**COS30019 – Introduction to Artificial Intelligence**

**Assignment 2 – Inference Engine**

Student Name: Nguyen Dinh Dung

Student ID: 104772138

A black and red sign with white text

Description automatically generated

Table of Contents

[Instructions 3](#_Toc182246074)

[Introduction 4](#_Toc182246075)

[Algorithms 9](#_Toc182246076)

[Testing 14](#_Toc182246077)

[Features/Bugs/Missing 38](#_Toc182246078)

[Research 39](#_Toc182246079)

[Conclusion 41](#_Toc182246080)

[References 43](#_Toc182246081)

# Instructions

1. Python Installation:

* Get the most recent version of Python from https://www.python.org/downloads/, the official Python website.
* Ensure that the "Add Python to PATH" option is chosen throughout the installation process. This crucial step allows you to run Python commands directly from your terminal or command prompt.

1. Visual Studio Code Installation:

* Visit the official website, https://code.visualstudio.com/, to download and install Visual Studio Code.

1. Python Library Installation:

* Tabulate import: pip install tabulate
* Lark import: pip install lark
* Regex import: pip install regex

1. Python Extension Installation:

* Get Visual Studio Code started.
* Use the keyboard shortcut Ctrl+Shift+X to open the Extensions window, or click the Extensions icon in the Activity Bar.
* Look for "Python" in the marketplace and download the official Microsoft Python extension.
* To confirm that Python has been installed, open a new terminal in Visual Studio Code by selecting Terminal -> New Terminal.
* Run the python --version command. The installed version of Python ought should appear here.
* Run the pip —version command. This ought to show the pip version, which is the Python package installer.
* Creating a Python Project: • We can establish a specific folder for the Python project in VS Code.
* Use File => Open Folder in VS Code to access this folder.
* Inside the folder, create a new Python file (such as main.py).
* Running the code:
* Open a terminal and navigate to the folder where the Python files are located.
* To execute the algorithm in either python or cmd, use the following command format:

**python iengine.py <file\_name> <method\_name> or .\iengine <file\_name> <method\_name>**

For example, to run the FC (Forward chaining) algorithm on test\_HornKB.txt, type:

A black screen with white text

Description automatically generated

Figure 1. Example

Replace <method\_name> with one of the following options to see different search algorithms:

1. FC for Forward Chaining
2. BC for Backward Chaining
3. TT for Truth Table
4. DPLL for Davis–Putnam–Logemann–Loveland (DPLL) algorithm

# Introduction

**1. Overview:**

This project implements a propositional logic inference engine, capable of determining whether a given query can be entailed from a provided knowledge base. The inference engine supports multiple inference methods: Truth Table Checking (TT), Forward Chaining (FC), Backward Chaining (BC), and DPLL (Davis-Putnam-Logemann-Loveland). Each of these algorithms plays a critical role in determining entailment, although they vary in efficiency and the types of knowledge bases they can process.

**Knowledge Bases: Horn Form and Generic Form**

The knowledge base (KB) in propositional logic consists of a set of clauses that represent known facts and implications. These clauses can either be in Horn form or generic form:

* Horn Form: A restricted form of clauses where each clause has at most one positive literal. Horn clauses are well-suited for efficient inference as they allow streamlined reasoning, especially with the Forward Chaining and Backward Chaining algorithms.
* Generic Form: A more general form that allows multiple positive literals in each clause. This form supports a wider range of logical expressions but can be more complex to process, particularly for chaining methods. The Truth Table Checking and DPLL algorithms are capable of handling both Horn and generic forms.

**Algorithms Implemented**

Each inference method in the engine serves distinct purposes and is suited for different types of queries or KB configurations:

1. Truth Table Checking (TT): This method evaluates all possible truth assignments to determine if the knowledge base entails the query. Although comprehensive, TT is computationally expensive for large knowledge bases due to exponential growth in possible assignments. TT can handle both Horn and generic knowledge bases effectively.
2. Forward Chaining (FC): FC is a data-driven approach suitable for Horn-form KBs. Starting from known facts, it deduces new facts by applying implication rules in the KB until it either derives the query or exhausts all possibilities. FC is efficient for Horn KBs but not suitable for generic forms with complex logical expressions.
3. Backward Chaining (BC): BC, in contrast to FC, is a goal-driven approach that starts from the query and works backward through implications in the KB to find supporting facts. Like FC, it is efficient for Horn KBs but struggles with more complex or generic forms.
4. DPLL (Davis-Putnam-Logemann-Loveland): The DPLL algorithm is a refined approach derived from the Truth Table method and is often used for propositional satisfiability problems (SAT). DPLL includes optimizations such as unit propagation and pure literal elimination, which allow it to efficiently search for a model that satisfies the KB while attempting to entail the query. DPLL can work with both Horn and generic forms, making it versatile and efficient for large KBs.

**Program Operation and Input**

The inference engine takes a KB and a query, which are read from a text file. The KB is defined after the keyword TELL and consists of clauses separated by semicolons, while the query follows the keyword ASK. The user specifies the inference method (TT, FC, BC, or DPLL) via the command line, and the engine returns either YES or NO, indicating whether the query can be derived from the KB. For TT and DPLL, YES outputs the number of models satisfying the KB; for FC and BC, it returns the symbols entailed in the process.

This report will discuss the implementation details of each algorithm, demonstrate their behavior with various test cases, and analyze their effectiveness in handling different KB configurations. Special emphasis will be placed on the DPLL algorithm's advantages in handling larger, more complex KBs, as well as the performance differences when working with Horn versus generic forms.

**2. Terminology:**

* **Logical connectives include:**
* AND (&)
* OR (||)
* NOT (~)
* IMPLICATION (=>)
* BICONDITIONAL (<=>)
* **Sentence**:

Represents a logical expression, which can be evaluated given a set of truth values for its symbols.

*Example*: A statement like A⇒BA \Rightarrow BA⇒B or A∧BA \land BA∧B.

* **Symbol**:

A basic unit of logic with a specific name and truth value.

It’s a subclass of Sentence and represents propositional variables.

*Example*: In model, a Symbol named "A" could be assigned True or False.

* **Negation (~)**:

Represents logical negation (NOT) of a Symbol.

Returns the opposite truth value of the evaluated Symbol.

*Example*: If A is True, then ~A is False.

* **Conjunction (&)**:

Represents logical conjunction (AND) of multiple Symbols or expressions.

Evaluates to True if all included expressions evaluate to True.

*Example*: A∧BA \land BA∧B is True only if both A and B are True.

* **Disjunction (||)**:

Represents logical disjunction (OR) of two Symbols or expressions.

Evaluates to True if at least one included expression evaluates to True.

