

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | |
|  | | Inference Engine | | | | |  | |
|  |  | | | | | | |  |
|  | | | |  |  | | | |
|  | | | | Mikkel Aguilar 103613812 |  | | | |
|  | | | | Due Date:May 27Subject CodeCOS30019 |  | | | |
|  | | |  | | |  | | |



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | | |  |  | | |  |
|  | Table of Contents | | | | | | |  |
|  |  | | |  |  | | |  |
|  |  | |  | | |  | |  |
|  |  |  | Instructions……………………………..3  Introduction…………………………….3  Features/Bugs/Missing…………….4  Methods/Test Cases..………………..5  Implementation……………………….5  Implementation(Cont.)…………….6  Implementation(Utilities)…………7  Implementation(Utilities Cont.)..8  Research Initiatives………………….9  Notes………………………………………9  Acknowledgements/Resources…10 | | |  |  |  |
|  | | | | |
|  |  |  |  |
|  |  |

##### The assignment is run on a DOS command line, written in the Python programming language. Here are the steps to using the program:

1. Navigate to the folder through folder through the command line
2. Input this format into the command line:
   * main.py
   * ‘BC’, ‘FC’, ‘TTC’. These are the methods the inference engine can use
   * ‘test\_HornKB.txt’ or any appropriate text file
3. Press ‘Enter’, and you will these results based on the method you chose:

Truth Table Checking – ‘YES : 3’ where 3 is the number of models left in the knowledge base.

Forward Chaining/Backward Chaining – ‘YES : a, b….’ where the items following the colon are the entailed symbols explored during the search.

1. User will be notified of invalid formats

Instructions

Introduction

##### The Inference Engine problem involves creating software that can evaluate propositional logic using three different algorithms – Truth Table Checking, Forward Chaining and Backward Chaining. The engine is compatible with text files that contain horn clauses, separated by semi-colons. As suggested by the assignment sheet, I also implemented Truth Table Checking for generic knowledge bases that contain three additional connectives. Glossary

Knowledge Base (KB) – Contains clauses that represent what an agent ‘knows’ at a certain point in time.

Clause – Found inside the KB, it contains propositional symbols and connectives that need to be proved.

Model – A dictionary used in TTC which contain a propositional symbol as a key and a Boolean value as value.

Cartesian Product – A set constructed by using the product of multiple other sets, where all possible combinations of size n is given.

Propositional Symbols – Symbols that represents a sentence about a world, such as P, Q, P1 etc.

Connectives – Used to form a connective proposition, can be in word form (AND) or a symbol (&).

##### Program can read data from a text file

##### Program can interpret and separate the clauses into a list form

##### Program can use find all possible models in a truth table using cartesian product

##### Program can simplify a clause, step by step, to find out whether the statement is true or false

##### Program can check if a clause is true given a model

##### Support for Truth Table Checking with a horn clause KB

##### Support for Backward Chaining with a horn clause KB

##### Support for Forward Chaining with a horn clause KB

##### Support for Truth Table Checking with general KB

##### Program can find all nodes passed during Forward Chaining

##### Program can find all nodes passed during Forward Chaining

##### Runs on the Command Line

##### Program prints whether a query is entailed in a knowledge base

##### Program knows when incorrect parameters have been given

##### Runs on the assumption that a text file follows a format

##### No known bugs have been found

##### Does not include a graphical representation

##### Does not show a list of nodes passed in Forward/Backward chaining if query is not entailed

##### Does not use Davis-Putnam-Logemann-Loveland algorithm

##### Does not use Resolution-Based theorem prover

Features/Bugs/Missing

##### TTC – Involves a table that finds the truth value for every possible combination of truth values given certain propositions.

FC – A method in which the engine goes through all the facts individually, in a certain, from a state (start) till it reaches its goal (end).

BC – A method in which the system starts from its goal and determines which facts need to be proven so that it reaches the state it needs to start from.

##### For the purpose of testing and debugging the program, I used three different text files, ‘test\_HornKB.txt’, ‘test.txt’, ‘test\_GenericKB.txt’. The first two are used to test Truth Table Checking, Forward Chaining and Backward chaining for knowledge bases that contain only horn clauses. However, the last is used to test Truth Table Checking for a knowledge base which contains general clauses.

Implementation – Methods

##### All three methods (TTC, FC, BC) takes in a query in the form of a propositional symbol (e.g., ‘p’). They all return either ‘YES’ or ‘NO’ followed by either number of models entailing the query, or the nodes passed when chaining.

