Assignment 2 – Inference Engine

- **Due** *11:59pm Friday 24th May 2024* (End of Week 12)
- **Contributes** 20% to your final subject result, subject to moderation if required.
- **This is a group assignment – Group size of up to 2 students**. (*For students with special requirements, please contact the convenor for the option to do this assignment individually.*)

Summary

You need to implement an inference engine for propositional logic in software based on the Truth Table (TT) checking, and Backward Chaining (BC) and Forward Chaining (FC) algorithms. Your inference engine will take as arguments a Horn-form Knowledge Base **KB** and a query **q** which is a proposition symbol and determine whether **q** can be entailed from **KB**. You will also need to write a report about how your program works with different knowledge bases and queries.

Implementation

You are encouraged to implement your software using Python. If you prefer to implement your assignment in one of the alternative languages Java, C++ or C#, you need to discuss with your tutor to seek their permission. **You must gain permission from the convenor before using anything else. Assignment work will be tested on a standard Microsoft Windows 10 system.** (If you can, please try to test it on a Swinburne lab computer.)

TT, FC and BC Inference Engine

The Truth Table Checking (*TT*) algorithm works with all types of knowledge bases. The Forward Chaining (*FC*) and Backward Chaining (*BC*) algorithms work with Horn-form knowledge bases. (See the description in the textbook – the 3rd edition 2010, page 256-259.)

Given a knowledge base **KB** in Horn form (with TELL) and a query **q** which is a proposition symbol (with ASK), your program needs to answer whether or not **q** is entailed from **KB** using one of the three algorithms *TT*, or *FC*, or *BC*.

File Format: The problems are stored in simple text files consisting of both the knowledge base and the query:

- The knowledge base follows the keyword TELL and consists of Horn clauses separated by semicolons.
- The query follows the keyword ASK and consists of a proposition symbol.

For example, the following could be the content of one of the test files (`test1.txt`):

```
TELL
p2=> p3; p3 => p1; c => e; b&e => f; f&g => h; p2&p1&p3 =>d; p1&p3 => c; a; b;
p2;

ASK
d
```

Command Line Operation

Your program needs to operate from a DOS command-line interface to support batch testing. A DOS command-line interface can be brought up in Windows 7/8/10 by typing **cmd** into the search box at the ***Start*** button. However, please ensure that your program works on Windows 10 because we will be testing your program on Windows 10. This can be accomplished with a simple DOS .bat (batch) file if needed. Below are the three different arguments formats you need to support. Note the unique argument count for each.

```
C:\Assignments> iengine <filename> <method>
```

where **`iengine`** is your .exe file or a .bat (batch) file that calls your program with the parameters, **`filename`** is for the text file consisting of the problem, and **`method`** can be either `TT` (for Truth Table checking), or `FC` (for Forward Chaining), or `BC` (for Backward Chaining) to specify the algorithm.

(if you program in Python, we can accept Python scripts and execute your scripts using the following command from the CLI:

```
C:\Assignments> python iengine.py <filename> <method> )
```

For instance:
```
> iengine test1.txt FC
```

Standard output is an answer of the form YES or NO, depending on whether the ASK(ed) query **q** follows from the TELL(ed) knowledge base **KB**. When the method is *TT* and the answer is YES, it should be followed by a colon (:) and the number of models of **KB**. When the method is *FC* or *BC* and the answer is YES, it should be followed by a colon (:) and the list of propositional symbols entailed from **KB** that has been found during the execution of the specified algorithm.
For example, running **`iengine`** on the example `test1.txt` file with **`method`** TT should produce the following output:

```
> YES: 3
```

On the other hand, when I run my implementation of **`iengine`** with **`method`** FC on the example `test1.txt` it produces the following output:

```
> YES: a, b, p2, p3, p1, c, e, f, d
```
Note that your implementation might produce a different output and it would still be correct, depending on how you store and order the sentences in the knowledge base. I'll check that carefully to ensure that you won't be disadvantaged if your results don't look exactly the same as our results.

