



Faculty of Science, Engineering and Technology

COS30019: Introduction to Artificial Intelligence

Assignment 2

Semester 1 2021

Edward Astbury

ID: 102579434

Lane (Danny) Macdonald

ID: 101614172

Lecturer: Bao Vo

Contents

Table of Figures	2
Instructions.....	3
Introduction.....	4
Features	4
Test cases	5
Acknowledgements and Resources.....	8
Notes	8
Research.....	11
Team summary report.....	13
Conclusion	13
References.....	14
Appendix.....	15

Table of Figures

Figure 1. TT output of the default text file	6
Figure 2. BC output of the default text file.....	6
Figure 3. FC output of the default text file	7
Figure 4. Truth table pseudo code (Vo , 2021).....	9
Figure 5. Forward chaining pseudo code (Vo ,2021)	9
Figure 6. Backward chaining pseudo code (Norvig & Russell, 2021, p.567)	10
Figure 7. Forward chaining diagram (Grosan & Abraham, p. 158)	10
Figure 8. Backward chaining diagram (Grosan & Abraham, p. 166).....	11
Figure 9. Output table from the inference engine.	11
Figure 10. Truth Table Visual Studio 19 Diagnostic times	12
Figure 11. Sequence formula for time to run the software program.....	12
Figure 12. Graph depicting the Truth Table size vs seconds to run program.....	13

Instructions

To start this program simply type the following:

iengine <method> <filename>

<method>: is the algorithm used to implement the Truth Table, Backward Chaining or Forward Chaining.

- | | |
|-------|--------------------|
| 1. TT | Truth Table |
| 2. BC | Backwards Chaining |
| 3. FC | Forward Chaining |

<filename>: is the name of the text file with arguments of a horn-form and a query with an extension .txt

The output is:

Yes: n with n being the number of entailments for the Truth Table or

No: which means that the query does not entail the KB

For the Forward and Backward Chaining the following result is like the one above. However, instead of the number of entailments, it will instead output the propositional variables entailed from the KB that have been discovered during the algorithm's execution period.

Yes: a, b, p2, p3, p1, d

The text file has a specific format that must be followed for this program to work correctly. This is set out below.

TELL

P= \Rightarrow Q; P;

ASK

Q

The above bolded writing is the requirements for the text file. TELL followed by the clauses with each one divided by a semi colon and ASK followed by a single query only as required. The TELL clauses is unlimited, however too many symbols used will result in a longer time period that may exceed one's patience as seen in the research component of this report.

Introduction

This report demonstrates the inference engine created within a group project by the authors that is written in C++. An inference engine is a system that utilises a Knowledge Base to apply a set of logical rules to infer new facts or answer queries (Coppin, 2004, p. 244). This engine uses propositional logic on a software developed using Truth Table checking, Backward Chaining and Forward Chaining algorithms to find a query and to output if it can be from the Knowledge Base.

A Truth Table, also known as a table of combinations, specifies the values of a Boolean expression for every single combination of values of the variables in the expression (Kinney & Roth, 2014).

Backward chaining starts at a goal and finds the facts relevant to prove the goal. It is a goal driven system whilst forward chaining is a data driven one (Abraham & Grosan, 2011, p. 156). Forward chaining starts with a list of facts and uses them to find a goal in which more facts can be learnt along the search for that goal.

Features

Extensive testing has been carried out on the code and there are few bugs and no missing code according to the assignment scope. Extensive testing and performance testing as seen in the research chapter have enabled a problem free program to date. This has enhanced the program and allowed us to fix bugs that we came across during this process.

The program itself can detect the following logical connectives:

1. & and
2. a (single variables) with names that can exceed a single character
3. => infers
4. a&b=>c multiple variables to this size as seen in the standard text file
5. A struct has been set to enable future development of the extended logical connectives

The current bugs are limited to it cannot fully test multiple &s greater than 2. This has not affected the program too much as it requires all symbols to equal 1 for the left side for this example to be true. It can fall into a category where an '&', is false whilst all others are true and declare it true, however have not come across this in testing.

It should be noted that on many occasions we had unanswered emails and questions, including a request to a quick 5-minute Zoom meeting to clarify the software requirements outside the given text file. This was left unanswered despite many requests, so the program was designed simply on the text file given. Later, and close to the submission date, another text file was released which had scenarios that were not tested for and would require a completely new design, without having messy unorganized code.

