COS30019

Introduction to Artificial Intelligence

Assignment 2: Inference Engine

NAM DUONG HOANG

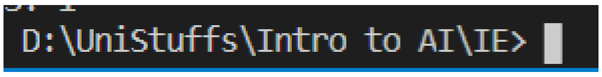
STUDENT ID: 101720286

Table of Contents

I. Instructions

1. Usage

The program is a Java-based program which can be executed using windows' **cmd** shell. Firstly, run **cmd** with normal user privilege. Then navigate to the folder which contains the **.bat** file of the program using the command **cd <folder address>**.

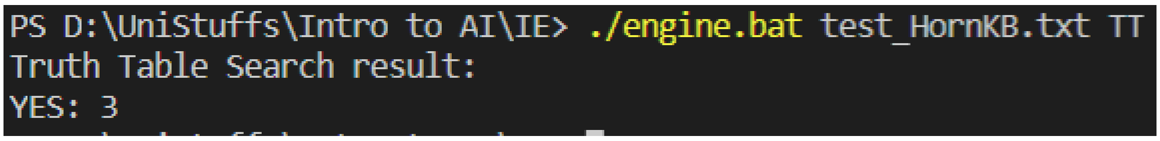


```
D:\UniStuffs\Intro to AI\IE>
```

After navigating to the folder, the program can be executed by running the command **engine.bat <file> <method>**. "*File*" is the file containing the test case, "*method*" is the keyword of the search method the user wanted to use. There are 3 implemented methods:

Short name	Full method name
TT	Truth Table Search
FC	Forward Chaining Search
BC	Backward Chaining Search

For example, the user wants to test a case in a text file called **test_HornKB.txt** using Truth Table Search, the command and response is as follow:



```
PS D:\UniStuffs\Intro to AI\IE> ./engine.bat test_HornKB.txt TT
Truth Table Search result:
YES: 3
```

II. Features

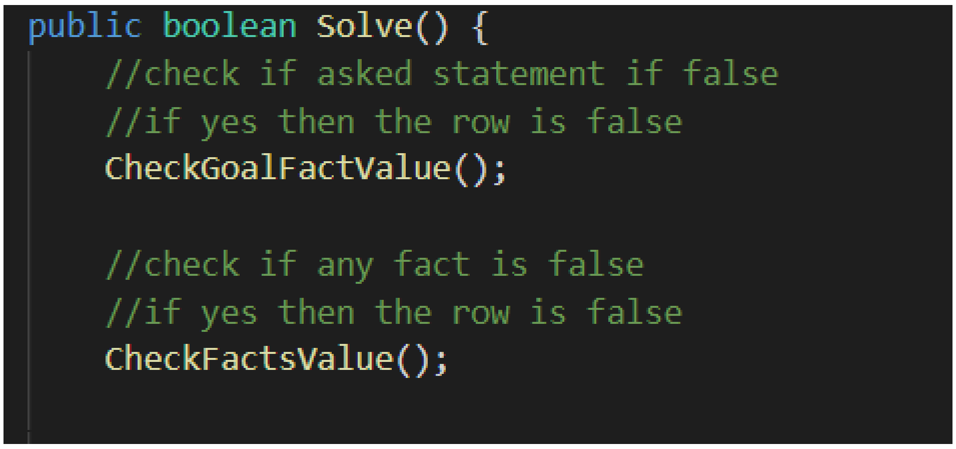
1. Truth Table Algorithm

a. Definition

A Truth Table is a math-related logical table which shows every functional values of logical expressions for each combination of values of its variables. (Anon., n.d.)