And running **`iengine`** with **`method`** BC on the example test file above should produce the following output:

```
> YES: p2, p3, p1, d
```

Assignment report

You must include a single report file (between 12 and 14 pages, either in Microsoft Word or in PDF) with your work with the following details:

- **Cover page:** Your full student names, IDs, and your group number (as allocated by ESP).
- **Table of contents (TOC)**.
- **Instructions:** Basic instructions of how to use your program. You can also include a **note** containing anything else you want to tell the marker, such as anything particular about your implementation.
- **Introduction:** Introduce the *Propositional Logic Inference Engine*, basic concepts and other related terminologies including logical connectives, truth table, models of a sentence, etc.
- **Inference methods:** Present and discuss the ideas behind each inference method implemented in your assignment.
- **Implementation:** Briefly present how each inference method was implemented. Class diagram and flow charts (pseudo code) are all suitable. Point out and briefly discuss differences in implementation or approach. Note important references.
- **Testing:** Provide an overview of the test cases you have created to test your program (either manually or automatically). Report the results of testing your program.
- **Features/Bugs/Missing:** Include a list of the features you have implemented. Clearly state if a required feature has not been implemented. Failure to do this will result in penalties. Include a list of any known bugs. Also, anything else you want to tell the marker, such as how to run other methods of inference that you have implemented as part of your research or anything particular about your implementation.
- **Research (if applicable):** If you show some initiatives in researching about the problem and solutions, or carrying out extensive & automated tests to provide interesting data about the algorithms, or getting some clever optimization, please include a Research section to report your initiative.
- **Team summary report (if applicable):** Summary of the teamwork in this assignment. You need to clearly indicate who did what and how each team member gave feedback to other members. In this report, the overall percentage of contribution by each student to the project has to be clearly specified and summed to 100%. This section of the report is only applicable to students who submit the assignment as a team.
- **Conclusion:** Conclude with a discussion about the best type of search algorithm you would use for this type of problem. Include thoughts about how you could improve performance.

- **Acknowledgements/Resources:** Include in your report a list of the resources you have used to create your work. A simple list of URL's is not enough. Include with each entry a basic description of how the person or website assisted you in your work.
- **References:** Cite the sources you referred to in your Assignment (implementation, report, etc.)

Tips:

- All figures and tables need to be properly captioned with sensible descriptions.
- Report presentation should include header/footer information (pages numbers, etc.)

Marking Scheme & Submission

You must submit your work via the online assignment collection system ESP https://esp.swin.edu.au/

Create a single zip file with your source code and a working version of your program and the report. Do not use deep folder hierarchies. Do not include the data files (we have them ☺). The upload limit to ESP will be 100 MB. Please consult early if you have a large binary/package for some.

Standard late penalties apply - 10% for each day late, more than 5 days late is 0%.

Up to 10% will be deducted for bad programming practices.

Up to 20% will be deducted for bad or no teamwork (if you work in a team of two students).

Up to 40% will be deducted for not showing the progress during the weeks leading up to the submission of the Assignment.

Marking Scheme

Requirements (or equivalent sections)		***Mark***
TT	Truth table method working to perfection	**25**
FC	Forward chaining method working to perfection	**20**
BC	Backward chaining method working to perfection	**20**
Testing	At least 16 test cases have been created to cover different problem scenarios. Test results have been checked and documented.	**10**
Report	Clear and provide sufficient information about your programs and your algorithm/solution, and about your teamwork (if applicable).	**10**
Research	You show some initiatives in researching about the problem and solutions, or carrying out extensive tests to provide interesting data about the algorithms, or getting some clever optimization, etc. and include a well-written section in the report to demonstrate these initiatives.	**15**
	Total	**100**
You need to follow good programming practice (e.g., well-designed, well-structure codes with clear and helpful comments). Failure to do so get penalty.		**Up to -10**
If you work in a team of two students, you need to demonstrate good teamwork in completing the Assignment. Note that joining someone in a team and then pulling out in the last two weeks will be marked as bad teamwork and incur a penalty.		**Up to -20**
You need to demonstrate the progress you make every week to your tutor. That is, if your tutor approaches you and asks for the progress, you have to be able to show the tutor the progress you have made in comparison to the previous week. Failure to do so get penalty.		**Up to -40**

One potential research idea is to allow your program to deal with general knowledge bases that don't have to be a Horn knowledge base. Then you can implement your Truth Table algorithm to deal with the general knowledge bases and also implement a generic theorem prover such as a resolution-based one.

If you want to implement inference engine for general knowledge bases, please use the following syntax for different logical connectives:

~ for negation ($\neg$)
& for conjunction ($\wedge$)
|| for disjunction ($\vee$)
=> for implication ($\Rightarrow$)
<=> for biconditional ($\Leftrightarrow$)

If you work in a group of two students, the quality of the report will be expected to be of very high quality with several research initiatives to demonstrate the substantial effort the team has collaborated on.