjunctive Elimination     from [ ((P → Q) ∧ (R → S)) : True ]

    * The model was unable to converge on a solution during direct proof. Attempting PROOF BY CONTRADICTION...

    [ (Q ∨ S) : FALSE ]                  Proof By Contradiction
    [ Q : False ]                        De Morgan's Law            from [ (Q ∨ S) : False ]
    [ S : False ]                        De Morgan's Law            from [ (Q ∨ S) : False ]
    [ (P → Q) : True ]                   Conjunctive Elimination    from [ ((P → Q) ∧ (R → S)) : True ]
    [ (R → S) : True ]                   Conjunctive Elimination    from [ ((P → Q) ∧ (R → S)) : True ]
    [ R : False ]                        Modus Tollens              from [ (R → S) : True ]
    [ P : False ]                        Modus Tollens              from [ (P → Q) : True ]
    [ (P ∨ R) : Fact.CONTRADICTION ]     Upward Pass

    * The model has found a CONTRADICTION.

    QUERY [ (Q ∨ S) ] is TRUE due to Proof by Contradiction

    QED.
```

Figure: Our Proof Trace

Constructive Dilemma Graph

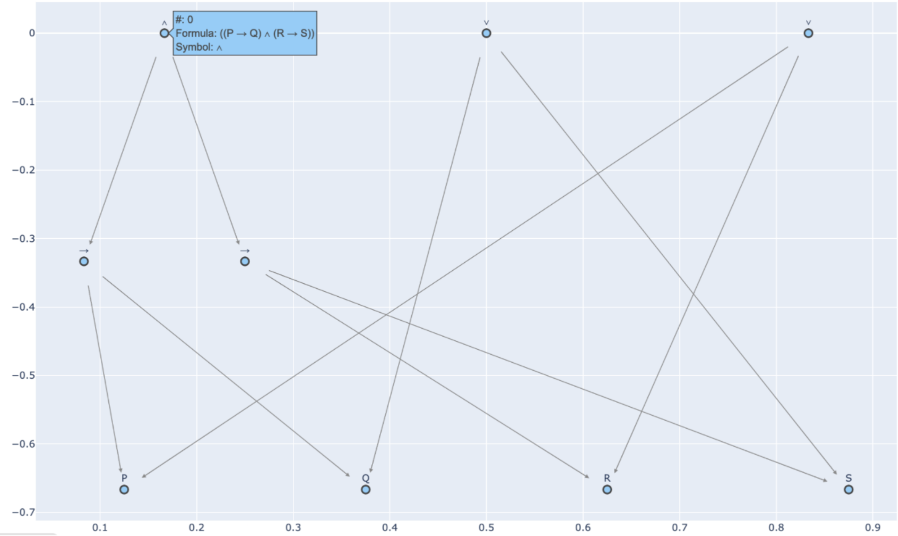


Figure: Graph

Challenges and Future Work

Overview

While the LNN is a significant advancement towards resolving the Black Box issue, we encountered several inconsistencies with IBM's current model. These areas present opportunities for further development and enhancement.

- **Incorporating Laws of Logical Equivalence:** Enhancing the model by integrating additional logical laws to ensure more robust reasoning capabilities.
- **Variable Grounding in Predicates:** Addressing the complexities of variable grounding, particularly how it interacts with the `ForAll()` and `Exists()` operators.
- **Handling Contradictory Statements:** Developing more effective learning methods to manage and resolve contradictions within the system.

References

IBM. (n.d.). *Introduction to Logic and Reasoning* [Video]. IBM.
IBM. (n.d.). *LNN Theory Introduction* [Video]. IBM.
IBM. (n.d.). *LNN Practical Introduction* [Video]. IBM.
Arizona State University. (n.d.). *Logical Neural Networks - Overview* [Video]. Arizona State University.
Arizona State University. (n.d.). *Logical Neural Networks Pt 1/4: Central Ideas* [Video]. Arizona State University.
Arizona State University. (n.d.). *Logical Neural Networks Pt 2/4: Logic and Inference* [Video]. Arizona State University.
Arizona State University. (n.d.). *Logical Neural Networks Pt 3/4: Training* [Video]. Arizona State University.
Arizona State University. (n.d.). *Logical Neural Networks Pt 4/4: LNN in the larger context of NSR* [Video]. Arizona State University.
Skirmilitor. (2020, October 6). *Logical Neural Networks: Seamless Neurosymbolic AI*. Medium. Retrieved from https://skirmilitor.medium.com/logical-neural-networks-31498d1aa9be
IBM Research. (2021, April 21). *Neuro-Symbolic Inductive Logic Programming with Logical Neural Networks*. IBM Research. Retrieved from https://research.ibm.com/publications/neuro-symbolic-inductive-logic-programming-with-logical-neural-netwo
Schulz, S., Hahn, U. (2000). *PyRes: A Python-based Theorem Prover*. Retrieved from https://github.com/eprover/PyRes
Pfenning, F. (2004). *Automated Theorem Proving*. Carnegie Mellon University. Retrieved from https://www.cs.cmu.edu/fp/courses/atp/handouts/atp.pdf