Assignment 2 – Inference Engine

- **Due** *4:30pm Friday 26th May 2017* (End of Week 12)
- **Contributes** 20% to your final subject result, subject to moderation if required.
- **This is a group assignment – Group size of 2 or 3 students**.

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 welcome to implement your software using Java or C++/C#. We recommend Java. You must gain permission before using anything else. Assignment work will be tested on a standard Microsoft Windows 7 system.

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; p1=>d; p1&p3 => c; a; b; p2;

ASK
d
```

Command Line Operation

Your program needs to operate in a simple command-line form to support batch testing. This can be accomplished with a simple DOS .bat (batch) file if needed. Below is an example of how your program will be run:

```
> iengine method filename
```

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

For instance:

```
> iengine FC test1.txt
```

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`** with **`method`** TT on the example `test1.txt` file 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, d
```

Note that your implementation might produce a different output and it would still be correct, depending on the order of 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 my results.

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

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

Readme.txt file

You must include a single readme.txt file with your work with the following details:

- **Student Details:** Your full student names, ids, and your group number (as allocated by ESP).
- **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.
- **Test cases:** The test cases you have developed ton test your program. What bugs have you found?
- **Acknowledgements/Resources:** Include in your readme.txt file 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.
- **Notes:** Anything else you want to tell the marker, such as how to use the GUI version of your program, and something particular about your implementation.
- **Summary report:** 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%.

Marking Scheme & Submission

You must submit your work via the online assignment collection system ESP https://esp.ict.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 10 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 20% will be deducted for bad or no teamwork.

Up to 50% 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	**30**
FC	Forward chaining method working to perfection	**25**
BC	Backward chaining method working to perfection	**25**
Readme file	Clear and provide sufficient information about your programs and your algorithm/solution, and about your teamwork.	**10**
Research	If 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. with a well-written 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 -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 -50**

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.