

COS30019 Introduction to AI

Assignment 2: Inference Engine

Student Names:

- Hasindu Sathsara Vithanage, ID: **104824660**
- Mallika Arachchillage Nidula Sanketh Mallikarachchi, ID: **104756611**

Group Name: **COS30019_A02_T112**

Table of Contents

Instructions	3
Introduction.....	3
Inference Methods.....	3
Implementation.....	3
Pseudocodes	4
Truth Table Algorithm	4
Forward Chaining Algorithm	5
Backward Chaining Algorithm.....	6
Testing	7
Basic Cases:.....	7
Complex cases:	8
Edge Cases:	10
Features/Bugs/Missing	11
Research.....	11
Team Summary Report	12
Conclusion.....	12
Acknowledgements/Resources.....	12
References	13

Instructions

- The python script can be executed in the CLI by running:
C:\Assignments\path_to_folder> python iengine.py <test_filename> <method>
- For example:
C:\Users\hasin\Desktop\Intro to AI\Assignment_2\AI_FINAL_NH\Trial_4>python iengine.py test1.txt FC

Introduction

In this report, we document our implementation of an inference engine for propositional logic using three methods:

- Truth Table (TT)
- Forward Chaining (FC)
- Backward Chaining (BC)

Our inference engine is made to process a Horn-Form Knowledge Base (KB) and to determine if the given query can be entailed by the KB or not. This project covers basic ideas in logic including truth tables, logical connectives, and sentence models.

Inference Methods

Truth Table (TT) Method: This method works by evaluating all possible models of the KB to check if the query can be derived from the Horn-Form KB. It systematically checks all possible truth assignments to get to an answer.

Forward Chaining (FC) Method: This method works for Horn-Form KBs, by applying the known rules to the known facts iteratively until it can't derive any more new facts. If the query is found, it will stop this checking and will return the output with a list of propositional symbols entailed from KB that has been found during the execution.

Backward Chaining (BC) Method: This method starts with the query first and then works its way backwards, finding supporting facts and rules until all the known facts are reached. This approach is goal driven.

Implementation

In our Python script for this implementation, there are several modules:

- **FileReader.py:** This script is used to read the KB and query from the test file and will separate 'TELL' which is the KB and 'ASK' which is the query.
- **KnowledgeBase.py:** This script manages all the facts and rules, the adding clauses and retrieves them for inference methods.
- **TT.py:** This script implements the TT method, which generates all possible models and checks if the KB entails the query.
- **FC.py:** This script implements the FC method, which derives facts until the query is entailed.
- **BC.py:** This script implements the BC method, which recursively checks if the query can be derived from the KB.

- **iengine.py**: This is the main entry point for the inference engine. It parses command-line arguments and calls the appropriate inference method.

Pseudocodes

Truth Table Algorithm

```
# Initialize
def initialize(kb, query):
    symbols = extract_symbols(kb) | extract_symbols(query) # Union of symbols in KB and query
    models_count = 0
    check_all(symbols, model {}, kb, query)
    return models_count

# Extract all unique symbols from KB and query
def extract_symbols(expressions):
    symbols = set()
    for expr in expressions:
        symbols.update(extract_symbols_from_expression(expr))
    return symbols

# Recursively assign truth values to each symbol and check model validity
def check_all(symbols, model, kb, query):
    if not symbols:
        # All symbols are assigned, check if KB is satisfied
        if is_satisfied(kb, model):
            # KB is satisfied, check if the query is also satisfied
            if is_satisfied(query, model):
                models_count += 1
                return True
            return False
        return True # If KB is not satisfied, this branch is irrelevant
    else:
        # Still have symbols to assign, pop a symbol and create two branches
        remaining_symbols = symbols.copy()
        symbol = remaining_symbols.pop()

        # Create model where the symbol is True
        model_true = model.copy()
        model_true[symbol] = True

        # Create model where the symbol is False
        model_false = model.copy()
        model_false[symbol] = False

        # Recursive call for both models
        return check_all(remaining_symbols, model_true, kb, query) or check_all(remaining_symbols, model_false, kb, query)

# Check if a KB or query is satisfied under the current model
def is_satisfied(expressions, model):
    for expr in expressions:
        if not evaluate(expr, model):
            return False
    return True

# Evaluate a logical expression based on the current model
def evaluate(expression, model):
    # (and, or, not, =>, etc.)
    pass

# Example Usage
kb = ["A => B", "B => C"]
query = "C"
models_count = initialize(kb, query)
print("Models satisfying both KB and Query:", models_count)
```