In a real-life situation, the client is involved in the initial design, answering questions and the requirements they expect. Given that we tried to involve this process and were denied, we would like this considered when marking. Preparing for text files that are different to the one provided to base our program off, whilst holding them classified is simply unrealistic.

Test cases

The implementation in C++ of the 3 methods was entirely designed and created from scratch using the methods learnt and researched either in class or in the textbook written and edited by Norvig and Russell (2021).

The architecture of the program can take in a text Knowledge Base and query filename and method to conduct the algorithm. The resultant gives an output as seen in the instructions chapter if the query has been successful in entailment or not.

Truth Table

The Truth Table method follows the mathematical Propositional Logic of $KB \vdash \alpha$ which simply means KB entails sentence Alpha *if and only if* alpha is true in worlds where KB is true.


The TruthTable.cpp file has multiple private data variables of vector, double vector, and string. Vectors are used because of the ease of growing an unknown size array. The file takes in a vector of clauses that has been developed within the LoadFile.cpp, a string query, a vector of sub clauses which breaks down the clauses further to aid in the program and a vector of string symbols.

A Model of the Truth Table is then developed that includes every single symbol within the text file and is 2^n deep and the number of symbols long. One can appreciate the depth can become quite large as it grows exponentially with each symbol.

Once the full Truth Table has been developed the Knowledge Base is developed and tested by using the developed Truth Table and using some clever logical methods such as 'Implication' and 'Amp' that test the clauses and return a value to put into the KB. The depth of the KB is the same as the developed Truth Table and to map each point correctly the vector of symbols is used for ordering.

The program then tests the entailment of the KB to find if there is a solution or not. It does this by a simple algorithm that maps the query location, tests each KB clause against the same clever method Amp(**bool** aLHS, **bool** aRHS) that "and's" each clause together and then tests if the developed Truth Table query symbol is true. If it is true a count increases and if not, it jumps to the next line on the KB and Truth Table and retests until the end.

If a solution is found it outputs as seen in the Figure 1. If no solution is found it simply prints out a message confirming the given query could not be solved.



```
YES: 3
```

Figure 1. TT output of the default text file

Backward Chaining

The Backward Chaining (BC) method functions by beginning at the goal state, in this case the query. It will then iterate backwards from the goal through all the clauses and validates if the subgoals can prove the query or not. This process is proving whether the query can be entailed from the KB.

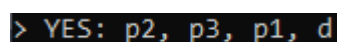
BC is a goal driven approach. Consequently, BC will only focus on the specific clauses that are relevant to the goal. In essence, BC has a goal and then it determines what are the requirements to achieve that goal. For example, a real-world analogy would be that both Lane and Edward had the specified goal of achieving a good mark for the second assignment in COS30019. Therefore, they determined the necessary requirements to achieve a good mark and worked backwards from there. In the case it entailed frequent meetings, peer code reviews, general revision and so on.

The complexity of BC in certain cases can be dramatically less than the linear size of the KB. This is largely due to BC's nature of iterating backwards from the goal and only exploring the relevant goal clauses.

In BC, the goal is broken down into sub-goals to prove the facts in the KB are true. BC needs to continuously check if the goal is already part of the list of facts. This prevents any unnecessary iterations of the loop occurring. In a similar vein, BC should also incorporate some form of checking to prevent the algorithm from rediscovering the same subgoals (newly discovered literals). This safeguard measure is illustrated in Grosan & Abraham's BC Figure 8 diagram.

Our group's implementation of the BC algorithm is reminiscent of both Norvig & Russell's pseudo code in Figure 6 and even more so Grosan & Abraham's BC diagram in Figure 8. All the functionality of BC is contained within the BackwardChaining.cpp. The algorithm operates by placing the premise in the agenda, from here it will then go through the clauses that have the potential of satisfying the query, then it will go through the individual facts that satisfy those clauses. In this case, every fact is treated as a subgoal. This process will continually loop until there is no more clauses to satisfy. This is reminiscent of a depth first search strategy and produces a linear time where it is proportional to the number of clauses in the KB.

If a solution can be found, then the facts discovered in the algorithm's execution will be printed out (see Figure 2 below):



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

Figure 2. BC output of the default text file

In the case where the query cannot be proven then a message will be printed which states the given query could not be proven.