*Example*: A∨BA \lor BA∨B is True if either A or B is True.

* **Implication (=>)**:

Represents logical implication; evaluates as True if the first expression (premise) being True leads to the second (conclusion) being True.

*Example*: A⇒BA \Rightarrow BA⇒B is False only if A is True and B is False; otherwise, it is True.

* **Biconditional (<=>)**:

Represents logical biconditional (IFF), where both expressions must have the same truth value to evaluate as True.

*Example*: A⇔BA \Leftrightarrow BA⇔B is True if A and B are both True or both False.

* **Model**:

A dictionary of truth values assigned to each Symbol.

Used to evaluate logical expressions within the Sentence subclasses.

*Example*: { 'A': True, 'B': False } is a model assigning True to A and False to B.

* **Evaluate**:

A method that calculates the truth value of a logical expression given a model.

Each subclass of Sentence has a custom implementation of evaluate based on its logical operation.

* **Symbols**:

A method that returns all unique Symbols (propositional variables) used in an expression.

Useful for ensuring that all variables are accounted for when evaluating a model.

* **Knowledge Base (KB)**:

A collection of known facts and rules represented as logical sentences.

Used to infer new truths through logical evaluation.

* **Query**:

A logical expression that the algorithm aims to prove true based on the KB.

* **Model Check**:

Function that checks whether a knowledge base entails a query.

Uses recursive check\_all to verify entailment by evaluating all possible truth assignments.

* **Check All**:

Helper function within model\_check, which exhaustively explores all possible truth assignments of Symbols.

Used to ensure that if the KB is true, the query also holds true for all possible assignments.

* **Recursive Splitting**:

A process where truth assignments are divided recursively into branches, enabling systematic exploration of models to determine entailment.

* **Entailment**:

A relationship where the truth of one statement (query) is guaranteed by the truth of another (knowledge base).

* **Satisfiability (SAT)**: The core problem solved by DPLL, which determines if there exists an assignment of truth values to variables that makes a given propositional formula true.
* **Backtracking**: DPLL systematically explores possible assignments for variables and backtracks when contradictions are found.
* **Unit Propagation**: A simplification step where clauses with a single unassigned literal force that literal to be true, reducing the search space.
* **Pure Literal Elimination**: Identifies literals that appear with only one polarity (either only true or only false) across all clauses, allowing them to be assigned accordingly.
* **Clause**: A disjunction (OR) of literals that represent conditions that must be satisfied.
* **Efficiency in Horn KBs**: FC is highly efficient with Horn-form KBs, as it incrementally builds up knowledge without needing exhaustive search.
* **Recursive Inference**: For each goal, BC recursively seeks supporting facts or rules, decomposing complex queries into subgoals.
* **Subgoal**: Intermediate queries that need to be satisfied in order to prove the main goal.
* **Knowledge Base (KB)**: A structured set of facts and rules used by BC to deduce the goal, ideally in Horn form for efficient chaining.
* **Horn Clause**: Essential for BC efficiency, Horn clauses allow BC to simplify logical deductions by focusing on simple implications.
* **Call Stack**: The list of active goals BC is attempting to prove, which can be thought of as the depth of recursive function calls.
* **Dependency Chain**: The sequence of intermediate facts or rules required to infer the goal, representing logical dependencies.
* **Efficiency in Goal-Oriented Tasks**: BC is effective when specific queries need to be verified, as it minimizes search to only those parts of the KB relevant to the goal.
* **Exhaustive Search**: TT generates all possible truth assignments to variables in the KB, checking each assignment to see if it satisfies the query.
* **Truth Assignment**: Each row in the truth table represents a unique assignment of true or false to each propositional variable.
* **Boolean Logic**: The foundational logic system of TT, based on true/false values of variables and the logical operations AND, OR, NOT.

# Algorithms

|  |  |  |
| --- | --- | --- |
| Strategies | Description | Method |
| Forward Chaining | Forward Chaining (FC) is a data-driven inference method used to derive conclusions by applying rules to known facts iteratively. Starting with the initial facts in the knowledge base, FC checks if any rule’s premises are satisfied, allowing the rule’s conclusion to be inferred as a new fact. This process repeats, adding new facts and triggering additional rules until no more rules can be applied or the goal is reached. | FC |
| Backward Chaining | Backward Chaining (BC) is a goal-driven inference method that attempts to prove a specific query by working backward from the goal. It starts with the query (the goal to be proven) and tries to find rules that would conclude this goal. For each rule that could lead to the goal, it examines the rule’s premises, treating each premise as a subgoal. BC recursively attempts to prove each subgoal, continuing until it reaches known facts or determines that the goal cannot be proven. | BC |
| Truth Table | Truth Table (TT) inference is a brute-force method in propositional logic that tests all possible truth assignments for the symbols in a knowledge base (KB) to determine if a given query can be inferred. TT constructs a table where each row represents a unique combination of truth values for all symbols, and each row is evaluated against the statements in the KB to see if it makes them all true. If every model that satisfies the KB also satisfies the query, then the query is entailed by the KB. | TT |
| Davis-Putnam-Logeman-Loveland | The **Davis–Putnam–Logemann–Loveland (DPLL)** algorithm is an optimized approach for checking the satisfiability of propositional logic formulas, particularly in Conjunctive Normal Form (CNF). DPLL builds upon the basic truth table approach but incorporates several efficiencies to make it faster and more suitable for complex problems. | DPLL |

**1. Forward Chaining:**

A screenshot of a computer program

Description automatically generated

Figure 3. Pseudo Code

**2. Backward Chaining:**

A screenshot of a computer program

Description automatically generated

Figure 4. Pseudo Code

**3. Truth Table:**

A screenshot of a computer program

Description automatically generated

Figure 5. Pseudo Code

**4. DPLL:**

A screenshot of a computer program

Description automatically generated

Figure 6. Pseudo Code

Execution

The execution here will be executed based on the test1.txt that the document provided so we can validate the output of each algorithm. **1. Forward Chaining:**

A computer screen shot of a black screen

Description automatically generated

Figure 7. Forward Chaining result

**2. Backward Chaining:**

A computer screen shot of a black screen

Description automatically generated

Figure 8. Backward Chaining result

**3. Truth Table:**

A screen shot of a black background

Description automatically generated

Figure 9. Truth Table result

**4. DPLL:**

A computer screen shot of a black screen

Description automatically generated

Figure 10. DPLL result

# Testing

**A. Overview Test Cases:**

To evaluate the efficacy of the four algorithms(FC, BC, TT, DPLL), sixteen unique test situations were created. The test examples are intended to highlight the benefits and drawbacks of every algorithm in various scenarios. Use the same format that was used during the instruction session to administer the test.