Figure 1: Pseudo Code for Truth Table

Forward Chaining Algorithm

```

# Initialize
def initialize():
    agenda = [] # List to manage symbols that need to be processed
    inferred = {} # Dictionary to track if each symbol has been inferred as True
    count = {} # Dictionary to track how many premises must still be satisfied for each clause

# Parse Knowledge Base
def parse_KB(KB):
    for clause in KB:
        if "=>" in clause:
            # If the clause is an implication (A & B => C)
            premise, conclusion = clause.split("=>")
            premises = premise.split("&")
            # Set the count for the number of premises
            count[clause] = len(premises)
            # Ensure the conclusion is in the inferred dictionary set to False
            inferred[conclusion.strip()] = False
        else:
            # If the clause is a fact, add it to the agenda
            agenda.append(clause.strip())
            inferred[clause.strip()] = False

# Process Agenda
def process_agenda(query):
    inferred_order = [] # List to keep track of the order in which symbols are inferred

    while agenda:
        agenda.sort() # Sort agenda to maintain a consistent processing order
        p = agenda.pop(0) # Pop first item from agenda

        if p == query:
            return True, inferred_order

        if not inferred[p]:
            # Set p as inferred
            inferred[p] = True
            inferred_order.append(p)

            # Go over each clause in KB
            for clause in KB:
                if "=>" in clause:
                    premise, conclusion = clause.split("=>")
                    premises = premise.split("&")

                    # If p is part of the premises of this implication
                    if p.strip() in [premise.strip() for premise in premises]:
                        count[clause] -= 1
                        # If all premises are now True, add conclusion to agenda
                        if count[clause] == 0:
                            agenda.append(conclusion.strip())

    # If agenda is empty and query was not inferred
    return False, inferred_order

# Example execution
KB = ["A & B => C", "A"]
query = "C"
initialize()
parse_KB(KB)
result, order_of_inference = process_agenda(query)
print("Result:", result)
print("Inferred Order:", order_of_inference)

```

Figure 2: Pseudo Code for Forward Chaining

Backward Chaining Algorithm

```

# Initialize
def initialize():
    agenda = []          # List to manage goals to be processed
    inferred = {}        # Dictionary to track if each symbol has been inferred as True
    proven_list = []     # List to keep track of proven goals

# Add Query to Agenda
def add_query_to_agenda(query):
    agenda.append(query)
    # Initialize all symbols as False in the inferred dictionary
    for symbol in extract_symbols(KB):
        inferred[symbol] = False

# Backward Chaining OR (Attempt to prove a single goal using knowledge base)
def bc_or(goal):
    if goal in proven_list:
        return True

    for clause in KB:
        if "=>" in clause:
            premise, conclusion = clause.split("=>")
            if goal == conclusion.strip():
                # Attempt to prove all premises using bc_and
                if bc_and(premise.strip()):
                    proven_list.append(goal)
                    return True
        else:
            if goal == clause.strip():
                proven_list.append(goal)
                return True

    return False

# Backward Chaining AND (Attempt to prove all conditions of a premise)
def bc_and(premise):
    conditions = premise.split("&")
    for condition in conditions:
        condition = condition.strip()
        if not inferred[condition]:
            result = bc_or(condition)
            if not result:
                return False
    return True

# Process Agenda
def process_agenda():
    while agenda:
        goal = agenda.pop() # Pop last goal from the agenda

        if not inferred[goal]:
            inferred[goal] = True
            if not bc_or(goal):
                return False, proven_list

    return True, proven_list

# Example Execution
KB = ["A & B => C", "A"]
query = "C"
initialize()
add_query_to_agenda(query)
result, order_of_inference = process_agenda()
print("Result:", result)
print("Inferred Order:", order_of_inference)

```

Figure 3: Pseudo Code for Backward Chaining

Testing

Basic Cases:

