PROCESS 2 - FORWARD CHAINING IN PROPOSITIONAL LOGIC

1st Pham Dang Anh Ngoc
Faculty of Information Technology
Ton Duc Thang University
Ba Ria Vung Tau City, Viet Nam
523h0162@student.tdtu.edu.vn

2nd Nguyen Dang Nam Khanh
Faculty of Information Technology
Ton Duc Thang University
Da Nang City, Viet Nam
523h0148@student.tdtu.edu.vn

3rd Nguyen Chi Vy
Faculty of Information Technology
Ton Duc Thang University
Ca Mau City, Viet Nam
523h0197@student.tdtu.edu.vn

4th Bui Quang Vinh
Faculty of Information Technology
Ton Duc Thang University
Ho Chi Minh City, Viet Nam
523h0193@student.tdtu.edu.vn

5th Nguyen Thanh An
Faculty of Information Technology
Ton Duc Thang University
Ho Chi Minh City, Viet Nam
nguyenthanhan@tdtu.edu.vn

***Abstract*—This project implements the Forward Chaining algorithm in Propositional Logic to determine whether a given knowledge base (KB) entails a specific query symbol. Using an object-oriented programming (OOP) model in Python, each clause is represented with its premises and conclusion, and inference is carried out through a count-tracking mechanism. A directed graph is automatically generated using Graphviz to visualize the logical relationships in the KB. The system is developed and executed on Google Colab. This report presents the problem-solving approach, system architecture, visual outputs, and each member's contributions.**

***Index Terms*—Forward Chaining, Propositional Logic, Inference, Knowledge Base, Python, Graphviz.**

I. Introduction to Forward Chaining

Forward chaining, also known as forward deduction or forward reasoning, is a data-driven inference method commonly used in Artificial Intelligence. It begins with known facts and repeatedly applies inference rules whose premises are satisfied, adding their conclusions to the knowledge base. This process continues until the desired query is derived or no further inference is possible. In this project, we work with definite clauses and implement the PL-FC-ENTAILS algorithm to test whether a given knowledge base entails a specific query symbol.

II. System Design and Implementation

A. Clause and Literal Representation

Literal class: Represents a single propositional symbol in a logical clause, which may be negated or non-negated. It is used to define the basic building blocks of clauses in a knowledge base, with:

- `symbol`: the name of the logical variable.
- `negated`: `true` if the literal is negated (e.g., `-a`).
- Includes a `__str__()` method to display the string, and a `draw_node()` method to draw the node representation.

Clause class: Represents a logical clause, which is a disjunction of literals (e.g., "-A -B C"). It is used to encode rules or facts in the knowledge base. Key methods include:

- `premise()`: returns the list of premise symbols.
- `conclusion()`: returns the conclusion symbol.
- `count_known_premises(state)`: counts how many premises are known to be true in the current state.
- `is_fact()`: determines whether the clause is a fact (i.e., a single positive literal).
- `add_literal()`: adds a literal to the clause.

B. Forward Chaining Algorithm

The function `forward_chaining(kb, query)` is implemented based on the `PL-FC-ENTAILS` algorithm. It is used to determine whether a particular query can be inferred from a Knowledge Base (KB) using the Forward Chaining strategy.

The KB consists of `Clause` objects. The algorithm maintains:

- `agenda`: a list of known facts (positive literals) that need to be processed.
- `count`: a dictionary that tracks, for each clause, the number of its premises (negative literals) that have not yet been proven true.

Initialization:

- The `agenda` is initialized with all known facts, i.e., all clauses that are single positive literals (facts).
- The `count` dictionary is initialized such that for each clause, it stores the number of negative literals (premises) it contains.

