COS30019-INTRODUCTION TO ARTIFICIAL INTELIGENCE

Assignment 2:Inference Engine

Nguyen Hoang Minh

104972886

Le Hoang Minh

104656973

CONTENTS

1. INTRODUCTION

1.1 BACKGROUND

This report outlines the approach and techniques used to fulfill the requirements of Assignment 2, where the task was to implement an inference engine for propositional logic in Python. The goal of this assignment is to demonstrate how an inference engine applies logical rules to a knowledge base (KB) to deduce information or make decisions.

In this assignment, the provided knowledge base (KB) and query (q) are given in a specific format. The KB, which follows the keyword **TELL**, consists of Horn clauses that are separated by semicolons, while the query (q), introduced by the keyword **ASK**, contains a proposition symbol that we evaluate for entailment within the KB.

1.2 INSTRUCTION

To run test file, use the terminal and follow this structure: python [code_filename] [test_filename] [method_name]

Example: python iengine.py test_HornKB.txt bc

The iegine.py is the name of our code file, test_HornKB is the example test filename, there will be 20 others test files in our folder and bc stand for backward chaining, there are 3 remain methods that you can use to run test, Truth Table(tt), Forward Chaining, DPLL.

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

Example output:

```
test_HornKB.txt bc
This is the KB: ['p2=> p3', 'p3 => p1', 'c => e', 'b&e => f', 'f&g => h', 'p2&p1&p3=>d', 'p1&p3 => c', 'a', 'b', 'p2']
This is the QUERY: d
YES: p2, p3, p1, d
```

The output:

- YES, followed by a colon (:) and the number of models of KB for TT method if the test case is valid, the list of propositional symbols entailed if the method are FC or BC
- It will return the KB and query for DPLL if the query is entailed
- No for invalid input

Figure 1: Example of an expected Horn statement.

The goal of the program is to evaluate whether a query (q), such as the proposition symbol "d," can be entailed from the provided KB using different inference methods. The inference engine is equipped with **Truth Table (TT) Checking**, **Backward Chaining (BC)**, and **Forward Chaining (FC)** algorithms.

2. INFERENCE METHODS

The inference engine developed for this assignment utilizes several inference methods to logically determine whether the query (q) is entailed by the KB. Each method works differently, offering unique advantages and limitations.

2.1 TRUTH TABLE CHECKING

Truth Table Checking is a brute-force method that involves enumerating all possible truth assignments for the propositional symbols in the KB and the query (q). The truth table contains all possible combinations of truth values (True or False) for these symbols, with each row representing a different model—a specific combination of truth values.

For every model, the KB and the query are evaluated. If both are true under a given model, the query is considered entailed by the KB. Truth table checking guarantees that all possible scenarios are considered, making it the most comprehensive method for determining entailment. However, its major drawback lies in its exponential complexity. As the number of propositional symbols increases, the size of the truth table grows exponentially, making it impractical for larger KBs or queries.

2.1.1 HOW IT WORKS

The truth_table function generates all combinations of truth values for the variables in the knowledge base. It does this by iterating through all possibilities using Python's itertools.product. For each combination, it checks if the KB holds true, and if it does, whether the query is also true. If the query holds true in all valid cases, it is entailed.

This method is guaranteed to be correct, but its main drawback is efficiency. The time complexity grows exponentially with the number of variables, making it impractical for larger knowledge bases.

2.1.2 EXAMPLE

The Truth Table method would check all combinations of truth values for A, B, and C to see if C is true whenever the KB is true.

2.1.3 ADVANTAGES:

1. The truth table provides clean representation which makes it easier to analyze.
2. It provides a systematic method for analyzing all possible combinations of truth values of a particular logical expression.
3. It is very easy to implement.

2.1.4 DISADVANTAGES:

1. The number of truth assignments grows exponentially with the number of symbols
2. For large KBs with many symbols, this makes the method computationally expensive and impractical.

2.2 FORWARD CHAINING

Forward Chaining is an inference method that begins by analyzing the facts within the KB. It then iteratively applies the rules in the KB to derive new facts, continuously updating the KB with new information until the query (q) is found or no further inferences can be made.