Forward Chaining

The Forward Chaining method (FC) functions by making a conclusion based on known facts. The known facts represent the initial state whilst the goal state is represented by the query being asked. FC is a data driven, automated process, it does not know whether the explored clauses are going to contribute towards the goal state. It simply explores the clauses in an unconscious manner. In essence, it is repeating the same inference process to get more and more data. Consequently, FC may find itself doing a lot of additional processing that is not relevant towards the goal. Therefore, BC is considered a more efficient algorithm compared to FC.

A real-world analogy of how FC operates can be viewed in a sporting competition and who is awarded the best and fairest for the year. At the end of every game the umpires will note down who were the best three players for the game. This data will be tallied every game until the season is over. From here we can derive the final goal of who is the best and fairest (data driven approach).

Programmatically, FC operates by finding any rule whose premises (values before the implication) are satisfied in the KB. It will then add its conclusion (value after the implication) to the list of fact results until the query is found. The goal always represents the termination condition for the algorithm. When the algorithm is at the point where the goal is derived no further rules should be applied (see Figure 4. Forward Chaining pseudo code (Vo, 2021)).

FC should be viewed as a form of reasoning which begins with facts in the KB and applies inference rules in a forward direction to extract more data until a goal is attained.

Our group's implementation of the FC algorithm operates in a similar manner to the BC algorithm. The primary difference is that the vector in the core algorithm loop holds a vector of facts that are present in the KB. The list of facts is then compared to every clause in the KB. If the fact is found within the KB the conclusion of the clause that holds the fact is added onto the vector and the previous fact is then popped off as it is no longer required as it has successfully inferred a clause. This process will continually iterate until the query can be proven or not.

The printing of the solution is identical to BC except that the facts discovered during the algorithm's execution pertain to how FC discovered the query.

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

Figure 3. FC output of the default text file

Acknowledgements and Resources

The resources used to create this program were the unit's book by Norvig and Russell (2021), the lecture materials on Canvas, emails to the lecturer Bao Quoc Vo that were replied to and explanations by tutor Charles Harold in person. Some other resources used to enhance our understanding of the algorithms were Eddie Woo's YouTube channel <[Eddie Woo - YouTube](#)>, Education 4u's YouTube channel <[forward chaining Example-2 | Artificial intelligence | Lec-39 | Bhanu Priya - YouTube](#)> and the books Intelligent Systems (Abraham & Grosan, 2011) & Artificial intelligence Illuminated (Coppin 2004).

Notes

The implementation of the algorithms, whilst following the sound and complete model, enhanced the simplicity of each one to make it easier for the program to replicate.

The Truth Table method, instead of a recursive model that passes a Boolean value back through to the beginning of the search of each model, develops it separately. This then allows the program to search and allowed for extensive testing whilst printing out a full table of each model. That is, Truth table, sub clause truth table and clause truth table.

The Forward Chaining method follows the same design structure as Vo's pseudo code in Figure 5. The primary difference is that it does not use a 'count' variable to deduce facts and add them to the agenda. Instead, an 'if' check is present that checks whether the conclusion of the clause being explored already contains a fact that is already present in the vector of facts. If it is, then there is no need to iterate down this path, thus we delete that clause.

The Backward Chaining method follows a similar process to Grosan & Abraham's flowchart in Figure 8, except it does not follow a recursive model. Instead, a loop is used which will continue to iterate until there is no more clauses to satisfy in the KB. This produces the same outcome but using the loop enables ease of use when using a C++ vector in a stack like manner.


```

function TT-ENTAILS?( $KB, \alpha$ ) returns true or false
     $symbols \leftarrow$  a list of the proposition symbols in  $KB$  and  $\alpha$ 
    return TT-CHECK-ALL( $KB, \alpha, symbols, []$ )

```

```

function TT-CHECK-ALL( $KB, \alpha, symbols, model$ ) returns true or false
    if EMPTY?( $symbols$ ) then
        if PL-TRUE?( $KB, model$ ) then return PL-TRUE?( $\alpha, model$ )
        else return true
    else do
         $P \leftarrow$  FIRST( $symbols$ );  $rest \leftarrow$  REST( $symbols$ )
        return TT-CHECK-ALL( $KB, \alpha, rest, \text{EXTEND}(P, \text{true}, model)$ ) and
            TT-CHECK-ALL( $KB, \alpha, rest, \text{EXTEND}(P, \text{false}, model)$ )