Test Case	KB	Output
Y1	TELL p2 => p3; p3 => p1; c => e; b & e => f; f & g => h; p2 & p1 & p3 => d; p1 & p3 => c; a; b; p2; ASK d	PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/Y1.txt TT YES PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/Y1.txt FC YES: a, b, c, p1, p2, p3 PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/Y1.txt BC YES: p2, p3, p1, d
Y2	TELL a => b; b => c; c => d; d => e; e => f; f & g => h; a; g; ASK h	PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/Y2.txt TT YES PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/Y2.txt FC YES: a, b, c, d, e, f, g PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/Y2.txt BC YES: a, b, c, d, e, f, g, h
Y3	TELL x => y; y => z; z & w => q; x; w; ASK q	PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/Y3.txt TT YES PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/Y3.txt FC YES: w, x, y, z PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/Y3.txt BC YES: x, y, z, w, q
Y4	TELL r => s; s => t; t => u; u & v => w; r; v; ASK w	PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/Y4.txt TT YES PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/Y4.txt FC YES: r, s, t, u, v PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/Y4.txt BC YES: r, s, t, u, v, w
Y5	TELL m => n; n => o; o => p; p & q => r; m; q; ASK r	PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/Y5.txt TT YES PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/Y5.txt FC YES: m, n, o, p, q PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/Y5.txt BC YES: m, n, o, p, q, r
N1	TELL p1 => p2; p2 => p3; a; b; ASK p4	PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/N1.txt TT NO PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/N1.txt FC NO PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/N1.txt BC NO
N2	TELL a => b; b => c; c => d; e; f; ASK g	PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/N2.txt TT NO PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/N2.txt FC NO PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/N2.txt BC NO

COS30019-Introduction to Artificial Intelligence

N3	TELL x => y; y => z; w; ASK z	PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/N3.txt TT NO PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/N3.txt FC NO PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/N3.txt BC NO
N4	TELL r => s; t => u; v; ASK w	PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/N4.txt TT NO PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/N4.txt FC NO PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/N4.txt BC NO
N5	TELL m => n; o => p; q; ASK r	PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/N5.txt TT NO PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/N5.txt FC NO PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Test_Files/N5.txt BC NO

Complex cases:

Test Case	KB	Output
Y1	TELL a => b; b & c => d; d => e; a; c; f => g; g & h => i; j => k; l & m => n; n => o; p & q => r; s => t; u & v => w; x & y => z; ASK e	~\Desktop\Intro to AI\Assignment_2\AI_FINAL_NH\Trial_4 git:[Hasi_Branch] python iengine.py .\Complex_Test_Files\Y1.txt fc YES: a, b, c, d ~\Desktop\Intro to AI\Assignment_2\AI_FINAL_NH\Trial_4 git:[Hasi_Branch] python iengine.py .\Complex_Test_Files\Y1.txt bc YES: a, b, c, d, e TT => NULL
Y2	TELL p1 => p2; p2 => p3; p3 => p4; p4 & p5 => p6; p6 & p7 => p8; p8 => p9; p1; p5; p7; ASK p9	~\Desktop\Intro to AI\Assignment_2\AI_FINAL_NH\Trial_4 git:[Hasi_Branch] python iengine.py .\Complex_Test_Files\Y2.txt fc YES: p1, p2, p3, p4, p5, p6, p7, p8 ~\Desktop\Intro to AI\Assignment_2\AI_FINAL_NH\Trial_4 git:[Hasi_Branch] python iengine.py .\Complex_Test_Files\Y2.txt bc YES: p1, p2, p3, p4, p5, p6, p7, p8, p9 PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Complex_Test_Files\Y2.txt TT YES
Y3	TELL a => b; b & d => e; c => d; e => f; f & g => h; i => j; j & k => l; l => m; m & n => o; o & p => q; a; c; g; i; k; n; p; ASK q	~\Desktop\Intro to AI\Assignment_2\AI_FINAL_NH\Trial_4 git:[Hasi_Branch] python iengine.py .\Complex_Test_Files\Y3.txt bc YES: i, j, k, l, m, n, o, p, q ~\Desktop\Intro to AI\Assignment_2\AI_FINAL_NH\Trial_4 git:[Hasi_Branch] python iengine.py .\Complex_Test_Files\Y3.txt fc YES: a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Complex_Test_Files\Y3.txt TT YES