b. Implementation

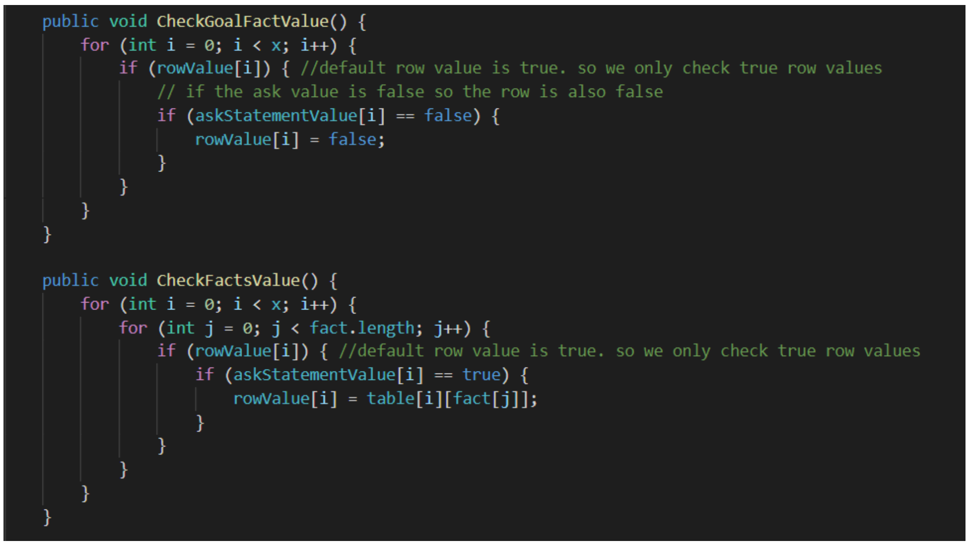
The Truth Table "TTMethod" Class is inherited from its parent class "SearchMethod". Firstly, all global variables are initialized in the constructor class. The Truth Table is implemented, and all values are assigned. When the user uses the method, the printResult() method is called, which triggers the Solve() method to start the algorithm.



```
public boolean Solve() {
    //check if asked statement if false
    //if yes then the row is false
    CheckGoalFactValue();

    //check if any fact is false
    //if yes then the row is false
    CheckFactsValue();
```

The Solve() method first call 2 special functions to check if any rows which have the goal statement's value or any of the given facts' value is false. If yes, the row's value is also set to false.

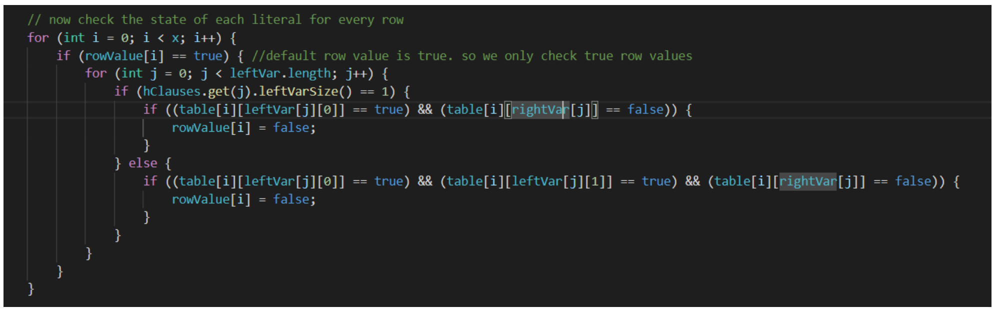


```
public void CheckGoalFactValue() {
    for (int i = 0; i < x; i++) {
        if (rowValue[i]) { //default row value is true. so we only check true row values
            // if the ask value is false so the row is also false
            if (askStatementValue[i] == false) {
                rowValue[i] = false;
            }
        }
    }
}

public void CheckFactsValue() {
    for (int i = 0; i < x; i++) {
        for (int j = 0; j < fact.length; j++) {
            if (rowValue[i]) { //default row value is true. so we only check true row values
                if (askStatementValue[i] == true) {
                    rowValue[i] = table[i][fact[j]];
                }
            }
        }
    }
}
```