```

Figure 4. Truth table pseudo code (Vo , 2021)

```

function PL-FC-ENTAILS?( $KB, q$ ) returns true or false
    local variables:  $count$ , a table, indexed by clause, initially the number of premises
                      $inferred$ , a table, indexed by symbol, each entry initially false
                      $agenda$ , a list of symbols, initially the symbols known to be true

    while  $agenda$  is not empty do
         $p \leftarrow$  POP( $agenda$ )
        unless  $inferred[p]$  do
             $inferred[p] \leftarrow true$ 
            for each Horn clause  $c$  in whose premise  $p$  appears do
                decrement  $count[c]$ 
                if  $count[c] = 0$  then do
                    if HEAD[ $c$ ] =  $q$  then return true
                    PUSH(HEAD[ $c$ ],  $agenda$ )

    return false

```

Figure 5. Forward chaining pseudo code (Vo ,2021)

function FOL-BC-ASK($KB, query$) **returns** a generator of substitutions
return FOL-BC-OR($KB, query, \{\}$)

function FOL-BC-OR($KB, goal, \theta$) **returns** a substitution
for each $rule$ **in** FETCH-RULES-FOR-GOAL($KB, goal$) **do**
 $(lhs \Rightarrow rhs) \leftarrow$ STANDARDIZE-VARIABLES($rule$)
for each θ' **in** FOL-BC-AND($KB, lhs, UNIFY(rhs, goal, \theta)$) **do**
yield θ'

function FOL-BC-AND($KB, goals, \theta$) **returns** a substitution
if $\theta = failure$ **then return**
else if LENGTH($goals$) = 0 **then yield** θ
else
 $first, rest \leftarrow$ FIRST($goals$), REST($goals$)
for each θ' **in** FOL-BC-OR($KB, SUBST(\theta, first), \theta$) **do**
for each θ'' **in** FOL-BC-AND($KB, rest, \theta'$) **do**
yield θ''

Figure 6. Backward chaining pseudo code (Norvig & Russell, 2021, p.567)

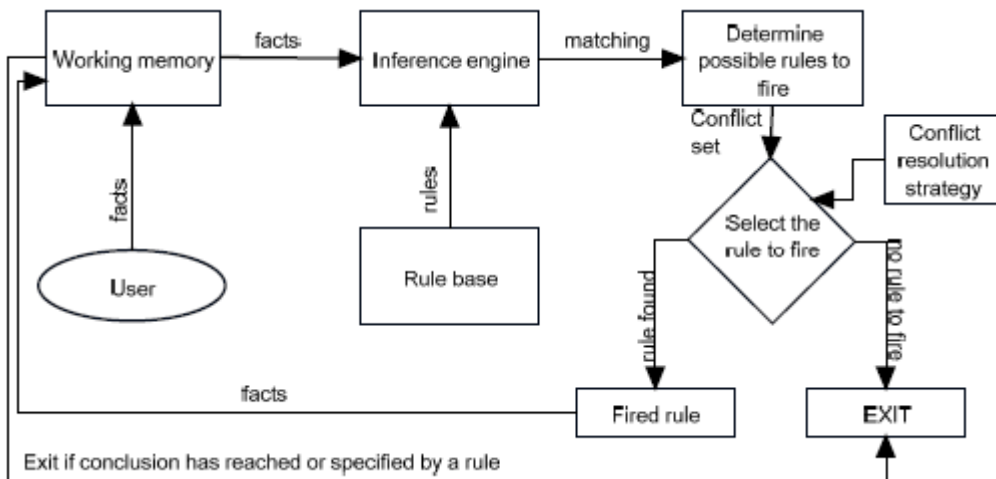


Figure 7. Forward chaining diagram (Grosan & Abraham, p. 158)