For example, **python iengine.py test1.txt FC** to run Forward Chaining on test 1

**B. Test Cases:**

1. Reasoning in propositional logic:

A computer screen shot of a black background

Description automatically generated with medium confidence A black background with white text

Description automatically generated

A computer code with white text

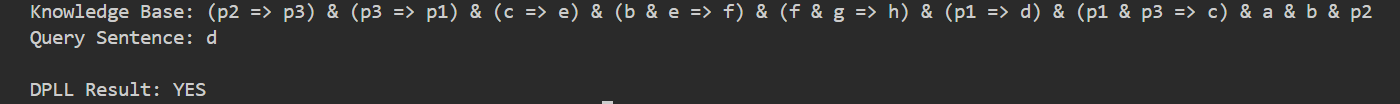
Description automatically generated A black background with white text

Description automatically generated

A screen shot of a computer

Description automatically generated

A black screen with white text

Description automatically generated 

CONCLUSION

|  |  |  |  |
| --- | --- | --- | --- |
| Methods | Advantages | Disadvantages | Demonstration |
| Forward Chaining | Efficient when facts are known and conclusions follow. | No backtracking, may miss alternative paths. | Best for situations with clear, direct paths to the goal. |
| Backward Chaining | Focuses on the query and avoids irrelevant paths. | Can explore unnecessary paths if query is complex. | Effective for proving specific queries without extra work. |
| Truth Table | Thorough, guarantees correctness. | Computationally expensive, exponential growth. | Exact, but inefficient for large problems. |
| DPLL | More efficient than Truth Tables, handles large problems. | Complex and requires heuristics for best performance. | More efficient than brute force, but can still be slow for some cases. |

1. Simple chain of implications:

A screen shot of a computer

Description automatically generated A black background with white text

Description automatically generated

A screen shot of a computer

Description automatically generated A black background with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

A black background with white text

Description automatically generated

CONCLUSION

|  |  |  |  |
| --- | --- | --- | --- |
| Methods | Advantages | Disadvantages | Demonstration |
| Forward Chaining | Very efficient if the goal is directly available. | May miss paths if the goal isn't already a fact. | Efficient when the goal is already in the knowledge base. |
| Backward Chaining | Focuses on the query and avoids irrelevant paths. | Needs more effort if the goal is not a fact. | Directly useful for querying known facts. |
| Truth Table | Exhaustive and guarantees correctness. | Infeasible for larger problems due to exponential growth. | Exact but inefficient for problems with many variables. |
| DPLL | More efficient than Truth Tables, handles large problems. | Complex and requires heuristics for best performance. | Efficient for satisfiability checking with many variables. |

1. Chained Implications with Direct Fact:

A black background with white text

Description automatically generatedA black screen with white text

Description automatically generated

A black screen with white text

Description automatically generatedA black background with white text

Description automatically generated

A black screen with white text

Description automatically generated

A black background with white and black symbols

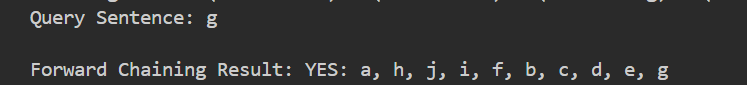
Description automatically generated

CONCLUSION

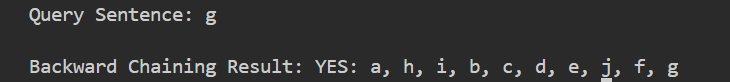
|  |  |  |  |
| --- | --- | --- | --- |
| Methods | Advantages | Disadvantages | Demonstration |
| Forward Chaining | Successfully propagates facts from a to d, even with additional irrelevant facts (like e and l). | May miss paths if the goal isn't already a fact. | Efficient when the goal is already in the knowledge base. |
| Backward Chaining | Still finds d by tracing through the chain, starting from the goal and checking the relevant facts. | Needs more effort if the goal is not a fact. | Directly useful for querying known facts. |
| Truth Table | Evaluate all combinations and determine that d holds. | Effective but inefficient, as it evaluates unnecessary facts (like e and l) that do not affect the outcome. | Exact but inefficient for problems with many variables. |
| DPLL | Evaluate satisfiability with the given facts and find the negation of d unsatisfiable. | Complex and requires heuristics for best performance. | Efficient for satisfiability checking with many variables. |

1. Chained Implications with Multiple Paths:

A black background with white text

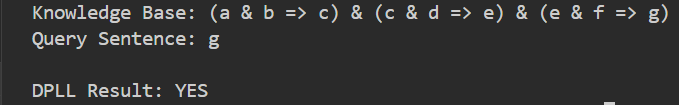
Description automatically generated 

A black screen with white text

Description automatically generated 

A screenshot of a computer

Description automatically generated

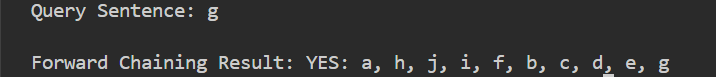


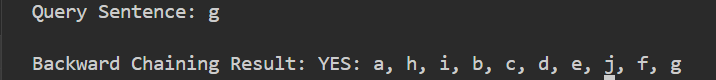
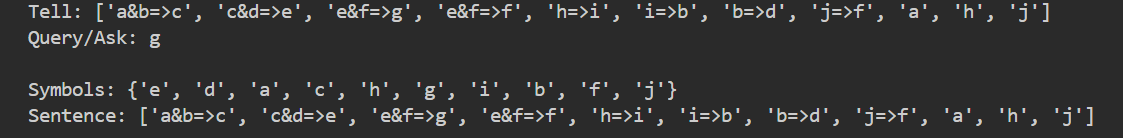
CONCLUSION

|  |  |  |  |
| --- | --- | --- | --- |
| Methods | Advantages | Disadvantages | Demonstration |
| Forward Chaining | Successfully infers **g** by applying rules from facts **a**, **h**, and **j**. Gradually apply rules to reach the goal. | Can explore unnecessary facts and conditions that don’t lead directly to the goal. | Efficient when the goal is already in the knowledge base. |
| Backward Chaining | Proves **g** by tracing backward from the goal, satisfying conditions along the way. | May explore many paths or branches, leading to inefficiencies in complex scenarios. | Directly useful for querying known facts. |
| Truth Table | Exhaustive, guarantees correctness. | Exhaustively checks all combinations of truth values for symbols and concludes **g**. | Exact but inefficient for problems with many variables. |
| DPLL | Successfully solves using unit propagation and pure symbol elimination. | More complex than necessary for small problems. Requires conversion to CNF. | Efficient for satisfiability checking with many variables. |