This method excels in scenarios involving Horn clauses, where each rule has a single positive literal. It is particularly effective in linear time reasoning, as it processes rules sequentially based on facts already known to be true. Forward Chaining is best suited for environments where the facts and rules can be applied in a straightforward manner. However, it struggles with more complex scenarios where rules are highly interconnected or involve more general logic, which may not align well with the Horn-clause structure.

2.2.1 HOW IT WORKS

The FC class implements Forward Chaining. It begins by adding all the known facts to a queue. Then, it iteratively checks each rule in the KB. If all the premises of a rule are satisfied (i.e., they are known to be true), the rule's conclusion is added to the queue as a new fact. This process continues until the query is deduced or no new facts can be generated.

2.2.2 EXAMPLE

Forward Chaining starts with the known fact A, then uses the rule A => B to infer B. Next, it uses the rule B => C to infer C, thus proving the query C.

2.2.3Advantages

1. Forward chaining is straightforward and easy to implement.
2. It processes data as it arrives, making it suitable for dynamic environments where new data continuously becomes available.

2.2.4Disadvantages

1. Forward Chaining generates all possible facts from the KB, even if many of them are not relevant to the query

2.3 BACKWARD CHAINING

Backward Chaining operates in the opposite direction of Forward Chaining. It starts with the query (q) and works backward, attempting to find facts or rules in the KB that lead to the conclusion of the query. If a premise needed to prove the query is not already a known fact, it becomes a sub-goal. The backward chain then recursively searches for facts that could support this sub-goal until it either finds a valid logical path or fails to find a solution.

Backward Chaining is especially effective when the query is complex but does not require examining every possible rule or fact in the KB. It is well-suited for cases where the KB may have many facts, but only a subset of them are relevant to the query. However, like Forward Chaining, it is less effective in environments where generalized logic or interconnected rule sets need to be evaluated.

2.3.1 HOW IT WORKS

The BC class is responsible for backward chaining. It takes the query and checks whether it can be proven by finding rules in the KB that lead to the query. If the premises of a rule aren't already known, it recursively tries to prove those premises, continuing this process until either the query is proven or no further progress can be made.

2.3.2 EXAMPLE

Backward Chaining starts with the query C and attempts to prove it. First, it checks whether B => C can be satisfied. Since it does not know whether B is true, it then tries to prove B from A using the rule A => B.

2.3.3 Advantages

1. Backward Chaining works backward from the query, checking only the rules and facts necessary to determine whether the query is entailed
2. Backward Chaining does not derive all possible facts—it only explores paths relevant to the query.

2.3.4 Disadvantages

1. It requires predefined goals
2. If multiple goals need to be achieved, backward chaining may need to be repeated for each goal

2.4 DPLL (DAVIS-PUTNAM-LOGEMANN-LOVELAND)

DPLL is a more advanced method, typically used for solving the propositional satisfiability problem (SAT). It's a backtracking algorithm that performs unit propagation (simplifying the formula by assigning truth values to certain literals) and pure literal elimination (removing literals that are always true or false). These optimizations help it handle larger and more complex problems.

2.4.1 HOW IT WORKS

The DPLL class implements the DPLL algorithm. It starts by checking if the KB is satisfiable under the current assignment of truth values. If all clauses are satisfied, the formula is satisfiable. If any clause is falsified, the algorithm tries a different assignment. It recursively assigns truth values to literals and simplifies the formula until it finds a satisfying assignment or determines that no solution exists.

2.4.2 EXAMPLE

The DPLL algorithm would try different truth assignments and simplify the formula as it proceeds, ultimately finding a satisfying assignment or concluding that none exists.

3. IMPLEMENTATION

3.1 TEXT FILE ANALYSIS

3.1.1 Purpose:

- This function reads the Knowledge Base and query from a text file. It splits the content into TELL (defining the KB) and ASK (specifying the query).

3.1.2 Implementation details:

- read_file(filename) method:
  - Reads the input file and checks for proper formatting of TELL and ASK sections.
  - Splits the KB (after TELL) into clauses based on semicolons (;) and strips whitespace.
  - Validates the method compatibility for generic KBs:
    - Ensures only TT and DPLL methods can handle connectives like <=>, ||, and ~.
  - Returns the KB and query.