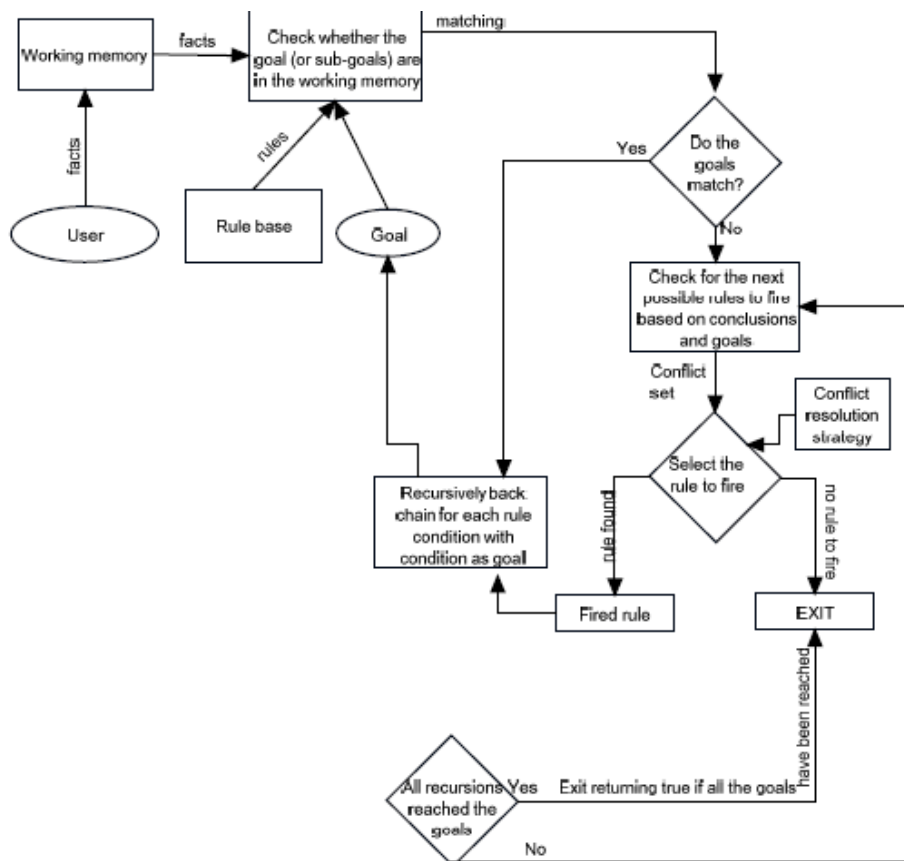


Figure 8. Backward chaining diagram (Grosan & Abraham, p. 166)

Research

The research component involves thorough testing of multiple complex text files and a mix of known outputs and discovered outputs. The table below in figure 9 displays the list of text files that can be viewed in full in the Appendix.

Discussion

Text File	Truth Table	Truth Table size	Forward Chaining	Backward Chaining
1	3	2048	a, b, p2, p3, p1, d	p2, p3, p1, d
2	1	16384	P14 could not be proven	P12&p11, p12&p4&p5&p10, p2&p6&p8&p10, p11, p13, p14
3	3	4096	a, b, p2, p3, p1, p4, d	p2, p3, p1, p4, d
4	3	2048	a, b, p2, p3, p1, d	p2, p3, p1, d
5	1	16	a, b, c, d	a, b, c, d
6	4	16	d could not be proven	d could not be proven
7	4	16	d	d
8	9	256	a, b, c, d	a, b, c, d
9	1	1024	a, h, i, b, d, c, e, j, f, g	j, f, d, h, i, b, c, e, g
10	1	1024	a, h, i, b, d, c, e, j, f, g	j, f, d, h, i, b, c, e, g

Figure 9. Output table from the inference engine.

Whilst it is difficult to obtain a correlation with the data, its apparent that backward chaining uses less facts than the forward chaining method. Priya Pedamkar explains that forward chaining checks all the rules, making it slow, while backward chaining is faster as it only checks the required rules (2020).

The Truth Table size plays a key role in the time it takes to create the table within a C++ vector. The larger number of symbols the more time it takes. Figure 10 gives a limited view into the time it takes, which is linear.

2^n	2^{10}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Time taken (s)	0	2.154	4.473	9.516	20.218	42	89	189

Figure 10. Truth Table Visual Studio 19 Diagnostic times

Figure 11 clearly shows the data collected from Figure 10, in which the larger the Truth Table the linear time shift in seconds to complete an outcome. This is in line with expectations with a Truth Table being 2^n in size. A 2^{30} size Truth Table was run and tested, however after 1.5 hours the program was shutdown. According to this graph from Figure 11 it can be deduced it would take approximately 56.6 hours with the formula developed from the data and produced in figure 11.