1. Causal Inference with Complex Rule Chains:

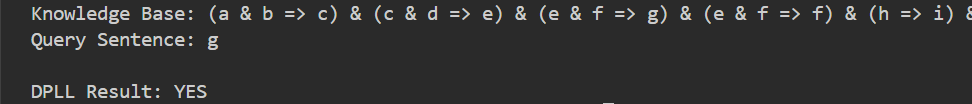
A black screen with white text

Description automatically generated



A screen shot of a computer

Description automatically generated

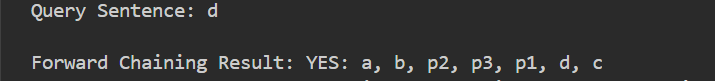


CONCLUSION

|  |  |  |  |
| --- | --- | --- | --- |
| Methods | Advantages | Disadvantages | Demonstration |
| Forward Chaining | Successfully infers **g** using facts **a**, **h**, and **j** by applying rules. | Can explore unnecessary facts and conditions that don’t lead directly to the goal. | Efficient when the goal is already in the knowledge base. |
| Backward Chaining | Proves **g** by tracing backward from the goal, satisfying conditions along the way. Explores only relevant conditions. | May explore many paths or branches, leading to inefficiencies in complex scenarios. | Directly useful for querying known facts. |
| Truth Table | Exhaustively checks all combinations of truth values for symbols and concludes **g**. | Computationally expensive for problems with many symbols. Inefficient for large KBs. | Exact but inefficient for problems with many variables. |
| DPLL | Prunes search space, reducing unnecessary evaluations. | More complex than necessary for small problems. Requires conversion to CNF. | Efficient compared to Truth Table. |

1. Causal Reasoning with Multiple Dependencies:

A computer screen with white text

Description automatically generated

A computer code with white text

Description automatically generated with medium confidenceA black background with white text

Description automatically generated

A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

CONCLUSION

|  |  |  |  |
| --- | --- | --- | --- |
| Methods | Advantages | Disadvantages | Demonstration |
| Forward Chaining | Successfully infers **d** starting from the facts **a**, **b**, and **p2** by applying the given rules. | Can explore unnecessary facts or rules that do not contribute to the goal, leading to inefficiency. | Efficient when the goal is already in the knowledge base. |
| Backward Chaining | Derives **d** by working backward from the goal and finding rules that satisfy conditions along the way. | May explore many paths or branches, leading to inefficiencies in complex scenarios. | Directly useful for querying known facts. |
| Truth Table | Exhaustively checks all possible combinations of truth values for symbols and concludes that **d** can be inferred. | Computationally expensive, especially for larger KBs, due to exponential time complexity. | Exact but inefficient for problems with many variables. |
| DPLL | Solves using unit propagation and pure symbol elimination, proving **d** by checking satisfiability. | Requires transformation into CNF, which adds complexity, and might be overkill for smaller problems. | Efficient compared to Truth Table. |

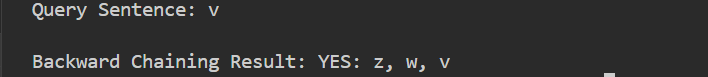
1. Symbol Propagation with Multiple Implications:

A black screen with white text

Description automatically generatedA black background with white text

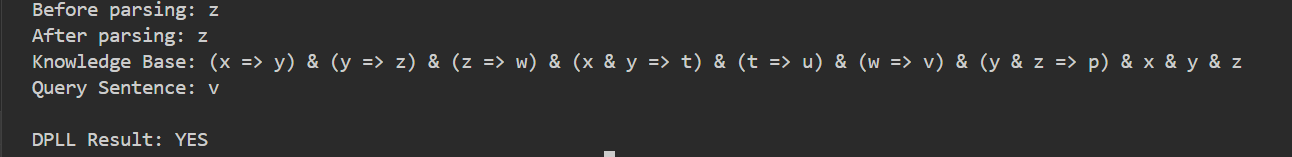
Description automatically generated

A black screen with white text

Description automatically generated

A black background with white text

Description automatically generated



CONCLUSION

|  |  |  |  |
| --- | --- | --- | --- |
| Methods | Advantages | Disadvantages | Demonstration |
| Forward Chaining | Applies rules progressively using facts x, y, and z to derive v. | Can explore unnecessary facts or rules that do not contribute to the goal, leading to inefficiency. | Efficient when the goal is already in the knowledge base. |
| Backward Chaining | Traces backward from v, exploring the rules and facts needed to prove it. | May explore many paths or branches, leading to inefficiencies in complex scenarios. | Directly useful for querying known facts. |
| Truth Table | Exhaustively checks all possible truth assignments for symbols to derive v. | Computationally expensive, especially for larger KBs, due to exponential time complexity. | Exact but inefficient for problems with many variables. |
| DPLL | Efficiently solves using unit propagation and pure symbol elimination. | Requires transformation into CNF, which adds complexity, and might be overkill for smaller problems. | Efficient compared to Truth Table. |

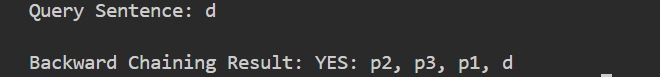
1. Cyclic Dependencies with Multiple Pathways:

A computer screen with white text

Description automatically generatedA black background with white text

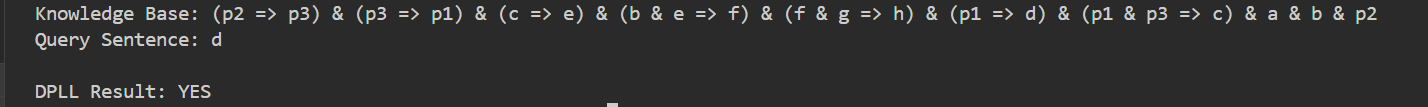
Description automatically generated

A black screen with white text

Description automatically generated

A black background with white text

Description automatically generated



CONCLUSION

|  |  |  |  |
| --- | --- | --- | --- |
| Methods | Advantages | Disadvantages | Demonstration |
| Forward Chaining | **Forward Chaining** applies facts **a**, **b**, and **p2** to generate intermediate conclusions (like **c** and **f**) and ultimately concludes **d**. | In cases of **cyclic dependencies**, can lead to unnecessary explorations. May visit facts multiple **times**, especially when rules are applied repeatedly. | Efficient when the goal is already in the knowledge base. |
| Backward Chaining | Starts from **d** and works backwards through rules like **p1 => d**, **p1 & p3 => c**, and uses available facts (**a**, **b**, **p2**) to eventually conclude **d**. | Can be **inefficient in large KBs** or with many possible paths. **Recurse excessively** in the case of circular dependencies. | **Goal-oriented** approach that explores relevant conditions. |
| Truth Table | Exhaustively checks every combination of truth values for all variables and finds that **d** is true, ensuring completeness. | **Scalability issue**: rapidly becomes inefficient as the number of symbols grows. | **Effective for small-to-medium-sized problems** with clear relationships. |
| DPLL | Efficiently solves the problem by converting the sentences to CNF, performing unit propagation, and eliminating pure symbols, ultimately concluding **d**. | Requires **conversion to CNF**. **More complex** for small, straightforward problems. Might be overkill for simpler cases. | **Optimized approach** with **unit propagation** and **pure symbol elimination**. |