Main loop:

1) While the `agenda` is not empty:
   a) Pop a fact (literal) from `agenda`.
   b) If the fact matches the `query`, return `True`.

c) Otherwise, for each clause that contains this fact as a premise:
   - Decrement its count.
   - If the count becomes zero, it means all premises are known to be true, so the conclusion can be inferred.
   - Add the conclusion to `agenda` to continue propagation.

```
Processing symbol: d
 Marking d as inferred

  Clause: b -d, Known premises: 1/1
   Premise count decremented to 0
    All premises satisfied for b -d, adding conclusion b to agenda
  Clause: f -b -d -h, Known premises: 1/3
   Premise count decremented to 2
  Clause: c -d, Known premises: 1/1
   Premise count decremented to 0
    All premises satisfied for c -d, adding conclusion c to agenda
  Clause: h -d -e -f, Known premises: 1/3
   Premise count decremented to 2
  Clause: c -a -d, Known premises: 1/2
   Premise count decremented to 1
```

Fig. 1. Visualization of entailment checking steps starting from known fact

Termination: If the agenda becomes empty and the query has not been inferred, return `False`, indicating that the query is not entailed by the knowledge base.

Note: Optionally, a set of already inferred symbols can be maintained to avoid redundant processing of the same facts multiple times.

C. Graph Visualization with Graphviz

To enhance the understanding of logical inference, each clause in the knowledge base is visualized as a directed graph using the `Graphviz` library. This graphical representation helps to observe how premises lead to conclusions in a clear and intuitive way.

Visualization Method:

- Each literal is represented as a node in the graph.
- For each clause, directed edges are created from each premise (negative literal) to the conclusion (positive literal).
- The clause is assumed to be in the form of a disjunction (e.g., `-a -b c`), and the visual translation considers that `a` and `b` imply `c`.

Example: Given the clause `-a -b c`, the graph will contain:

- Nodes: `a`, `b`, `c`
- Edges: `a` $\rightarrow$ `c`, `b` $\rightarrow$ `c`

This implies that both `a` and `b` are required for inferring `c`.

Implementation Details:

- The method `Clause.draw()` is responsible for rendering the graph of each clause.
- It uses the `graphviz.Digraph` object to add nodes and edges.
- Each call to `draw()` outputs a visual diagram that corresponds to one logical clause, helping users verify and trace the reasoning process visually.

This approach is particularly useful for educational purposes, debugging logic rules, and ensuring correctness in knowledge-based systems.

III. Experiment and Results

A. Test Objective

- Evaluate the system's inference performance for different queries.
- Check whether the system can entail a specific query (e.g. h).
- Record the number of inference steps through the sequence of processed symbols.
- Evaluate performance as the number of propositions and symbols increases.

B. Test Dataset and Inference Results

Input: The input of the system is a list of *definite clauses*, each formatted as a *Horn clause*—a disjunction of literals containing at most one positive literal. For example:

TABLE I
Conversion from input clauses to Horn clause representation

Input Clauses	Horn Clause Representation
a -b -c	$b \wedge c \rightarrow a$
b -d	$d \rightarrow b$
c -e -f	$e \wedge f \rightarrow c$
d -g -h	$g \wedge h \rightarrow d$
e -a	$a \rightarrow e$
f -b -d -h	$b \wedge d \wedge h \rightarrow f$
g -c -e	$c \wedge e \rightarrow g$
h -f	$f \rightarrow h$
a -g -h	$g \wedge h \rightarrow a$
b -a -e -f	$a \wedge e \wedge f \rightarrow b$
c -d	$d \rightarrow c$
d -b -g	$b \wedge g \rightarrow d$
e -h	$h \rightarrow e$
f -a -c	$a \wedge c \rightarrow f$
g -b -h	$b \wedge h \rightarrow g$
h -d -e -f	$d \wedge e \wedge f \rightarrow h$
a -c	$c \rightarrow a$
b -f -g -h	$f \wedge g \wedge h \rightarrow b$
c -a -d	$a \wedge d \rightarrow c$
d -e -g	$e \wedge g \rightarrow d$
d	**Initial Fact:** d

Each clause can be interpreted as a logical rule:

- If all *negative literals* (the premises) are known to be true, then the *positive literal* (the conclusion) can be inferred.
- For instance, the clause `a -b -c` means "if `b` and `c` are true, then `a` is true."