Figure 2: Code snippet of 2 check functions

Next, the Solve() method checks the Truth Table for the values of each rows in the table by comparing 2 sides of each clauses. For example, P => Q, if P is true and Q is false, then the row value is set to false, or P + A => Q, if P is true, A is true and Q is false, then the row value is set to false. Any other cases sets the row value to true.



```
// now check the state of each literal for every row
for (int i = 0; i < x; i++) {
    if (rowValue[i] == true) { //default row value is true. so we only check true row values
        for (int j = 0; j < leftVar.length; j++) {
            if (hClauses.get(j).leftVarSize() == 1) {
                if ((table[i][leftVar[j][0]] == true) && (table[i][rightVar[j]] == false)) {
                    rowValue[i] = false;
                }
            } else {
                if ((table[i][leftVar[j][0]] == true) && (table[i][leftVar[j][1]] == true) && (table[i][rightVar[j]] == false)) {
                    rowValue[i] = false;
                }
            }
        }
    }
}
```

Finally, the function calls a final check function to check if the truth table satisfies the conditions to prove the goal statement or not, and then returns the result in Boolean state.

2. Forward Chaining Algorithm

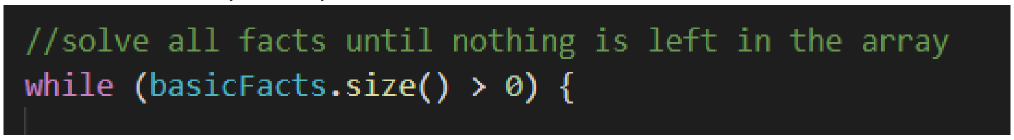
a. Definition

Forward Chaining is an algorithm which starts with the given data, in this case is the facts provided, and explores them to get mor data and information. (Anon., n.d.)

b. Implementation

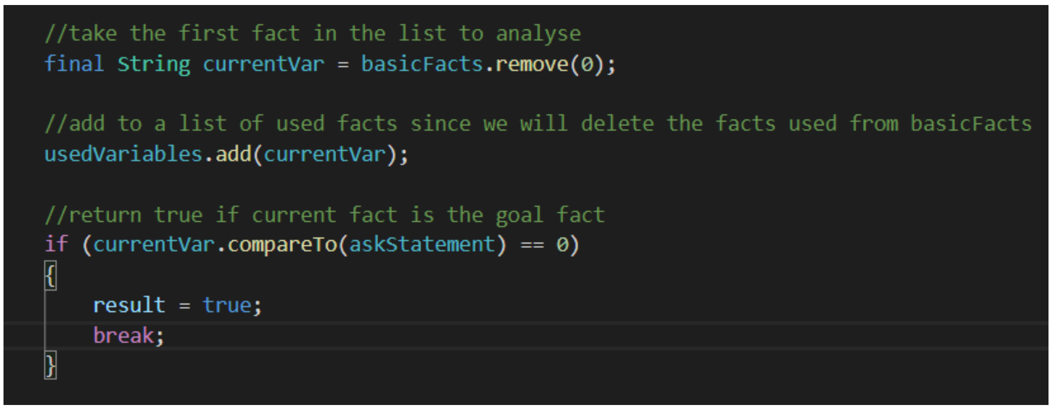
The process of calling and functioning in Forward Chaining "FCMethod" function is similar to the process provided in 1. Truth Table Algorithm.

Forward Chaining Method also inherits from the parent class "SearchMethod". The abstract method Solve() in "FCMethod" first starts a while loop to run until all facts in the basicFacts array are explored.



```
//solve all facts until nothing is left in the array
while (basicFacts.size() > 0) {
```

The used fact is popped from the array basicFacts and stored in a temporary variable to be explored. It is also added to another array to be output to the terminal. First the fact is checked if it is the goal state or not. If yes, the loop is ended and the search succeeded.