COS30019-Introduction to Artificial Intelligence

Y4	<pre> TELL p & q => r; r => s; s & t => u; u => v; a => b; b & c => d; d & e => f; f => g; h & i => j; j => k; k & l => m; m => n; p; q; t; a; c; e; h; i; l; ASK n </pre>	<pre> ~\Desktop\Intro to AI\Assignment_2\AI_FINAL_NH\Trial_4 git:[Hasi_Branch] python iengine.py .\Complex_Test_Files\Y4.txt fc YES: a, b, c, d, e, f, g, h, i, j, k, l, m ~\Desktop\Intro to AI\Assignment_2\AI_FINAL_NH\Trial_4 git:[Hasi_Branch] python iengine.py .\Complex_Test_Files\Y4.txt bc YES: h, i, j, k, l, m, n PS C:\Users\nidul\Desktop\Trial_4> python iengine.py Complex_Test_Files\Y4.txt TT YES </pre>
Y5	<pre> TELL a & b => c; c & d => e; e => f; f & g => h; h & i => j; j & k => l; l & m => n; n => o; p & q => r; r & s => t; t => u; v & w => x; x => y; y & z => aa; aa => ab; a; b; d; g; i; k; m; p; q; s; v; w; z; ASK ab </pre>	<pre> ~\Desktop\Intro to AI\Assignment_2\AI_FINAL_NH\Trial_4 git:[Hasi_Branch] python iengine.py .\Complex_Test_Files\Y5.txt fc YES: a, aa, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z ~\Desktop\Intro to AI\Assignment_2\AI_FINAL_NH\Trial_4 git:[Hasi_Branch] python iengine.py .\Complex_Test_Files\Y5.txt bc YES: v, w, x, y, z, aa, ab TT => NULL </pre>
N1	<pre> TELL a => b; b & c => d; e => f; f & g => h; i & j => k; l & m => n; n => o; p & q => r; r => s; t & u => v; v & w => x; y => z; z & aa => ab; ac => ad; ae & af => ag; ah & ai => aj; a; c; e; g; i; k; m; p; q; t; w; y; ac; ae; ASK aj </pre>	<pre> ~\Desktop\Intro to AI\Assignment_2\AI_FINAL_NH\Trial_4 git:[Hasi_Branch] python iengine.py .\Complex_Test_Files\N1.txt fc NO ~\Desktop\Intro to AI\Assignment_2\AI_FINAL_NH\Trial_4 git:[Hasi_Branch] python iengine.py .\Complex_Test_Files\N1.txt bc NO TT => NULL </pre>
N2	<pre> TELL p1 => p2; p3 => p4; p5 & p6 => p7; p8 => p9; p10 & p11 => p12; p13 => p14; p15 & p16 => p17; p18 => p19; p20 & p21 => p22; p23 => p24; p1; p3; p5; p6; p8; p10; p13; p15; p16; p18; p20; p21; p23; ASK p25 </pre>	<pre> ~\Desktop\Intro to AI\Assignment_2\AI_FINAL_NH\Trial_4 git:[Hasi_Branch] python iengine.py .\Complex_Test_Files\N2.txt fc NO ~\Desktop\Intro to AI\Assignment_2\AI_FINAL_NH\Trial_4 git:[Hasi_Branch] python iengine.py .\Complex_Test_Files\N2.txt bc NO TT => NULL </pre>
N3	<pre> TELL a1 & a2 => b1; b1 & b2 => c1; c1 & c2 => d1; d1 & d2 => e1; e1 & e2 => f1; f1 & f2 => g1; g1 & g2 => h1; h1 & h2 => i1; i1 & i2 => j1; j1 & j2 => k1; k1 & k2 => l1; l1 </pre>	<pre> ~\Desktop\Intro to AI\Assignment_2\AI_FINAL_NH\Trial_4 git:[Hasi_Branch] python iengine.py .\Complex_Test_Files\N3.txt fc NO ~\Desktop\Intro to AI\Assignment_2\AI_FINAL_NH\Trial_4 git:[Hasi_Branch] python iengine.py .\Complex_Test_Files\N3.txt bc NO TT => NULL </pre>