As seen, this can be a lengthy wait for larger tables, therefore other methods of deducing facts should be examined before developing the Truth Table. Whilst forward chaining examines all facts, backwards chaining only outputs the required facts to gain the query asked. It would be recommended to backwards chain first and use only those in the TELL text file to develop the Truth Table, greatly eliminating and saving precious time and computer resources.

$$a_{t_1} = a_1 + (2 \times n)$$

$$a_{t_2} = a_1 \times 2 + (2 \times n)$$

$$a_{t_3} = a_2 \times 2 + (2 \times n) \dots a_{t_n} = a_{n-1} \times 2 + (2 \times n)$$

Note: $a_1 = 40$ with the formula starting at 2^{18}

Figure 11. Sequence formula for time to run the software program.

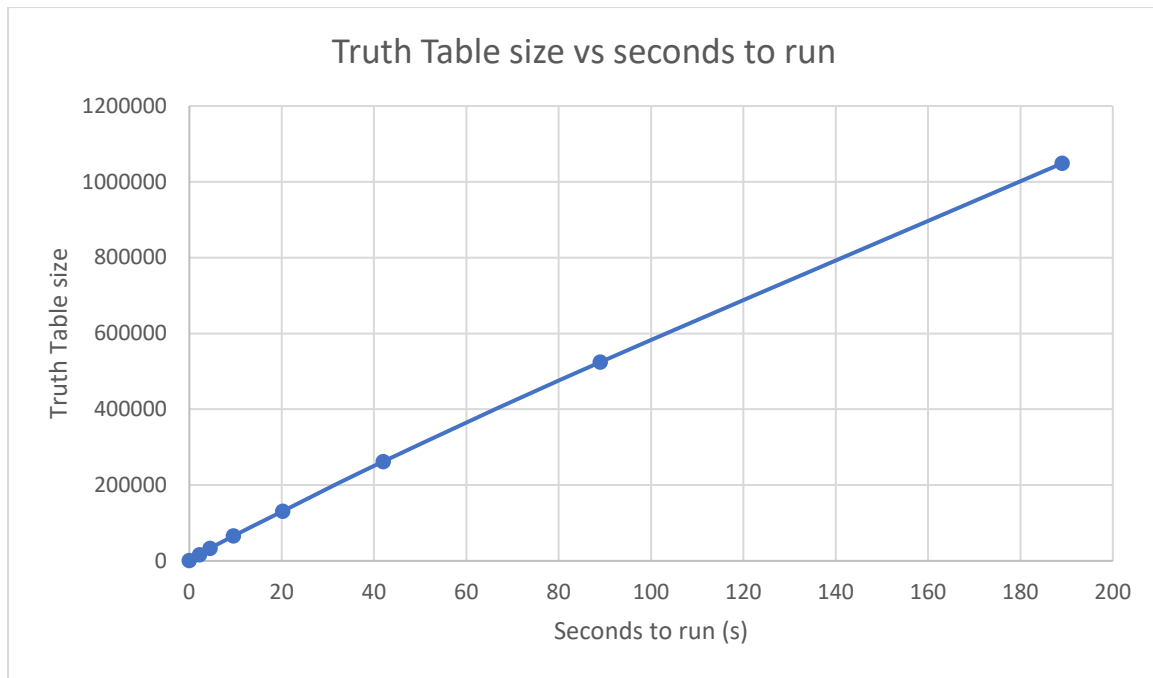


Figure 12. Graph depicting the Truth Table size vs seconds to run program

Team summary report

Both the authors have worked together regularly over the last 3 years, so have both a good understanding of where each other's strengths are and what is required to complete a high-level project.

Regular constant communication, planning, Discord meetings and questioning methods has worked well during this project with early planning and work. Whilst it was a group effort, Edward concentrated on both the Forward and Backwards Chaining whilst Lane worked on the Truth Table and Report structure. Both members tested and went through each other's code to ensure both a fundamental understanding but also to find flaws that needed covering to ensure a better program.

Both have inputted 50% each into the project with many hours spent well beyond the scope of the 12.5 hours a week.

Conclusion

This report steps out the instructions on how to use the program designed on C++ to use the algorithms in the inference engine as well as how to write and use the text files to create clauses and queries. It introduces the idea of the inference engine and propositional logic as well as its features.

The methods used, being the Truth Table, Backward Chaining and Forward Chaining were extensively tested to ensure as many bugs as possible were accounted for.

The Truth Table, whilst incredibly resource heavy due to the exponential increase in resources for each added symbol certainly makes writing one on paper a thing of the past.

Forward Chaining on the other hand is rather simplistic in nature but does possess pitfalls. Its inability to be conscious of whether a clause is going to contribute towards solving the query creates the opportunity for unnecessary processing in the KB.