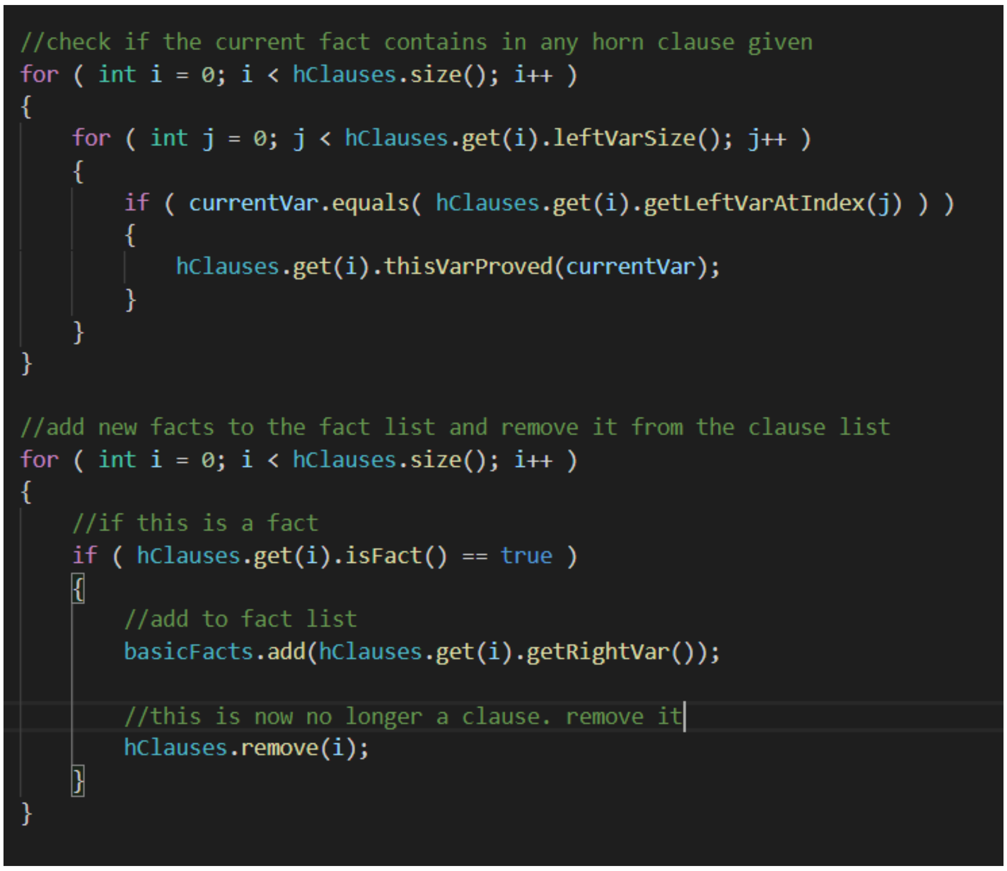


```
//take the first fact in the list to analyse
final String currentVar = basicFacts.remove(0);

//add to a list of used facts since we will delete the facts used from basicFacts
usedVariables.add(currentVar);

//return true if current fact is the goal fact
if (currentVar.compareTo(askStatement) == 0)
{
    result = true;
    break;
}
```

If not, the fact is continued to be explored. It will be checked if it is contained in any left sides of given clauses. If yes, the clause will check that variable is proved by removing it from the left variable array. When the left variable array contains no elements, that means the right variable has been proved as a fact and added to the basicFacts array to be explored. The loop continues until the goal is found or the loop ended. The method will then returns a Boolean value of the search result.



```java
//check if the current fact contains in any horn clause given
for ( int i = 0; i < hClauses.size(); i++ )
{
    for ( int j = 0; j < hClauses.get(i).leftVarSize(); j++ )
    {
        if ( currentVar.equals( hClauses.get(i).getLeftVarAtIndex(j) ) )
        {
            hClauses.get(i).thisVarProved(currentVar);
        }
    }
}

//add new facts to the fact list and remove it from the clause list
for ( int i = 0; i < hClauses.size(); i++ )
{
    //if this is a fact
    if ( hClauses.get(i).isFact() == true )
    {
        //add to fact list
        basicFacts.add(hClauses.get(i).getRightVar());

        //this is now no longer a clause. remove it
        hClauses.remove(i);
    }
}
```