##### Truth Table Checking:

##### A computer screen capture Description automatically generated with low confidence It involves using a list of models (dictionary), looping over all the models, and checking if any of those models cause the clause to become true. Once all the valid models have been found, another check will be done to ensure that the query is true in all those valid models. It returns a string ‘YES’ plus the number of models that entail those queries or ‘NO’ if there are no models that entail the query.

Methods/Test Cases

Implementation – Methods (Cont.)

##### Forward Chaining:

##### Text Description automatically generated

It creates a copy of the knowledge base and finds all the propositional symbols assumed to be true (clauses with a single symbol) and adds it to a list of symbols entailed. A while loop continues until the query is added to symbols entailed. The function loops through all the clauses in the KB and checks if the head of the clause it true, if it is, the tail of the clause will be added to a list of symbols found to be entailed. If there is no valid query, returns ‘NO’, otherwise it returns ‘YES’ and the list of symbols entailed.

##### Backward Chaining:

Text

Description automatically generated

Creates a copy of the knowledge base and keeps track of symbols that need to be proved, starting with the query. It loops through all the values to be proved and all the clauses, checking if the value can be found in the tail of a clause. It then checks if the head of clause contains logic that has already been proved, extends visited with the CP values and returns ‘YES’ plus the list of visited nodes. If after looping through all the values in CP and clauses, there is no valid next value to prove, the function returns ‘NO’

Implementation – Utilities

##### Create a Table of Possible Combinations:

##### Create a Truth Table:

A screenshot of a computer screen

Description automatically generated with medium confidence

##### Check if Single Statement is True/False:

Text

Description automatically generated

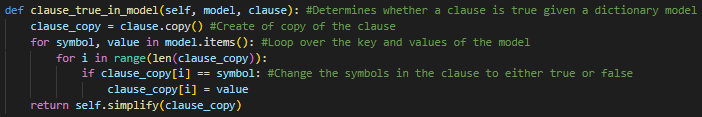
##### Simplify a Clause:

Text

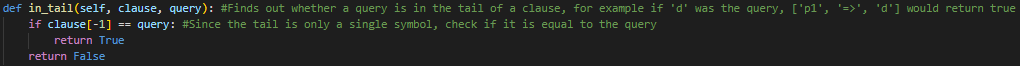
Description automatically generated

Implementation – Utilities (Cont.)

##### Check if Clause is True in the Model:



##### Check if Query is in the Clause’s Tail:



##### Check if the Clause’s head Values are Proved:

Text

Description automatically generated

##### Get Head Values of a Clause:

Text

Description automatically generated

##### Remove Duplicates from List:

Text

Description automatically generated

Research Initiative

##### The implementation was developed individually, no teams involved

##### You cannot use BC or FC for text files that contain a general propositional knowledge base, if you try, it will print out ‘NO’

##### The ‘test.txt’ file was a self-made for testing purposes

##### Documentation is in the python files

##### ‘Itertools’ and ‘sys’ were the only external libraries used in the project

##### Mainly Object-Oriented

Notes

##### The core aim of the project was to use methods to evaluate multiple horn clause statements. However, as an additional element to my program, I extended the code so that Truth Table Checking would work on general propositional knowledge bases.

##### Originally, the program only recognized and understood how to evaluate clauses that use Conditional (->) and Conjunction (&) connectives I was able to add functionality so that the engine would recognize three other connectives as well, which include Negation (~), Disjunction (||) and Biconditional (⬄). I also had to adjust and change the method in which I initially read the clauses from the file. Although the part that caused the most complications was adapting the ‘simplify’ method to consider the order of operations as the presence of open and closed parenthesis made it difficult.

I attempted to incorporate either the Resolution-Based Theorem Prover or the Davis-Putnam-Logemann-Loveland algorithms however found a lot of issues and could not successfully implement because of the way I coded my solutions.

Overall, this area of the task was not incredibly difficult ; it required extra time on research but relied more on adjusting, extending, and retesting already working functions to ensure they work with a general knowledge base.

Acknowledgements/Resources

##### <https://docs.python.org/3/library/itertools.html> – Documentation that helped me understand how to use cartesian product

<https://www.youtube.com/watch?v=EZJs6w2YFRM&ab_channel=FranciscoIacobelli> – Helped me visualize forward and backward chaining with a diagram

<https://web.stanford.edu/class/cs103/tools/truth-table-tool/> – Used to test by comparing my results to this truth table tool

<https://www.section.io/engineering-education/forward-and-backward-chaining-in-ai/#:~:text=Backward%20chaining%20is%20a%20concept,taken%20to%20attain%20this%20goal> – Used to further understand the concepts of backward and forward chaining

<https://docs.python.org/3/tutorial/inputoutput.html> – Contains documentation for reading data from a file

Canvas Resources – Contains the text files to test the implementation