The input is parsed and converted into a list of `Clause` objects, where:

- Premises: consist of all negative literals (e.g., `a`, `d`),
- Conclusion: is the single positive literal (e.g., `c`).

Each clause represents a logical state, and together they form the *Knowledge Base* (KB). This KB serves as the input to the *Forward Chaining* algorithm.

Output: The output of the system is a *boolean value* indicating whether the given *query symbol* (e.g., `e`) is entailed by the knowledge base:

- `True`: if the query can be inferred through a series of rule applications,
- `False`: if the query cannot be reached from the known facts and rules.

```
Path: ['d', 'b', 'c', 'a', 'e', 'f', 'g', 'h']
Can KB entail h? → True
```

Fig. 2. Entailment result

C. Graph Visualization with Graphviz

In addition to the boolean result, the system also generates a *directed graph* to visualize the inference process:

- Each edge represents a logical implication (from premises to conclusion),
- The graph aids in understanding, explaining, and verifying the reasoning chain visually.

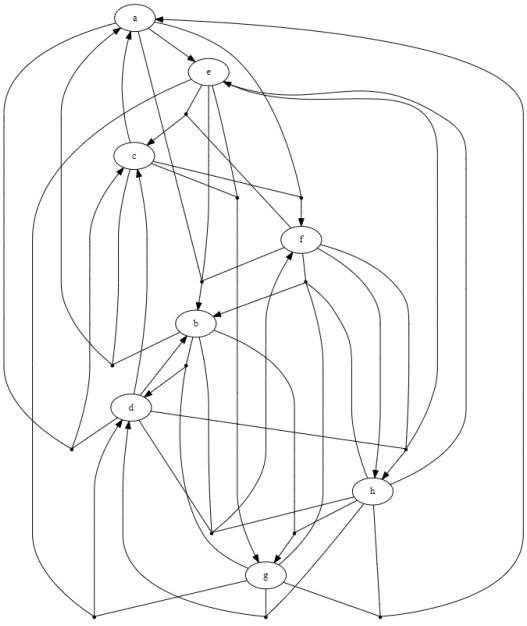

Fig. 3. Forward Chaining Inference Graph.

IV. Contributions

TABLE II
Individual Contributions

Member	Contribution	Evaluation
Pham Dang Anh Ngoc	Code implementation and testing	100%
Nguyen Dang Nam Khanh	Code implementation and testing	100%
Nguyen Chi Vy	Graphviz visualization, report writing	100%
Bui Quang Vinh	Graphviz visualization, report writing	100%

V. Self-Evaluation

- **Task 1 (Forward Chaining):** Fully implemented and verified with multiple test cases.
  Score estimate: 8.0/8.0
- **Task 2 (Report):** Structured in IEEE format with complete and coherent content.
  Score estimate: 2.0/2.0

VI. Conclusion

This project successfully implements the Forward Chaining algorithm for propositional logic inference using an object-oriented approach. The system is able to effectively determine whether a given knowledge base entails a specific query by systematically processing known facts and applying inference rules step-by-step.

The inference process was demonstrated with clear output showing the sequence of symbols inferred and the number of inference steps taken to reach a conclusion. The final results confirm that the query can be entailed by the knowledge base, validating the correctness of the algorithm implementation.

Future work could explore the scalability of the system by testing with larger knowledge bases, optimizing inference efficiency, and integrating backward chaining or other inference methods for comparison.

References

[1] M. T. Reilly, "Forward Chaining vs. Backward Chaining," *Built In*, [Online]. Available: https://builtin.com/artificial-intelligence/forward-chaining-vs-backward-chaining, [Accessed: May 15, 2025].