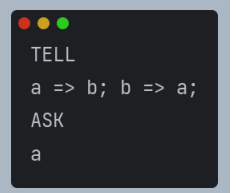
```

& l2 => m1; m1 & m2 =>
n1; n1 & n2 => o1; o1 &
o2 => p1; p1 & p2 => q1;
q1 & q2 => r1; r1 & r2
=> s1; s1 & s2 => t1; t1
& t2 => u1; a1; a2; b2;
c2; d2; e2; f2; g2; h2;
i2; j2; k2; l2; m2; n2;
o2; p2; q2; r2; s2;
ASK
u1

```

Edge Cases:

Test Case	KB	Output
Minimal Knowledge Base with Contradiction	 <pre> TELL a; ~a; ASK b </pre>	<pre> PS C:\Users\nidul\Desktop\Trial_4> python iengine.py edge_test/E1.txt TT NO PS C:\Users\nidul\Desktop\Trial_4> python iengine.py edge_test/E1.txt FC NO PS C:\Users\nidul\Desktop\Trial_4> python iengine.py edge_test/E1.txt BC NO </pre>
Empty Knowledge Base	 <pre> TELL ; ASK a </pre>	<pre> PS C:\Users\nidul\Desktop\Trial_4> python iengine.py edge_test/E2.txt TT NO PS C:\Users\nidul\Desktop\Trial_4> python iengine.py edge_test/E2.txt FC NO PS C:\Users\nidul\Desktop\Trial_4> python iengine.py edge_test/E2.txt BC NO </pre>
Irrelevant Facts	 <pre> TELL a; b; c; ASK z </pre>	<pre> PS C:\Users\nidul\Desktop\Trial_4> python iengine.py edge_test/E3.txt TT NO PS C:\Users\nidul\Desktop\Trial_4> python iengine.py edge_test/E3.txt FC NO PS C:\Users\nidul\Desktop\Trial_4> python iengine.py edge_test/E3.txt BC NO </pre>
Self-Referencing Rule	 <pre> TELL a => a; ASK a </pre>	<pre> PS C:\Users\nidul\Desktop\Trial_4> python iengine.py edge_test/E4.txt TT NO PS C:\Users\nidul\Desktop\Trial_4> python iengine.py edge_test/E4.txt FC NO PS C:\Users\nidul\Desktop\Trial_4> python iengine.py edge_test/E4.txt BC YES: a </pre>

Circular Rules	 <pre> TELL a => b; b => a; ASK a </pre>	<pre> PS C:\Users\nidul\Desktop\Trial_4> python iengine.py edge_test/E5.txt TT NO PS C:\Users\nidul\Desktop\Trial_4> python iengine.py edge_test/E5.txt FC NO PS C:\Users\nidul\Desktop\Trial_4> python iengine.py edge_test/E5.txt BC YES: b, a </pre>
----------------	---	--

Features/Bugs/Missing

Features:

- Truth Table implemented.
- Forward Chaining method implemented.
- Backward Chaining method implemented.
- Reading input from a file
- Can run method in CLI regardless of whether method type is given in uppercase or not (for e.g.: can run 'tt' instead of 'TT')

Bugs:

There are no known bugs found at the time of submission.

Missing:

- In our Truth Table, it is not able to run overly complex Knowledge Bases as running the Truth table is computationally expensive.
- Our Truth Table is missing the number of modules to be displayed along with the result.
- Optimization for large knowledge bases is missing.

Research

Extensive Testing:

To validate our algorithms and to ensure how robust our inference engine is, we conducted an extensive number of tests across various scenarios.

We had three types of test files:

- Basic KB cases: Simple KBs with straightforward queries.
- Complex KB cases: Larger KBs consisting of multiple interdependent rules and propositions.
- Edge Cases: Knowledge Bases with little to no information and KBs with contradictory information.

All these test cases confirmed the effectiveness of each algorithm, and we were able to find missing functionalities, errors and bugs in each algorithm.

Team Summary Report

Nidula's Contributions (50%):

- He implemented the Forward Chaining method.
- Conducted thorough testing and wrote test cases.
- Drafted sections of the report.

Hasindu's Contributions (50%):

- He implemented the Backwards Chaining method.
- He developed the command-line interface.
- Compiled final report and conducted additional research.

Both contributed together:

- Both of us worked on the implementation of the Truth Table method.
- Both of us worked on the facts, rules, and the adding clauses for the knowledge base.