3.2 INFERENCE METHODS

3.2.1 TRUTH TABLE CHECKER (TT CLASS)

- Purpose:
  - Implements the Truth Table Entailment method to determine if the KB entails the query.
- Method implementation details
  - ExtractSymbols(kb):
    - Uses a regex pattern to extract all distinct symbols from the KB.
  - CheckEntails():
    - Calls CreateTruthTable to generate truth assignments and evaluate entailment.
    - Prints YES: <valid_model_count> if the query is entailed; otherwise, prints NO.
  - CreateTruthTable():
    - Generates all possible truth assignments for the symbols in the KB and query using itertools.product.
    - For each assignment:
    - Checks if the KB is true.
    - Checks if the query is true.
    - Displays results in a formatted table using the tabulate library. Evaluates the KB and query for each assignment.
    - Highlights valid models (where KB and query are both true) in green using ANSI escape codes.
    - Displays results in a formatted table using the tabulate library.
  - Check_if_clause_true():
    - Checks if a clause is true under a specific truth assignment

- EvaluateClause(self, clause, model):
    - Implementation:
        - Splits the clause into left, op, and right using FindMainOperator.
    - Handles logical operators:
        - <=> (Biconditional): Ensures left and right evaluate to the same truth value.
        - => (Implication): not left or right.
        - || (Disjunction): left or right.
        - & (Conjunction): left and right.
        - ~ (Negation): not left.
        - Base case: Retrieves the truth value of simple symbols from model.
- FindMainOperator(self, clause):
    - Implementation:
        - Uses a bracket_level counter to track whether the current position is inside parentheses.
        - Scans the clause for operators (<=>, =>, ||, &) at the top level
        - Splits the clause into left, op, and right for further processing.

3.2.2 CHAINING (BASE CLASS) – CHAINING CAN ONLY RUN THE HORN KNOWLEDGE BASE

- Purpose
    - Create base functions for the FC and BC (Forward Chaining and Backward Chaining)
- Method Implementation Details:
    - FindSingleClause():
        - Identifies facts and adds them to a processing queue.
    - GenerateSentenceList():
        - Parses rules into:
            - Premises: The conditions (antecedents).
            - Conclusions: The outcomes (consequents).
        - Tracks the count of unmet premises for each rule.

3.2.3 FORWARD CHAINING (FC)

- Purpose:
    - Implements the Forward Chaining (FC) algorithm
- Method Implementation detail
    - Initializes a queue of known facts and processes them iteratively.
    - For each fact:
        - Reduces the count of unmet premises in applicable rules.
        - If all premises of a rule are satisfied, infer the conclusion and use appends to add it to the queue
    - Stops when the query counts down to 0 and return the output, YES or NO

3.2.4 BACKWARD CHAINING (BC)

- Purpose
    - Implements the Backward Chaining algorithm
- Method Implementation detail
    - CheckEntails

        - Calls FindSingleClause() to extract known facts (standalone clauses).
        - Calls GenerateSentenceList() to parse the KB into rules with premises and conclusions.
        - Uses the recursive helper function TruthValue() to check if the query can be derived.
        - Prints YES with inferred facts if the query is entailed; otherwise, prints NO.
    - TruthValue
        - If the symbol is already in the inferred list, return True
        - If the symbol is in the queue, it is true, add it to the inferred list to avoid re-checking.
        - If the symbol has already been visited during recursion, return False.
        - For loop:
            - If the symbol is the conclusion, the code look for rules in the sentence_list
            - For each rule, it recursively check if all the premises are true using TruthValue().
            - If all premises are true, infer the symbol as true and add it to inferred.

3.2.5 DPLL (DAVIS-PUTNAM-LOGEMANN-LOVELAND)

- Purpose:
    - Implements the Davis-Putnam-Logemann-Loveland (DPLL) algorithm
- Method Implementation Detail:
    - convert_to_clauses(self, kb)
        - Converts the KB from a string representation into a list of clauses
        - Each clause is parsed using the parse_clause method.
    - parse_clause(self, clause)
        - This method splits a clause into a list of individual literals
        - The clause is split by the | (OR) operator
    - dpll(self, clauses, assignment)
        - This is the core recursive method of the DPLL algorithm
        - IF 1: If all clauses are satisfied, return TRUE with the count of valid models
        - IF 2: if any of the clause is falsified, return False
        - Unit clauses:
            - If there are any unit clauses: clauses with exactly one unassigned literal, it assigns the corresponding truth value to that literal and simplifies the formula.
        - Pure_literals
            - If a literal has only one polarity like x, it is assigned the corresponding truth value.
        - Literal:
            - Try to pick one literal and assign value TRUE or FALSE to it
    - simplify_clauses(self, clauses, literal)
        - Simplifies the set of clauses by removing or updating clauses
        - For each clause:
            - If the literal is in the clause, it removes the clause
            - If the literal is not in the clause, remove the negated literal from the clause and put it in the filtered_clause then append it back t the Simplifies

3.3 MAIN CLASS

- Purpose:

- Handles command-line arguments, processes the input file, and selects the appropriate inference method.