1. Simple Fact and Rule Application:

A computer screen shot of a black screen

Description automatically generatedA computer screen shot of a computer program

Description automatically generated

A black and white screen with white text

Description automatically generatedA screen shot of a computer program

Description automatically generated

CONCLUSION

|  |  |  |  |
| --- | --- | --- | --- |
| Methods | Advantages | Disadvantages | Demonstration |
| Forward Chaining | Starts with the fact a, then applies the rule a => b, concluding b. | Can be inefficient with complex KBs or when multiple steps are needed. | Efficient when the goal is already in the knowledge base. |
| Backward Chaining | Starts with the goal **b** and works backward, finding **a** and applying the rule **a => b** to conclude **b**. | Can struggle with indirect connections or when the goal requires multiple steps. | **Goal-oriented** approach that explores relevant conditions. |
| Truth Table | Checks all possible combinations of truth values for **a** and **b**, concluding that **b** is true when **a** is true. | Computationally expensive for large KBs. Overkill for simple problems with few symbols. | **Effective for small-to-medium-sized problems** with clear relationships. |
| DPLL | Efficiently solves by converting to CNF and using unit propagation to conclude **b** from **a**. | Requires **conversion to CNF**. **More complex** for small, straightforward problems. Might be overkill for simpler cases. | Optimized approach for satisfiability checking. Efficient with larger KBs. |

1. Test Horn Knowledge Base:

A black background with white text

Description automatically generatedA black background with white text

Description automatically generated

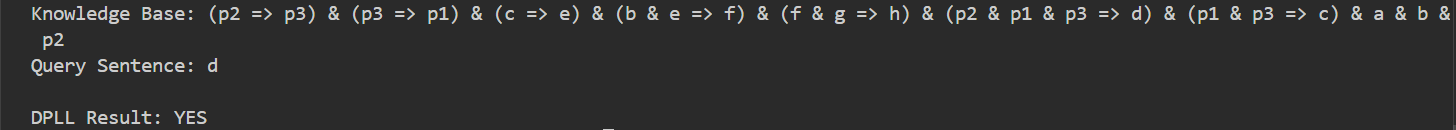
A computer screen with text and symbols

Description automatically generated with medium confidenceA black background with white text

Description automatically generated

A black background with white text

Description automatically generated



CONCLUSION

|  |  |  |  |
| --- | --- | --- | --- |
| Methods | Advantages | Disadvantages | Demonstration |
| Forward Chaining | Starts with facts **a**, **b**, and **p2**. It applies rules to eventually deduce **d** using **p2 & p1 & p3**. | Can be inefficient with complex KBs or when multiple steps are needed. | Efficient when the goal is already in the knowledge base. |
| Backward Chaining | Works backward from goal **d**, finding the rule **p2 & p1 & p3 => d**, and traces the premises, which are true based on the facts **a**, **b**, and **p2**. | Can struggle with indirect connections or when the goal requires multiple steps. | **Goal-oriented** approach that explores relevant conditions. |
| Truth Table | Exhaustively checks all combinations of truth values for **p2**, **p3**, **p1**, and others, confirming that **d** can be true. | Computationally expensive for large KBs. Overkill for simple problems with few symbols. | **Effective for small-to-medium-sized problems** with clear relationships. |
| DPLL | Simplifies the KB into CNF, applies unit propagation and pure symbol elimination, efficiently concluding **d** from available facts and rules. | Requires **conversion to CNF**. **More complex** for small, straightforward problems. Might be overkill for simpler cases. | Optimized approach for satisfiability checking. Efficient with larger KBs. |

1. Test Generic Knowledge Base:

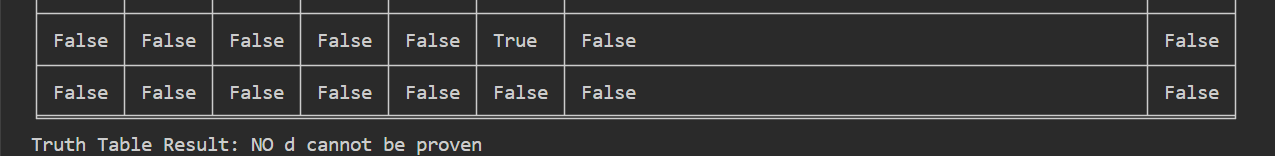
A computer screen shot of a black background

Description automatically generated

**A computer screen shot of a black background

Description automatically generatedA black background with white text

Description automatically generated**

****

**A black background with white and black text

Description automatically generated**

CONCLUSION

|  |  |  |  |
| --- | --- | --- | --- |
| Methods | Advantages | Disadvantages | Demonstration |
| Forward Chaining | Starts with facts **c** and ~f | It applies the rules, including the equivalence **a <=> (c => ~d)**, eventually deducing **d**. |  |
| Backward Chaining | Starts with the goal **d** and traces backward, using the equivalence **a <=> (c => ~d)**, which leads to the facts **c** and ~f |  | **Goal-oriented** approach that explores relevant conditions. |
| Truth Table | Exhaustively checks all combinations of truth values for symbols **a**, **b**, **c**, **d**, **f**, **g**, and others, verifying that **d** can be true. | Computationally expensive for large KBs. Overkill for simple problems with few symbols. | **Effective for small-to-medium-sized problems** with clear relationships. |
| DPLL | Converts the formula to CNF and efficiently applies unit propagation and pure symbol elimination to conclude **d** from available facts and rules. | Requires **conversion to CNF**. **More complex** for small, straightforward problems. Might be overkill for simpler cases. | Optimized approach for satisfiability checking. Efficient with larger KBs. |

1. Test Generic Knowledge Base 1:

A computer code with colorful text

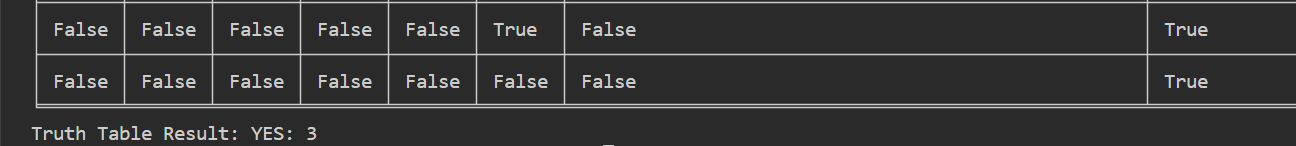
Description automatically generated with medium confidence A black background with white text