Backward Chaining is the most efficient of the three. Beginning from the goal state enables the algorithm to only explore the necessary clause and subclauses that infer the goal.

Ultimately, all the algorithms provide their own distinctive attributes, features and unique functionality when working with an inference engine for propositional logic. However, it is evident that the efficiency and complexity of the algorithms vary drastically, meaning in certain scenarios one algorithm may be favored over another.

References

- Abraham, A, Grosan, C 2011, *Intelligent Systems, A modern approach*, Springer, vol 17, Berlin Heidelberg.
- Coppin, B 2004, *Artificial Intelligence Illuminated*, Jones and Bartlett Publishers, Sudbury, MA
- Kinney, L, Roth, C 2014, *Fundamentals of Logic Design*, 7th edn, CENGAGE Learning, Stamford, CT
- Norvig, P, Russell, P 2021, *Artificial Intelligence: A Modern Approach*, 4th edn, Pearson, Hoboken, NJ
- Pedamkar, P 2020, 'Forward Chaining vs Backward Chaining', EDUCBA, viewed 22 May 2021, <[Forward Chaining vs Backward Chaining | Top 9 Differences to Learn \(educba.com\)](https://www.educba.com/forward-chaining-vs-backward-chaining/)>.
- Vo, B 2021, 'Lecture 7. Logical Agents & Knowledge Representation' COS30019- Introduction to Artificial Intelligence, learning materials via Canvas, Swinburne University of Technology, 19 April, viewed 19 April 2021.

Appendix

Text files used in testing.

Text File	Contents
1	TELL p2=> p3; p3 => p1; c => e; b&e => f; f&g => h; p1=>d; p1&p3 => c; a; b; p2; ASK d
2	TELL p1&p2&p3&p4&p5&p6&p7 => p8; p1&p3&p4&p6&p7&p8 => p9; p6&p7&p4&p5&p9 => p10; p13&p12&p11 => p14; p11&p12&p4&p5&p10 => p13; p11&p1&p4&p5&p6 => p12; p1&p2&p6&p8&p10 => p11; p1&p2&p6&p8 => p; p1; p2; p3; p4; p5; p6; p7; ASK p14
3	TELL p2=> p3; p3 => p1; p1 => p4; p4 => p1; c => e; b&e => f; f&g => h; p4=>d; p1&p3 => c; a; b; p2; ASK d
4	TELL p2=> p3; p3 => p1; c => e; b&e => f; f&g => h; p1&p2=>d; p1&p3 => c; a; b; p2; ASK d
5	TELL a => b; b =>c; c => d; a; ASK d
6	TELL a => b; b =>c; c => d; ASK d
7	TELL a => b; b =>c; c => d; d; ASK d
8	TELL a => b; b =>c; c => d; e => g; l => m; a; ASK d
9	TELL a&b => c; c&d => e; e&f => g; h => i; i =>b; b => d; j => f; a; h; j; ASK g
10	TELL a&b => c; c&d => e; e&f => g; e&f => f; h => i; i =>b; b => d; j => f; a; h; j; ASK g

Text File	Description
1	Default text file provided.
2	Multiple '&s'.
3	Situation where the algorithm could iterate into a dead end.
4	When the '&' symbol is required to prove the query.
5	Simple set of clauses.
6	When the query cannot be proven (no facts present).
7	When the query is the fact.
8	When there are irrelevant clauses.
9	Complex clauses with multiple where the '&' is required to prove the query.
10	When an entailed literal is entailed by itself (the value after the implication is already part of the list of facts).

Text File	Truth Table Result	Forward Chaining Result	Backward Chaining Result
1	3	a, b, p2, p3, p1, d	p2, p3, p1, d
2	1	P14 could not be proven	P12&p11, p12&p4&p5&p10, p2&p6&p8&p10, p11, p13, p14
3	3	a, b, p2, p3, p1, p4, d	p2, p3, p1, p4, d
4	3	a, b, p2, p3, p1, d	p2, p3, p1, d
5	1	a, b, c, d	a, b, c, d
6	4	d could not be proven	d could not be proven
7	4	d	d
8	9	a, b, c, d	a, b, c, d
9	1	a, h, i, b, d, c, e, j, f, g	j, f, d, h, i, b, c, e, g
10	1	a, h, i, b, d, c, e, j, f, g	j, f, d, h, i, b, c, e, g