- Method Implementation Detail:
  - Reads the file using TextFileAnalyzer.
  - Verifies the KB and query format.
  - Selects the inference method based on command line input and executes it.
  - Prints the result

4. TESTING

4.0.1 HORN TESTING

The system has been tested with Horn clauses, which are a special form of propositional logic often used in forward chaining. Horn clauses ensure that each rule has at most one positive literal, which simplifies the process of inference. Below are 9 different test cases using Horn Knowledge Base.

2	HornKB_1	TELL R&N => W ASK R
3	Truth Table	This is the KB: ['R&N => W'] This is the QUERY: R NO
4	FC	This is the KB: ['R&N => W'] This is the QUERY: R NO
5	BC	This is the KB: ['R&N => W'] This is the QUERY: R NO

HornKB_2	TELL P=>Q;L&M=>P;B&L=>M;A&P=>L;A&B=>L;A;B ASK Q
Truth Table	This is the KB: ['P=>Q', 'L&M=>P', 'B&L=>M', 'A&P=>L', 'A&B=>L', 'A', 'B'] This is the QUERY: Q YES: 1
FC	This is the KB: ['P=>Q', 'L&M=>P', 'B&L=>M', 'A&P=>L', 'A&B=>L', 'A', 'B'] This is the QUERY: Q YES: A, B, L, M, P, Q
BC	This is the KB: ['P=>Q', 'L&M=>P', 'B&L=>M', 'A&P=>L', 'A&B=>L', 'A', 'B'] This is the QUERY: Q YES: A, B, L, M, P, Q

HornKB_3	TELL D=>P; P=>E; E=>K; K; ASK K
Truth Table	This is the KB: ['D=>P', 'P=>E', 'E=>K', 'K'] This is the QUERY: K YES: 4
FC	This is the KB: ['D=>P', 'P=>E', 'E=>K', 'K'] This is the QUERY: K YES: K
BC	This is the KB: ['D=>P', 'P=>E', 'E=>K', 'K'] This is the QUERY: K YES: K

HornKB_4	TELL A & B => C; C => D; E => F; ASK F
Truth Table	This is the KB: ['A & B => C', 'C => D', 'E => F'] This is the QUERY: F NO
FC	NO
BC	NO

HornKB_5	TELL G => H; I & J => K; K => L; ASK L
Truth Table	This is the KB: ['G => H', 'I & J => K', 'K => L'] This is the QUERY: L NO
FC	Same as Truth Table
BC	Same as Truth Table

HornKB_6	TELL M & N => O; O & P => Q; Q => R; ASK R
Truth Table	This is the KB: ['M & N => O', 'O & P => Q', 'Q => R'] This is the QUERY: R NO
FC	Same as Truth Table
BC	Same as Truth Table

HornKB_7	TELL A; ASK A
Truth Table	This is the KB: ['A'] This is the QUERY: A YES: 1
FC	This is the KB: ['A'] This is the QUERY: A YES: A
BC	test_HornKB_7.txt bc This is the KB: ['A'] This is the QUERY: A YES: A

HornKB_8	TELL A; B => C; D => E; ASK A
Truth Table	This is the KB: ['A', 'B => C', 'D => E'] This is the QUERY: A YES: 9
FC	This is the KB: ['A', 'B => C', 'D => E'] This is the QUERY: A YES: A
BC	This is the KB: ['A', 'B => C', 'D => E'] This is the QUERY: A YES: A