Description automatically generated

A computer screen shot of a black background

Description automatically generated A black background with white text

Description automatically generated



A black background with white text

Description automatically generated

CONCLUSION

|  |  |  |  |
| --- | --- | --- | --- |
| Methods | Advantages | Disadvantages | Demonstration |
| Forward Chaining |  | **Can explore irrelevant facts**, resulting in unnecessary computations. May not be efficient if too many facts are involved. | Efficient for fact-driven problems. Incremental reasoning based on facts. |
| Backward Chaining |  | **May require exploring many paths** depending on the complexity of the rules. | **Goal-oriented** approach that explores relevant conditions. |
| Truth Table | Exhaustively checks all possible combinations of truth values for **a**, **b**, **c**, **d**, **f**, **g**, and others to check if the query holds. This guarantees correctness. | Computationally expensive for large KBs. Overkill for simple problems with few symbols. | **Effective for small-to-medium-sized problems** with clear relationships. |
| DPLL | elimination to conclude **d** from available facts and rules. | Requires **conversion to CNF**. **More complex** for small, straightforward problems. Might be overkill for simpler cases. | Optimized approach for satisfiability checking. Efficient with larger KBs. |

1. Test Generic Knowledge Base 2:

A screen shot of a computer code

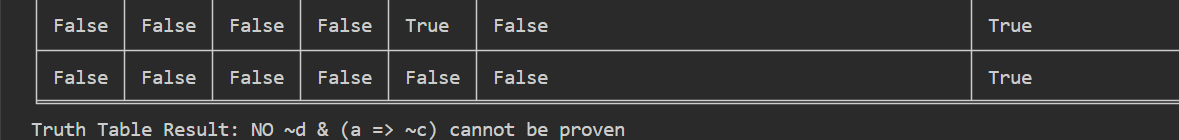
Description automatically generatedA black background with white text

Description automatically generated

A black screen with white text

Description automatically generatedA black background with white text

Description automatically generated



A black background with white symbols

Description automatically generated

CONCLUSION

|  |  |  |  |
| --- | --- | --- | --- |
| Methods | Advantages | Disadvantages | Demonstration |
| Forward Chaining |  | **Can explore irrelevant facts**, resulting in unnecessary computations. May not be efficient if too many facts are involved. | Efficient for fact-driven problems. Incremental reasoning based on facts. |
| Backward Chaining |  | **May require exploring many paths** depending on the complexity of the rules. | **Goal-oriented** approach that explores relevant conditions. |
| Truth Table |  | Computationally expensive for large KBs. Overkill for simple problems with few symbols. | **Effective for small-to-medium-sized problems** with clear relationships. |
| DPLL |  | Requires **conversion to CNF**. **More complex** for small, straightforward problems. Might be overkill for simpler cases. | Optimized approach for satisfiability checking. Efficient with larger KBs. |

1. Test Generic Knowledge Base 3:

**A screen shot of a computer

Description automatically generated** **A black background with white text

Description automatically generated**

**A computer screen shot of a black screen

Description automatically generated** **A black background with white text

Description automatically generated**

**A black screen with white text

Description automatically generated**

**A black background with white text

Description automatically generated**

CONCLUSION

|  |  |  |  |
| --- | --- | --- | --- |
| Methods | Advantages | Disadvantages | Demonstration |
| Forward Chaining |  | Limited with negations and conditional chains, failing to reach c due to the structure of the KB | Efficient for fact-driven problems. Incremental reasoning based on facts. |
| Backward Chaining |  | **May require exploring many paths** depending on the complexity of the rules. | Goal-directed reasoning that only evaluates conditions leading to c. |
| Truth Table | A black and white text  Description automatically generated | Computationally expensive for large KBs. Overkill for simple problems with few symbols. | **Effective for small-to-medium-sized problems** with clear relationships. |
| DPLL | A black background with white text  Description automatically generated | Requires **conversion to CNF**. **More complex** for small, straightforward problems. Might be overkill for simpler cases. | Optimized approach for satisfiability checking. Efficient with larger KBs. |

1. DPLL test:

A screen shot of a computer

Description automatically generated A black background with white text

Description automatically generated

A screen shot of a computer

Description automatically generated A black screen with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

A black background with white text

Description automatically generated

CONCLUSION

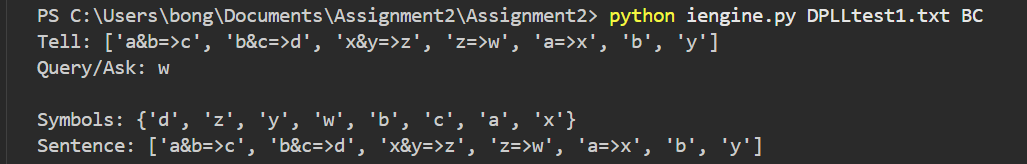
|  |  |  |  |
| --- | --- | --- | --- |
| Methods | Advantages | Disadvantages | Demonstration |
| Forward Chaining |  | **Can explore irrelevant facts**, resulting in unnecessary computations. May not be efficient if too many facts are involved. | Efficient for fact-driven problems. Incremental reasoning based on facts. |
| Backward Chaining |  | **May require exploring many paths** depending on the complexity of the rules. | Goal-directed, examining only the conditions relevant to deriving d. |
| Truth Table |  | **Computationally expensive** for larger sets of variables, though efficient for this small KB. | **Effective for small-to-medium-sized problems** with clear relationships. |
| DPLL |  | Requires **conversion to CNF**. **More complex** for small, straightforward problems. Might be overkill for simpler cases. | Optimized approach for satisfiability checking. Efficient with larger KBs. |

1. DPLL test 1:

**A computer screen shot of math equations

Description automatically generated** A black background with white text

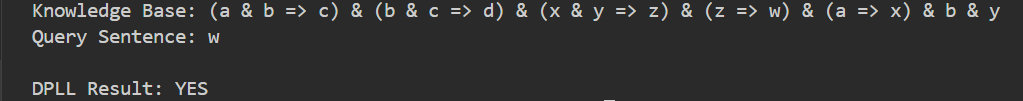
Description automatically generated

**** A black background with white text

Description automatically generated

**A black screen with white text

Description automatically generated**

****

CONCLUSION

|  |  |  |  |
| --- | --- | --- | --- |
| Methods | Advantages | Disadvantages | Demonstration |
| Forward Chaining |  | **Can explore irrelevant facts**, resulting in unnecessary computations. May not be efficient if too many facts are involved. | Efficient in exploring fact-driven chains, reaching w by applying each relevant rule. |
| Backward Chaining |  | **May require exploring many paths** depending on the complexity of the rules. | Goal-directed approach, only checking necessary conditions for w. |
| Truth Table |  | Computationally expensive for large KBs. Overkill for simple problems with few symbols. | **Effective for small-to-medium-sized problems** with clear relationships. |
| DPLL |  | Requires **conversion to CNF**. **More complex** for small, straightforward problems. Might be overkill for simpler cases. | Efficient, quickly infers w using unit propagation and logical simplification. |