3. Backward Chaining Algorithm

a. Definition

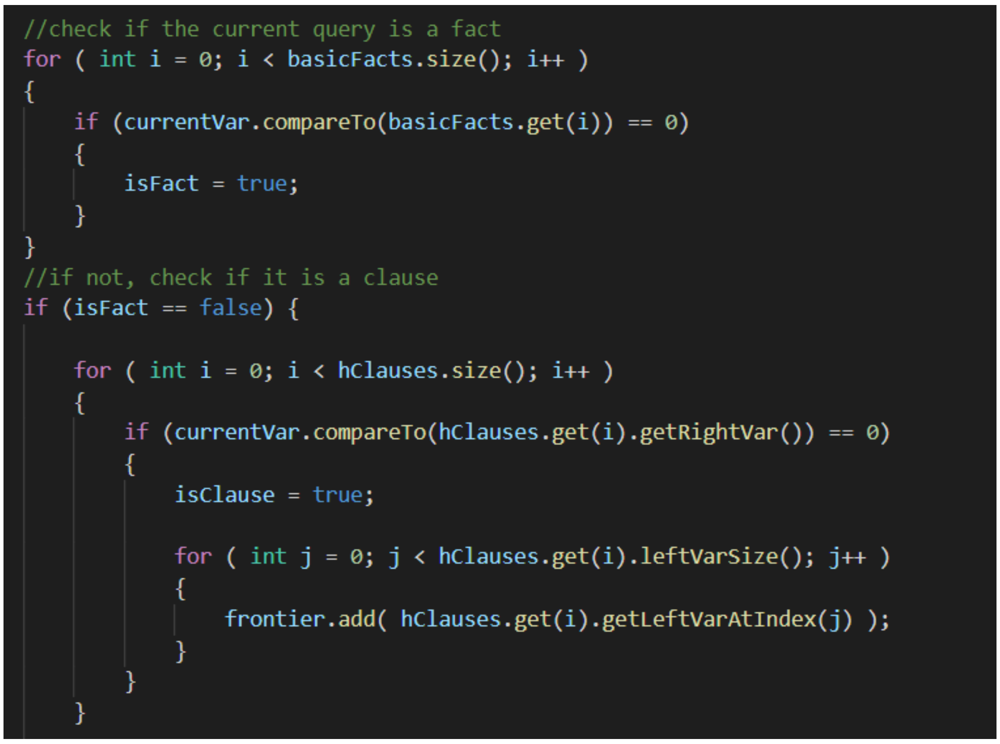
Backward Chaining is the opposite of Forward Chaining Algorithm stated before. This algorithm starts exploring the goal statement to find the facts the supports in finding it.

b. Implementation

The Implementation of this method is very much similar to the implementation of Forward Chaining. Therefore, I will talk about how the Solve() method works.

Firstly, the method starts a while loop to keep running until the variable “frontier” has no elements left. This variable only contains the goal statement at first. During the loop, explored facts will be added to the array to be explored. Each time the loop starts, an element in the “frontier” array will be popped and examined.

The popped variable “currentVar” is first examined to see whether it is a fact or a clause. If it is a fact, it is marked as a fact and continues the loop. If it is not a fact, it will be compared with the right variables of all clauses. If it is a match, then it will be marked as a clause. And the clauses’ left variables will be added to the “frontier” queue.