HornKB_9	TELL A & B => C; C & D => E; E & F => G; G & H => I; I & J => K; K & L => M; M & N => O; O & P => Q; Q & R => S; S & T => U; U & V => W; W & X => Y; Y & Z => AA; AA & AB => AC; AC & AD => AE; AE & AF => AG; AG & AH => AI; AI & AJ => AK; AK & AL => AM; AM & AN => AO; AO & AP => AQ; AQ & AR => AS; AS & AT => AU; AU & AV => AW; AW & AX => AY; AY & AZ => BA; BA & BB => BC; BC & BD => BE; BE & BF => BG; BG & BH => BI; BI & BJ => BK; BK; C; E; G; I; K; M; O; Q; S; U; W; Y; AA; AC; AE; AG; AI; AK; AM; AO; AQ; AS; AU; AW; AY; BA; BC; BE; BG; BI; ASK BK
Truth Table	Took too long to generate output
FC	YES: BK
BC	YES: BK

4.0.2 GENERAL LOGIC TESTING

Below are 9 test cases for the Generic KB for Truth Table and DPLL, chaining cannot run the generic KB

GenericKB_1	TELL a => b; b & c; ~f || g; a <=> d; ASK b
Truth Table	Yes: 6
DPLL	This is the KB: ['a => b', 'b & c', '~f || g', 'a <=> d'] This is the QUERY: b YES: 1

GenericKB_2	TELL (e <=> (a => ~b)) & d & (d => e); a; ~h || i; ASK ~b & (~i => ~h)
Truth Table	Yes: 3
DPLL	This is the KB: ['(e <=> (a => ~b)) & d & (d => e)', 'a', '~h || i'] This is the QUERY: ~b & (~i => ~h) YES: 1

GenericKB_3	TELL (x <=> (y => ~z)) & w & (w => x); y; ~u || v; ASK z & (~v => ~u)
Truth Table	NO
DPLL	This is the KB: ['(x <=> (y => ~z)) & w & (w => x)', 'y', '~u || v'] This is the QUERY: z & (~v => ~u) YES: 1

GenericKB_4	TELL (x => y) & (z => w); (w & y <=> p); p => q; r || s; ASK q
Truth Table	NO
DPLL	This is the KB: ['(x => y) & (z => w)', '(w & y <=> p)', 'p => q', 'r || s'] This is the QUERY: q YES: 1

GenericKB_5	TELL ~a || b; (c & d => ~e); (f <=> ~g); h & i; ASK ~e & h
Truth Table	NO
DPLL	This is the KB: ['~a || b', '(c & d => ~e)', '(f <=> ~g)', 'h & i'] This is the QUERY: ~e & h YES: 1

GenericKB_6	TELL (a <=> b); (b <=> c); (c <=> d); (d <=> e); (e <=> f); ASK f
Truth Table	NO
DPLL	This is the KB: ['(a <=> b)', '(b <=> c)', '(c <=> d)', '(d <=> e)', '(e <=> f)'] This is the QUERY: f YES: 1

GenericKB_7	TELL (p || q) & (~p || r) & (q || s) & (~r || ~s); ASK p & q
Truth Table	YES: 2
DPLL	This is the KB: ['(p || q) & (~p || r) & (q || s) & (~r || ~s)'] This is the QUERY: p & q YES: 1

GenericKB_8	TELL (x => ~y) & (~y => z); (a & b => c) || (d & e => f); (g <=> h) & (~i || j); ASK z & f & j
Truth Table	YES: 264
DPLL	This is the KB: ['(x => ~y) & (~y => z)', '(a & b => c) || (d & e => f)', '(g <=> h) & (~i || j)'] This is the QUERY: z & f & j YES: 1

GenericKB_9	TELL (a || b) & (~a || b) & (a || ~b) & (~a || ~b); ASK a & ~a
Truth Table	No
DPLL	This is the KB: ['(a || b) & (~a || b) & (a || ~b) & (~a || ~b)'] This is the QUERY: a & ~a YES: 1

4.0.3 INDEPTH AND COMPARISON

1.TRUTH TABLE (TT)

How It Works

The TT algorithm evaluates all possible combinations of truth values for the symbols in the KB and checks:

- If **every clause in the KB** is true.
- If the **query** is also true for models where the KB is satisfied.

Steps for Execution

1. **Extract Symbols:**
    - The function parses the KB to identify all unique symbols (variables) present in the logical statements.
    - Example: From A & B => C, the symbols are A, B, C.
2. **Generate Models:**
    - The algorithm creates a **truth table** for all combinations of truth values for these symbols.
    - For n symbols, there are 2n2^n2n rows in the table.
3. **Evaluate Each Row:**
    - For each row (model), evaluate the truth value of:
        - Each clause in the KB.
        - The query.
    - If all clauses are true and the query is true for some rows, return YES with a count of valid models.
4. **Highlight Valid Models:**
    - Rows where the KB and query are true are marked (e.g., green text).

