

|  |  |
| --- | --- |
| Assignment 2 – Inference Engine | |
|  |  |
| Mihir Bhadauria (103075328)Kingsley Brodie (102147941) Group: **COS30019\_A02\_T022** |  |

Contents

[Assignment 2 – Inference Engine 1](#_Toc135946965)

[Mihir Bhadauria (103075328) 1](#_Toc135946966)

[Kingsley Brodie (102147941) 1](#_Toc135946967)

[Instructions: 3](#_Toc135946968)

[Features/Bugs/Missing 3](#_Toc135946969)

[Data Extraction 3](#_Toc135946970)

[Command Line Implementation 4](#_Toc135946971)

[Truth Table implementation 5](#_Toc135946972)

[Bugs/Limitations 7](#_Toc135946973)

[Test Cases 8](#_Toc135946974)

[Acknowledgement/Resources 8](#_Toc135946975)

[Research 9](#_Toc135946976)

[Team Summary Report 9](#_Toc135946977)

# Instructions:

The program is developed in python and is stored in a zip file with all the necessary components.

To execute the following program in CMD, you need to enter in this format:

|  |
| --- |
| ..\user> main.exe <method> <filename> |

To break this down

* main.exe runs the program.
* <method> is the type of search you want to do and those are shown below.
* <filename> is the name of the input file which in this case is *“test\_HornKB.txt”*

|  |  |
| --- | --- |
| Truth Table | main.exe TT <filename> |
| Forward Chaining | main.exe FC <filename> |
| Backward Chaining | main.exe BC <filename> |

*Table 1: Command line program execution instruction*

The <method> is not case sensitive however if any issues occur type the <method> shown above in lowercase.

# Features/Bugs/Missing

## Data Extraction

In order for the program to work we first need to gather data. Data is given to us from a txt file like *test\_HornKB.txt.* We have implemented a similar manner to the previous assignment done by us in getting contents extracted. The function read\_file\_data(data) does this.

In the screenshot you can see that the Knowledge Base (KB) and query (q) are stored in variables once taken from the file and then KB goes further transformation to sanitize the data.

## 

## Command Line Implementation

A screen shot of a computer program

Description automatically generated with medium confidence

As per the requirements of this assignment and the previous, the code must run through the command line. By using the “sys” library we can make this happen.

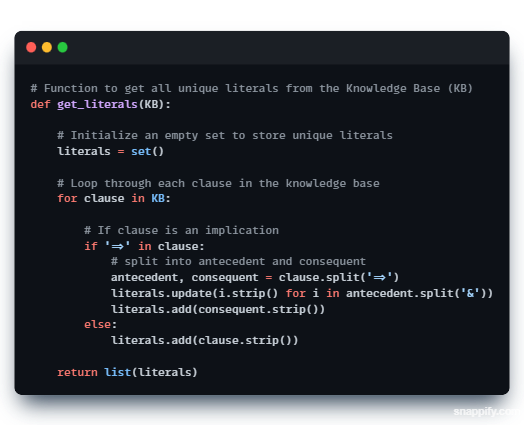
In the code if 3 arguments are not presented then an error will come up. From there the contents and the method of inference are stored in variables so that additional validation and which method to use can be determined.

## 

## Truth Table implementation

Truth tables are essential tools for evaluating the truth values of formulas, particularly in valid Horn clauses. They comprise of all possible symbol combinations in a knowledge base. By carefully examining the truth table, we can determine the truth value of a formula or query based on its alignment with the represented scenarios. This enables us to make well-informed judgments while considering the logical relationships in the knowledge base.

In our code the truth table is built by multiple functions:



Get\_literals(KB) as the name implies gets each literal value. It does this by creating an empty set to store literals and then going through each clause in the knowledge base which contains an implication and splits it to the Left-hand side (LHS/antecedent) and Right Hand Side (RHS/consequent) whilst eliminating white spaces then returns the list of literals.

A screen shot of a computer program

Description automatically generated with low confidence

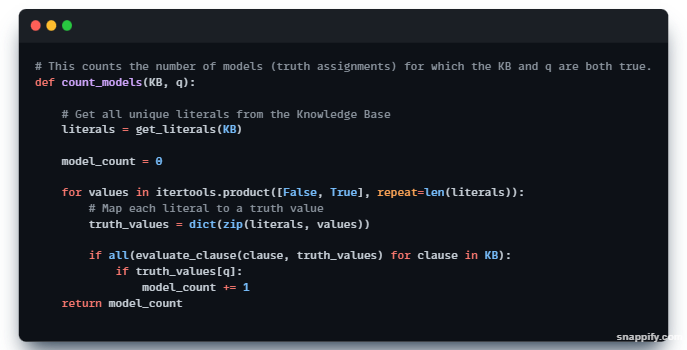
The evaluate\_clause(clause, truth\_values) function evaluates the truth value of a clause in the knowledge base. It handles implications by checking if the antecedent implies the consequent based on provided truth values. For conjunctions, it checks if all literals are true. The function retrieves truth values from a dictionary and returns the resulting truth value of the clause.