Conclusion

From all the three inference methods, we would say that the Forward Chaining method proved to be the most efficient for large Horn-Form KBs. The Truth Table method, although it is more versatile, was computationally expensive for larger KBs and was unable to find a conclusion. Future improvements could be optimizing the Truth Table so that it is able to handle larger KBs. We should optimize all the chaining methods to handle more complex logical expressions in general.

Acknowledgements/Resources

Education 4u. (2019, September 14). *backward chaining example | Artificial intelligence | Lec-40 | Bhanu Priya*. YouTube. <https://www.youtube.com/watch?v=6DU42so8k48>

- The above youtube tutorial was used to gain a better basic idea of how the backward chaining method could be used in the inference engine.

Education 4u. (2019b, September 14). *forward chaining example | Artificial intelligence | Lec-38 | Bhanu Priya*. YouTube. <https://www.youtube.com/watch?v=RYIjxRfOPWY>

- The above youtube tutorial was used to gain a better basic idea of how the forward chaining method could be used in the inference engine.

GeeksforGeeks. (2020, July 16). *Difference between Backward and Forward Chaining*.

GeeksforGeeks. <https://www.geeksforgeeks.org/difference-between-backward-and-forward-chaining/>

- The above site helped us get an idea on forward chaining and backward chaining.

Ap-Atul. (n.d.). *Expert-System/engine/inference.py at master · AP-Atul/Expert-System*. GitHub.

<https://github.com/AP-Atul/Expert-System/blob/master/engine/inference.py>

MaxWong. (n.d.). *aima-python/logic.ipynb at master · MaxWong03/aima-python*. GitHub.

<https://github.com/MaxWong03/aima-python/blob/master/logic.ipynb>

- We used these python scripts as references when developing our own codes, we tried to see how their logic worked and compared it to our own logic to troubleshoot our own scripts.

References

- Swinburne University of Technology. (2024). COS30019 Introduction to AI Lecture Notes.
- Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach, Global Edition*. Pearson Higher Ed.
[http://books.google.ie/books?id=cb0qEAAQBAJ&dq=Russell,+S.+J.,+%26+Norvig,+P.,+\(2010\).+Artificial+Intelligence:+A+Modern+Approach+\(3rd+ed.\).&hl=&cd=3&source=gbs_api](http://books.google.ie/books?id=cb0qEAAQBAJ&dq=Russell,+S.+J.,+%26+Norvig,+P.,+(2010).+Artificial+Intelligence:+A+Modern+Approach+(3rd+ed.).&hl=&cd=3&source=gbs_api)
- Cai, Z., Liu, L., Chen, B., & Wang, Y. (2021). *Artificial Intelligence: From Beginning To Date*. World Scientific.
[http://books.google.ie/books?id=x30xEAAQBAJ&pg=PA136&dq=Nilsson,+N.+J.,+\(1998\).+Artificial+Intelligence:+A+New+Synthesis.+Morgan+Kaufmann.&hl=&cd=3&source=gbs_api](http://books.google.ie/books?id=x30xEAAQBAJ&pg=PA136&dq=Nilsson,+N.+J.,+(1998).+Artificial+Intelligence:+A+New+Synthesis.+Morgan+Kaufmann.&hl=&cd=3&source=gbs_api)
- Education 4u. (2019b, September 14). *backward chaining example | Artificial intelligence | Lec-40 | Bhanu Priya*. YouTube. <https://www.youtube.com/watch?v=6DU42so8k48>

- Education 4u. (2019b, September 14). *forward chaining example* | Artificial intelligence | Lec-38 | Bhanu Priya. YouTube. <https://www.youtube.com/watch?v=RYIjxRfOPWY>
- GeeksforGeeks. (2020, July 16). *Difference between Backward and Forward Chaining*. GeeksforGeeks. <https://www.geeksforgeeks.org/difference-between-backward-and-forward-chaining/>
- Ap-Atul. (n.d.). *Expert-System/engine/inference.py at master · AP-Atul/Expert-System*. GitHub. <https://github.com/AP-Atul/Expert-System/blob/master/engine/inference.py>
- MaxWong. (n.d.). *aima-python/logic.ipynb at master · MaxWong03/aima-python*. GitHub. <https://github.com/MaxWong03/aima-python/blob/master/logic.ipynb>