2. FORWARD CHAINING (FC)

How It Works

FC starts with **known facts** and applies **inference rules** to derive new facts until:

- The query is proved.
- All derivable facts have been exhausted without proving the query.

Steps for Execution

1. **Identify Initial Facts:**
    - Extract all symbols in the KB that do not depend on others (e.g., standalone facts like A).
2. **Generate Sentence List:**
    - Break each implication (A & B => C) into:
        - A premise (e.g., A & B).
        - A conclusion (e.g., C).
        - A count of how many premises need to be satisfied.
3. **Inference Process:**
    - Iteratively check if current facts satisfy the premises of any rule:
        - If satisfied, derive the conclusion and add it to the known facts.
    - Stop if the query is derived.

3. BACKWARD CHAINING (BC)

How It Works

BC works **backwards** from the query. It recursively tries to prove the query by:

- Checking if it's a known fact.
- Breaking down the premises of rules leading to the query into sub-goals.

Steps for Execution

1. **Start with the Query:**
    - Check if the query is directly in the KB or is a standalone fact.
2. **Expand Sub-Goals:**
    - For rules that lead to the query (e.g., A & B => C for query C):
        - Add A and B as new sub-goals to prove.
3. **Recursive Resolution:**
    - Continue until:
        - All sub-goals are facts in the KB (query is proved).

- A sub-goal cannot be resolved (query is disproved).

4. DPLL (DAVIS-PUTNAM-LOGEMANN-LOVELAND)

How It Works

DPLL is a more advanced SAT-solving algorithm. It:

- Works directly with **CNF clauses**.
- Uses optimizations like **unit propagation** and **pure literal elimination**.

Steps for Execution

1. **Convert KB to CNF:**
    - Each clause is split into literals (e.g., A | ~B).
2. **Base Case Checks:**
    - If all clauses are satisfied, return YES.
    - If any clause is unsatisfiable, return NO.
3. **Unit Propagation:**
    - Simplify clauses with **unit literals** (e.g., A forces A=True).
4. **Pure Literal Elimination:**
    - If a literal appears only in a positive or negative form (e.g., A but not ~A), assign it truthfully.
5. **Split:**
    - Choose an unassigned literal and try both True and False assignments recursively.

COMPARISON:

Method	Strengths	Weaknesses	Best Use Case	Computational Complexity	Key Operations
Truth Table (TT)	- Exhaustive and guarantees correctness. - Works with any propositional logic formula. - Simple to implement.	- Exponential growth in models as the number of symbols increases - Inefficient for large KBs.	Small KBs with a limited number of symbols.	O($\boldsymbol{2^n}.\boldsymbol{k}$), where $\boldsymbol{n}$ is the number of symbols and kkk is the number of clauses.	- Generate all truth value combinations. - Evaluate KB and query for each model.
Forward Chaining (FC)	- Linear complexity for Horn clauses. - Derives all logical consequences. - Efficient for large KBs with clear rules.	- Limited to **Horn clauses** (e.g., A & B => C). - Cannot solve problems with general propositional logic.	Reasoning from facts to derive all possible conclusions.	O($\boldsymbol{n}.\boldsymbol{k}$), where $\boldsymbol{n}$ is the number of symbols and kkk is the number of rules.	- Identify facts in KB. - Iteratively apply inference rules.
Backward Chaining (BC)	- Goal-directed reasoning avoids irrelevant computations. - Efficient for queries involving few rules.	- Requires well-structured KB. - Not suitable for deriving all conclusions. - Can encounter infinite recursion if not handled.	Proving specific queries in well-structured KBs.	Depends on query depth O($\boldsymbol{d}.\boldsymbol{k}$), where ddd is the depth of recursion and kkk is the number of rules.	- Start with query. - Decompose goals into sub-goals. - Recursively resolve sub-goals.
DPLL (Davis-Putnam-Logemann-Loveland)	Highly optimized for propositional satisfiability. - Handles general propositional logic, including non-Horn clauses. - Reduces search space with heuristics (unit propagation, pure literal elimination).	- Complex to implement. - May require advanced understanding of propositional logic.	Solving SAT problems or complex KBs with many clauses and variables.	Worst case: O($\boldsymbol{2^n}$). Average case: Sub-exponential with heuristics.	- Simplify clauses with unit propagation. - Eliminate pure literals. - Recursively explore truth assignments.