**C. Test report:**

|  |  |
| --- | --- |
| **Algorithm** | **Recommendation** |
| **Forward Chaining** | Suitable for fact-driven reasoning with straightforward rules. Efficient when inferring new facts from initial conditions. However, it may explore many irrelevant paths in larger or complex knowledge bases. Best for cases where facts naturally lead to the query without extensive branching. |
| **Backward Chaining** | Recommended for goal-driven tasks where the query is highly specific. It efficiently works backward from the goal, examining only necessary conditions. However, in knowledge bases with many interdependencies, it may explore redundant paths. Best used for targeted, goal-oriented reasoning. |
| **Truth Table** | Best for exhaustive verification and complete certainty, particularly in smaller knowledge bases. It evaluates all possible truth assignments, guaranteeing correctness, but becomes computationally impractical with large or complex knowledge bases. Use for problems requiring thorough and exhaustive checks. |
| **DPLL** | The preferred choice for complex knowledge bases with numerous dependencies. It uses efficient strategies like unit propagation and pure symbol elimination to solve queries without unnecessary evaluations. Especially effective in larger, conjunctive normal form (CNF) knowledge bases. Recommended for situations needing efficient, logically rigorous solutions. |

# Features/Bugs/Missing

Because this assignment I work alone so maybe some of code still missing or does not work properly due to huge amount of work load and time pressure.

**1. Implemented Features:**

* Truth Table: For a specific knowledge base and query, the system is able to produce an entire truth table. Complex logical expressions can be handled by it, and it accurately compares them to every potential model.
* Forward Chaining: Forward chaining inference is a capability of the system. It can correctly ascertain whether a question is covered by the knowledge base and efficiently infers new facts from the information provided in the knowledge base.
* Backward Chaining: An inference mechanism for backward chaining is also included in the system. It essentially searches the knowledge base for any evidence that supports the goal by starting with the goal and working backward.

**2. Features Not Implemented:** The evaluation's specified algorithms (FC, BC, TT, DPLL) have all been implemented successfully.

**3. Bugs:**

* **Logical Expression Handling**
  + Errors in parsing or evaluating logical expressions can lead to incorrect results. This typically occurs due to mistakes in mapping logical variables and operators. Ensuring that logical expressions are correctly interpreted and processed is essential.
* **Input/Output Data Management**
  + Bugs in reading or writing data from files or other sources can result in incomplete or inaccurate data being fed into the system. This can cause issues in subsequent processing and logical inference. Proper error handling and validation of input data are critical.
* **Recursive Processing**
  + Algorithms that use recursion, such as backward chaining, can encounter stack overflow issues if not implemented and monitored carefully. Properly managing recursive calls is necessary to prevent these errors.
* **Memory Management**
  + In algorithms like DPLL, effective memory management is crucial. Poor memory management can lead to memory overflows and impact the overall performance of the system. Ensuring efficient use of memory resources is essential to maintain system stability.

**4. Missing:**

* **Limitations of the DPLL Algorithm**
  + Output Restriction: The implementation of the Davis-Putnam-Logemann-Loveland (DPLL) method only produces a yes or no response for satisfiability. It lacks thorough justifications and counterexamples, which could be helpful in determining why a proposition is unsatisfactory.
  + Propositional Limitations: If a propositional variable is not defined in the code, the algorithm might not support it. The ability to handle novel or dynamically produced propositions from input files is hence limited.
* **Managing Input and Output** 
  + Error Handling: For input files that are faulty, the Reader.py and Parser.py modules may not have strong error handling. When consumers enter inaccurate or incomplete data, they might not receive any explicit feedback or error notifications.
  + Flexible Data Formats: The system's adaptability to multiple applications or datasets may be limited if it does not accept data formats other than the one for which it was created.

# Research

**A. DPLL Algorithm:**

* **Overview of the DPLL Algorithm:**
  + A key approach in computer science for resolving the satisfiability issue of propositional logic formulas is the Davis-Putnam-Logemann-Loveland (DPLL) algorithm. Finding an interpretation that satisfies a given Boolean formula is the goal of this problem, which is sometimes referred to as SAT. The formula is considered satisfiable if such an interpretation is possible; if not, it is considered unsatisfiable. Through the use of strategies like backtracking, unit propagation, and pure literal elimination, the DPLL method improves the fundamental SAT-solving process.
* **DPLL's Objective:**
  + Determining whether propositional logic formulations are satisfiable is the main goal of the DPLL algorithm. It is employed in many different applications, such as:
  + Automated Theorem Proving: By assessing whether a logical theorem's negations are satisfiable, DPLL is utilized to prove it.
  + Artificial Intelligence: In AI, reasoning systems employ DPLL to draw conclusions from a collection of facts and logical principles.
  + Verification: To make sure that systems operate as intended under all circumstances, DPLL is used in hardware and software verification.
  + Planning: DPLL can be utilized in automated planning to identify action sequences that result in desired outcomes.
* **Overview of the Code Functionality:**
  + The class methods work together to convert logical sentences into CNF, simplify the CNF using heuristics like pure symbols and unit clauses, and apply recursive satisfiability testing.

1. Class DPLL

* This class encapsulates the logic for using the DPLL algorithm to determine if a given query follows logically from a knowledge base.