A screen shot of a computer program

Description automatically generated with low confidence

The evaluate\_truth\_table(KB, q) function checks all possible combinations of truth values for the literals in the knowledge base by iterating through all combinations using the itertools.product function and evaluates each combination by calling evaluate\_clause for each clause in the knowledge base.

If all clauses evaluate to true for a particular combination, it adds the truth values to a list of true models. Finally, it checks if the query q is true in all the true models and returns the result.

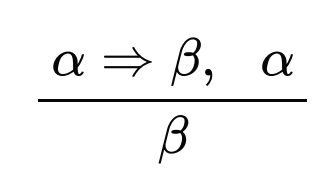


count\_models(KB, q) counts the number of models where both KB and q are true. It iterates through all possible truth value combinations for the literals, checks if the clauses in KB evaluate to true for each combination, and increments the model\_count if q is also true. Finally, it returns the count of models.

## Forward Chaining

Forward Chaining is an inference algorithm that takes a knowledge base in horn form and a query q and returns whether the knowledge base entails q.

This entailment is found by inference checking using modus ponens.



*Artificial Intelligence: A modern approach 4th edition page 441*

The rule states that if a sentence is in the form of an implication and alpha is given then beta can be inferred. With this logic we can create a forward chaining algorithm.

A screen shot of a computer program

Description automatically generated with low confidence

Forward\_chaining accepts two arguments, a knowledge base, and a query. Firstly, we find all propositions which are given and add them to a queue.

A screen shot of a computer program

Description automatically generated with low confidence

Next, we create a dictionary with the keys being each symbol and a number associated with how many symbols infer that symbol.

A screen shot of a computer program

Description automatically generated with medium confidence

Finally, we iterate over the inferred symbols in the queue popping a symbol off the stack, we then check if that symbol equals q else we iterate over the knowledge base checking if the true symbol implies another symbol, if it does, we decrement that symbols count if a symbols count equals 0 we add it to the queue. The function is excited if either a true symbol equals q or there are no symbols in the stack, and it returns false.

Forward Chaining is an efficient algorithm which has a runtime of O(n) considerably more efficient than truth table generator algorithms.

## Backward Chaining



In backward chaining the idea is that we have a query q and a knowledge base in horn form, and we want to see if q is inferred from the modus ponens rule by working backwards to see if the symbols that infer q are true.

Firstly, we create a dictionary with symbols associated with true or false value.

We then run a recursive function working recursively calling the function with the symbol that implies q to return q’s truth value.

We then check the truth value of q and print the results.

Backward Chaining is also O(n) runtime complexity making it an efficient inference algorithm.

## Bugs/Limitations

The program crashes when a general knowledge base is inputted.

The current program has some limitations that should be addressed:

* **Lack of Input Validation:** The program assumes that the input is always in a specific format as follows:

A picture containing text, font, algebra, screenshot

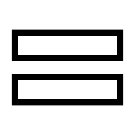
Description automatically generated

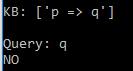
* Solution: To mitigate this issue, we propose two potential solutions:
  + Implement robust input validation methods within the program to ensure the correctness of the input data structure.
  + In addition, or alternatively, we can prepare detailed documentation outlining the required input format, thus guiding the user to provide the data points in the correct sequence.

These improvements will enhance the program's robustness and usability.

# Test Cases

To ensure this program is robust and reliable we will test it in a variety of situations to ensure it works.

1. A picture containing text, font, white, screenshot

   Description automatically generatedSingle literal Knowledge Base with single literal query:
2. ![A picture containing black, darkness

   Description automatically generated](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAIcAAACHCAMAAAALObo4AAAAAXNSR0IArs4c6QAAAARnQU1BAACxjwv8YQUAAAAYUExURQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFY3HCoAAAAHdFJOUwAKCxwdYGHSVhpCAAAACXBIWXMAACHVAAAh1QEEnLSdAAAApklEQVR4Xu3VK5LDQBBEQWm9n/vfeEFNgTYxsSYUjkzWqB7rAwAAALjE+bPXuXafPf72+l67z3RMOqabdazrShnSURnSURnSURnSURnSURnSURnSURnSURnSURl60bGNjknHdPcOAPhk/v6kY9IxrY51XSlDOipDOipDOipDOipDOipDOipDOipDOipDOipDLzq20THpmO7ecf7u9bV2AQAAgHc6jn/snVvNteT84gAAAABJRU5ErkJggg==)A black text on a white background