5. FEATURES/BUGS

5.1 FEATURES

- **Multiple Inference Methods**: Users can choose from different methods depending on their needs.
- **File Input**: The system reads from text files, making it easy to input new KBs and queries.

- **Recursive and Iterative Inference**: The system combines recursive and iterative approaches for effective reasoning.
- **GenericKB implementation**: We have modified the code from only read HornKB to read and run both Horn and Generic knowledge base.
- **GUI Table for better understanding**: We also create a table when the user run code for the genericKB test, it will provide all the possible value of each assignment into the table for better understanding

5.2 BUGS

- **Error Handling**: The system could benefit from better error handling for malformed input files. Currently, it may crash or fail silently when it encounters invalid formats.
- **Scalability**: The Truth Table method doesn't scale well for larger hornKBs due to its exponential time complexity.

6. RESEARCH

6.1 INFORMATION ABOUT RESEARCH COMPONENT

The research component of the project focused on comparing the performance of different inference methods. The aim was to understand their strengths and weaknesses, especially in terms of efficiency and scalability.

6.2 GENERAL PROPOSITIONAL LOGIC AND DPLL

DPLL is a well-established algorithm used for propositional satisfiability. By applying optimizations like unit propagation, it can handle much larger logical formulas than simpler methods like Truth Table.

6.3 DPLL IMPLEMENTATION

The DPLL algorithm was implemented with backtracking and optimizations to efficiently handle complex logical formulas. This method was particularly useful for large KBs, where other methods would be too slow.

6.4 OPERATOR COMPONENTS

In order to make the Truth Table to read Generic Knowledge Base, it requires some researches about methods to handle additional conditional and connective operators such as Disjunction (||), Negation (~), Biconditional (<=>) and bracket rules. In addition, these are non-Horn-form, that mean they can only be applied to Truth Table, not chaining.

6.4.1 Disjunction (||)

1. Definition
    a. The disjunction operator (||) represents a logical OR operation. It is true if at least one of its operands is true.
    b. Represents situations where multiple possibilities exist
    c. Example, if A and B are true then C must also be true: (A || B) => C
2. Challenges
    a. Each operand must be evaluated independently
3. Method
    a. Parsing:
        i. Detect the || operator in the clause during parsing
        ii. Split the clause into left and right sub-expressions
        iii. Example: A || B becomes left = "A", right = "B"

b. Evaluation
    i. If one of assignment evaluates to True, return True

6.4.2 Negation (~)

1. Definition
    a. The negation operator (~) inverts the truth value of its operand. If a proposition is true, its negation is false
2. Method handlinga
    a. When a negation (~) is encountered, strip the ~ and treat the remainder as the operand. The evaluation phase then flips the truth value of the operand and the evaluation returns the negated value of the assignment

6.4.3 Biconditional (<=>)

1. Definition
    a. Biconditional represents that both sides of the expression rely each other. It is true if both operands have the same truth value
2. Method handilng
    a. The expression is split into left and right components. For example, A => B and B => A
    b. These sub-expressions are evaluated individually and their conjunction determines the truth of the biconditional

6.4.4 Bracket rules

1. Brackets are used to prirotize certain logical expressions. When the code detected braket, the expressions inside the brackets will be analyze first.
2. Each entry in the knowledge base is assigned a bracket level to ensure logical expressions are evaluated in the proper order.

7. STUDENT CONTRIBUTIONS

7.1 HORN KNOWLEDGE BASE IMPLEMENTATION:

- **Truth Table Implementation:** Le Hoang Minh was responsible for designing and coding the Truth Table functionality for Horn KBs. This involved creating a systematic approach to generating truth assignments and evaluating logical entailment. Minh’s implementation ensured that all possible scenarios were considered, providing accurate results for smaller, manageable Horn-form KBs

- **Chaining Implementation:** Nguyen Hoang Minh contributed by implementing both Forward Chaining and Backward Chaining methods. These methods allowed for linear reasoning from facts to conclusions and goal-driven reasoning to verify queries, respectively. Minh's chaining solutions were efficient for processing Horn KBs and demonstrated their suitability for handling specific types of logical entailment.