2. Key Methods in the DPLL Class

* \_\_init\_\_: Initializes the DPLL instance with a knowledge base and a query, storing the union of all unique symbols in both for later processing.
* to\_cnf: Converts a given logical sentence into Conjunctive Normal Form (CNF), a format required by the DPLL algorithm.
  + This method recursively transforms logical components (e.g., Symbol, Negation, Conjunction, Implication, Disjunction) into CNF.
  + This transformation is necessary because DPLL can only operate on clauses in CNF.
* find\_pure\_symbol: Identifies pure symbols within the CNF clauses.
  + A pure symbol appears with only one polarity (either always true or always false) across all clauses. Pure symbols can be assigned values that satisfy the clauses they appear in, potentially simplifying the satisfiability check.
  + This method iterates through symbols and clauses to find such pure symbols, returning a tuple of the symbol and its preferred polarity if one exists.
* find\_unit\_clause: Finds unit clauses in the CNF clauses.
  + A unit clause is one with only a single literal that has not yet been assigned a truth value, allowing DPLL to simplify by directly assigning a satisfying truth value.
  + This method returns the unassigned literal’s symbol and value if a unit clause is found.
* dpll\_satisfiable: Core DPLL algorithm implementation.
  + This recursive function checks the satisfiability of the given CNF clauses under current symbol assignments.
  + Key steps in this method:
    - Checks for base cases: if all clauses are satisfied (returns current assignment) or if an empty clause is found (indicating unsatisfiability).
    - Unit Propagation: Attempts to simplify the problem by applying any unit clauses found.
    - Pure Symbol Elimination: Assigns values to pure symbols to satisfy related clauses.
    - Symbol Selection: Chooses a symbol to assign True or False, attempting each recursively to see if a satisfying assignment can be found.
* solve: Initiates the DPLL algorithm on the knowledge base and query.
  + This method first combines the knowledge base (KB) with the negation of the query (¬query) to form KB ∧ ¬query and converts it into CNF.
  + If dpll\_satisfiable shows that KB ∧ ¬query is unsatisfiable, this implies the knowledge base entails the query (KB ⊨ query).
  + Returns "YES" if entailment is proven, otherwise returns "NO".

All the tests done with the DPLL algorithms were demonstrated earlier.

# Conclusion

In this report, we analyzed four main inference engine algorithms—Forward Chaining, Backward Chaining, Truth Tables, and the DPLL (Davis–Putnam–Logemann–Loveland) algorithm—each of which addresses different aspects of **propositional logic reasoning**. Each algorithm has distinct strengths and limitations, making them suitable for specific types of logical inference tasks. Understanding these differences is essential to selecting the best approach for a given problem, especially as logic-based reasoning becomes increasingly relevant across areas like artificial intelligence, expert systems, and complex decision-making.

**Forward Chaining** is a data-driven approach that derives conclusions by iteratively applying known facts to infer new information. This algorithm is particularly effective for cases where we begin with a broad base of knowledge and wish to explore its implications fully. Its main advantage lies in its efficiency in generating all possible outcomes from a given knowledge base, making it ideal for scenarios where all possible inferences need to be uncovered. However, this advantage is also its limitation: Forward Chaining does not prioritize reaching a specific goal and may, therefore, end up processing many irrelevant facts, especially in knowledge bases with unrelated or loosely connected information.

In contrast, **Backward Chaining** takes a goal-driven approach, starting from a query and working backward through the knowledge base to see if existing facts can support the target inference. This algorithm is highly effective for goal-oriented tasks, as it narrows down the search to relevant conditions only. Consequently, Backward Chaining is well-suited for applications like diagnostic systems or problem-solving tasks where we need to determine if a specific conclusion is true based on available information. However, this method can be inefficient for complex or highly interconnected knowledge bases, as it may require exploring multiple branches or paths that do not yield immediate answers, potentially increasing computational costs.

**Truth Tables** represent a different approach by systematically evaluating all possible combinations of truth values for each symbol in the knowledge base, guaranteeing a correct result if a solution exists. Truth Tables provide a thorough and exhaustive method for assessing logical consistency and deriving valid inferences. However, they are computationally prohibitive for larger problems due to the exponential growth of possible truth assignments as the number of symbols increases. Therefore, while Truth Tables are valuable for smaller knowledge bases or as a reference for validating other methods, their practicality decreases significantly with problem size.

Finally, the **DPLL algorithm** offers a sophisticated approach by leveraging unit propagation, pure symbol elimination, and recursive search to efficiently determine satisfiability. This method balances thoroughness with computational efficiency and effectively reduces the search space, making it particularly well-suited for larger or more complex logic problems, especially those represented in CNF (Conjunctive Normal Form). DPLL’s advantages include its ability to handle satisfiability checks without exhaustive enumeration, as it identifies early simplifications to streamline the process. However, the need to convert the knowledge base to CNF adds complexity to DPLL’s implementation. DPLL is, therefore, recommended for cases where the logical problem can be represented in CNF and efficiency is paramount.

In conclusion, each inference engine algorithm has unique strengths tailored to different types of logical inference tasks. For large and complex knowledge bases, **DPLL** is the preferred choice due to its computational efficiency and ability to prune the search space effectively. For tasks focused on specific queries or goals, **Backward Chaining** provides an efficient, goal-driven approach, especially in knowledge bases with relevant, goal-specific information. When all possible inferences from a set of facts are needed, **Forward Chaining** excels by providing comprehensive fact-driven reasoning. **Truth Tables**, while computationally demanding for large problems, offer a straightforward, exhaustive approach suitable for smaller problems. By selecting an algorithm based on the problem’s size, complexity, and objectives, we can ensure effective and efficient logical inference tailored to specific reasoning tasks.

# References

*Backward and forward chaining algorithm for (expert system) in Python*. (n.d.). Stack Overflow. https://stackoverflow.com/questions/43028336/backward-and-forward-chaining-algorithm-for-expert-system-in-python

*DPLL algorithm*. (2019, December 20). Wikipedia. https://en.wikipedia.org/wiki/DPLL\_algorithm

*Forward Chaining and Backward Chaining inference in Rule-Based Systems*. (2024, June 17). GeeksforGeeks. https://www.geeksforgeeks.org/forward-chaining-and-backward-chaining-inference-in-rule-based-systems/

*Forward chaining in AI with FOL proof*. (2022, October 6). GeeksforGeeks. https://www.geeksforgeeks.org/forward-chaining-in-ai-with-fol-proof/

Neutralise. (2011, April 27). *DPLL algorithm definition*. Stack Overflow. https://stackoverflow.com/questions/5811635/dpll-algorithm-definition

say786. (2015, April 9). *Creating a truth table for any expression in Python*. Stack Overflow. https://stackoverflow.com/questions/29548744/creating-a-truth-table-for-any-expression-in-python

Terrazzoni, J.-B. (2022, December 30). *Expert Systems: How to implement a backward chaining resolver in Python*. 42 Stories. https://medium.com/a-42-journey/expert-systems-how-to-implement-a-backward-chaining-resolver-in-python-bf7d8924f72f

*What Is An Inference Engine in Machine Learning?* (n.d.). Www.run.ai. https://www.run.ai/guides/machine-learning-inference/inference-engine

*What is an Inference Engine? Types and Functions*. (2024, April 30). GeeksforGeeks. https://www.geeksforgeeks.org/what-is-an-inference-engine-types-and-functions/

*Write a the fastest recursive Python function dpll sat solve that takes two arguments clause setand partial - brainly.com*. (2024). Brainly.com. https://brainly.com/question/49664739