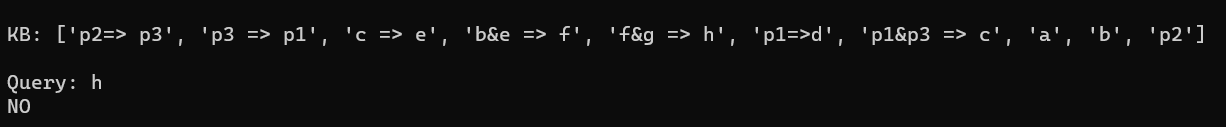
   Description automatically generated with low confidenceSingle clause Knowledge Base with single literal query:
3. Single clause Knowledge Base with conjunction query:
4. Testing with a false query, expected NO:

A picture containing screenshot, font

Description automatically generated

![A picture containing black, darkness

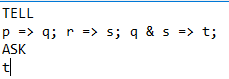
Description automatically generated](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAIcAAACHCAMAAAALObo4AAAAAXNSR0IArs4c6QAAAARnQU1BAACxjwv8YQUAAAAYUExURQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFY3HCoAAAAHdFJOUwAKCxwdYGHSVhpCAAAACXBIWXMAACHVAAAh1QEEnLSdAAAApklEQVR4Xu3VK5LDQBBEQWm9n/vfeEFNgTYxsSYUjkzWqB7rAwAAALjE+bPXuXafPf72+l67z3RMOqabdazrShnSURnSURnSURnSURnSURnSURnSURnSURnSURl60bGNjknHdPcOAPhk/v6kY9IxrY51XSlDOipDOipDOipDOipDOipDOipDOipDOipDOipDLzq20THpmO7ecf7u9bV2AQAAgHc6jn/snVvNteT84gAAAABJRU5ErkJggg==)



1. Multiple clause Knowledge Base with single literal query:

A picture containing text, font, screenshot, black

Description automatically generated![A picture containing black, darkness

Description automatically generated](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAIcAAACHCAMAAAALObo4AAAAAXNSR0IArs4c6QAAAARnQU1BAACxjwv8YQUAAAAYUExURQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFY3HCoAAAAHdFJOUwAKCxwdYGHSVhpCAAAACXBIWXMAACHVAAAh1QEEnLSdAAAApklEQVR4Xu3VK5LDQBBEQWm9n/vfeEFNgTYxsSYUjkzWqB7rAwAAALjE+bPXuXafPf72+l67z3RMOqabdazrShnSURnSURnSURnSURnSURnSURnSURnSURnSURl60bGNjknHdPcOAPhk/v6kY9IxrY51XSlDOipDOipDOipDOipDOipDOipDOipDOipDOipDLzq20THpmO7ecf7u9bV2AQAAgHc6jn/snVvNteT84gAAAABJRU5ErkJggg==)

1. Multiple clause Knowledge Base with disjunction query:
2. Knowledge Base with negation, with single literal query:

# Acknowledgement/Resources

<https://www.freecodecamp.org/news/big-o-cheat-sheet-time-complexity-chart/#:~:text=You%20get%20linear%20time%20complexity,of%20order%20O(n)>.

* Big O notation to help with report.

<https://regexr.com/>

* To help with regex pattern creations

<https://www.w3schools.com/python/>

* Python syntax

<https://swinburne.instructure.com/courses/49155/pages/assignment-helper?module_item_id=3421270>

* Understanding of assignment

<https://www.geeksforgeeks.org/proposition-logic/>

<https://cs50.harvard.edu/ai/2020/notes/1/>

<http://aima.cs.berkeley.edu/python/logic.py>

Stuart Russell, Peter Norvig - Artificial Intelligence\_ A Modern Approach (4th Edition)

* Referenced the textbook for understanding of concepts and algorithms.
* Aside from lectures and tutorials this has aided me in developing a greater understanding of propositional logic and its fundamentals.
* These resources have also enabled me to understand how to go about coding this inference engine.

I have used (Mihir) the parser coding style from my assignment #1.

# Research

# Team Summary Report

**Contribution Matrix**

|  |  |
| --- | --- |
| Kinglsey Brodie (102147941) | Mihir Bhadauria (103075328) |
| * Forward Chaining Implementation * Backward Chaining Implementation * Report writing | * Input parser * Command Line Operation * Truth Table implementation * Report writing |
| **Percentage Contribution:** 50% | **Percentage Contribution:** 50% |

Mihir & Kingsley distributed the workload through tasks, dividing the work into functionality needing to be implemented into the program. Mihir took care of the infrastructure of the program using the imperative programming paradigm having a main program take input from the command line and then run each specific method, forward chaining, backward chaining, and the truth table generator.

Mihir was responsible for the truth table implementation whilst Kingsley was responsible for the forward and backward chaining implementation.

We used git and github for version control and a public repository which we both could access and Microsoft word for the report.