7.2 GENERIC KNOWLEDGE BASE IMPLEMENTATION:

- **First Version of the Code:** Le Hoang Minh developed the initial version of the Generic KB implementation, extending the functionality of the inference engine to handle logical operators such as biconditional, negation, and disjunction. However, during testing, certain issues were

identified. Specifically, for test case 9, the Truth Table method incorrectly returned YES instead of NO. Additionally, test case 8 highlighted memory limitations due to the large size of the KB, which prevented the system from producing results effectively

- **Debugging and GUI Table Creation:** Nguyen Hoang Minh focused on addressing the issues in the first version. He debugged the Truth Table logic, ensuring correctness for edge cases like test case 9. Additionally, Minh developed a graphical user interface (GUI) table to enhance the system's usability and presentation. This table provided a clear, structured view of truth assignments and their evaluations, improving the readability and interpretability of the results.

7.3 REPORT PAPER

- **Writing the Report:** Le Hoang Minh undertook the task of drafting the report, which detailed the implementation process, testing outcomes, and analysis of the inference methods. Minh's writing captured the technical details while maintaining clarity for readers unfamiliar with the project
-
- **Formatting and Visualization:** Nguyen Hoang Minh focused on refining the report's layout and presentation. He enhanced its readability by organizing content into well-defined sections, incorporating visual elements like tables and charts, and ensuring a professional appearance. Minh's efforts made the report visually engaging and easy to follow, contributing to its overall impact.

8. CONCLUSION

This propositional logic inference system successfully implements four different reasoning methods: Truth Table, Forward Chaining, Backward Chaining, and DPLL. These methods cater to different problem sizes and complexity levels, providing a balanced mix of completeness and efficiency. Furthermore, we managed to implement more for the Truth Table and DPLL to read and understand test case that contain Generic Knowledge Base. The project also offered valuable insights into the trade-offs between various inference techniques, helping us appreciate the strengths and weaknesses of each approach.

9. ACKNOWLEDGEMENTS/RESOURCES

We used various resources, including textbooks on logic and automated reasoning, online articles, and research papers on propositional satisfiability. Python was the primary language used for implementation.

During this assignment, we relied on several valuable resources to enhance our understanding and implementation:

- **Rich While-Cooper's Video:** The tutorial available at this link offered a straightforward approach to creating and evaluating Python truth tables. The clear explanations and practical examples provided were instrumental in simplifying the process of assigning True/False values within the truth table, making our logical evaluations more efficient and comprehensible.
- **Neural Nine Tutorial:** The video at this link provided an excellent guide on using the Python tabulate library. It showcased various features for generating professional-looking tables, which significantly improved the readability and presentation of our data. This resource contributed to creating polished and visually appealing output for the project.

10. REFERENCES

1. Davis, M., Putnam, H., Logemann, G., & Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM, 5*(7), 394–397. https://doi.org/10.1145/368273.368557
2. Russell, S., & Norvig, P. (2010). *Artificial intelligence: A modern approach* (3rd ed.). Pearson.
3. Russell, S., & Norvig, P. (2020). *Artificial intelligence: A modern approach* (4th ed.). Prentice Hall.
4. LibreTexts. (2024). Propositional logic: Logical analysis using truth tables. *LibreTexts*. https://human.libretexts.org/Bookshelves/Philosophy/Thinking_Well_-_A_Logic_And_Critical_Thinking_Textbook_4e_(Lavin)/07%3A_Propositional_Logic/7.05%3A_Logical_Analysis_using_Truth_Tables [Accessed: May 12, 2024].
5. Hayes-Roth, F., Waterman, D., & Lenat, D. (1983). *Building expert systems*. Addison-Wesley.
6. Feigenbaum, E. (1988). *The rise of the expert company*. Times Books.
7. Davis, M., Logemann, G., & Loveland, D. (1961). A machine program for theorem proving. *Communications of the ACM, 5*(7), 394–397. https://doi.org/10.1145/368273.368557
8. Whitesitt, J. E. (2012). *Boolean algebra and its applications*. Courier Corporation.
9. Whitesitt, J. E. (2012). *Boolean algebra and its applications* (ISBN 978-0-486-15816-7). Courier Corporation.
10.