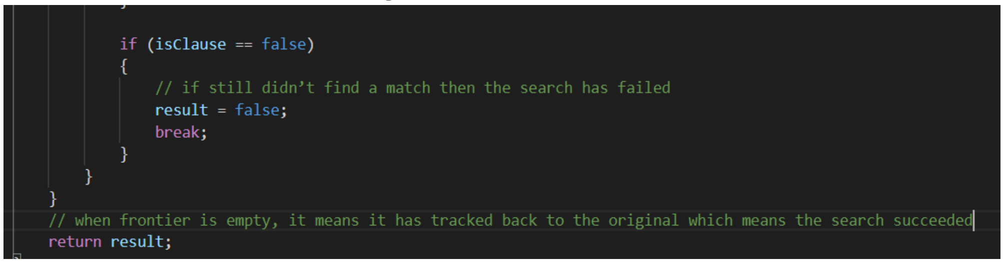


```java
//check if the current query is a fact
for ( int i = 0; i < basicFacts.size(); i++ )
{
    if (currentVar.compareTo(basicFacts.get(i)) == 0)
    {
        isFact = true;
    }
}
//if not, check if it is a clause
if (isFact == false) {

    for ( int i = 0; i < hClauses.size(); i++ )
    {
        if (currentVar.compareTo(hClauses.get(i).getRightVar()) == 0)
        {
            isClause = true;

            for ( int j = 0; j < hClauses.get(i).leftVarSize(); j++ )
            {
                frontier.add( hClauses.get(i).getLeftVarAtIndex(j) );
            }
        }
    }
```

The loop will continue until there is no elements left in the queue and returns TRUE. The loop will return FALSE and breaks if “currentVar” is not marked as a fact nor a clause, which means the backtrack has gone into a dead-end.

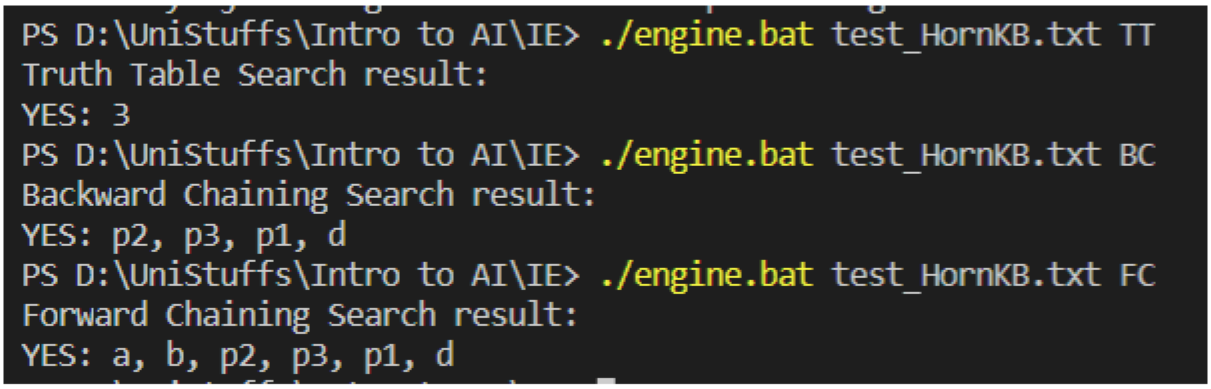


```java
            if (isClause == false)
            {
                // if still didn't find a match then the search has failed
                result = false;
                break;
            }
        }
    }
    // when frontier is empty, it means it has tracked back to the original which means the search succeeded
    return result;
```

III. Testing

1. Provided test case:

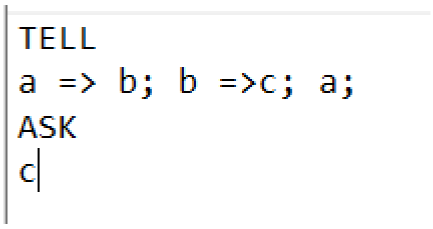
Output:



```
PS D:\UniStuffs\Intro to AI\IE> ./engine.bat test_HornKB.txt TT
Truth Table Search result:
YES: 3
PS D:\UniStuffs\Intro to AI\IE> ./engine.bat test_HornKB.txt BC
Backward Chaining Search result:
YES: p2, p3, p1, d
PS D:\UniStuffs\Intro to AI\IE> ./engine.bat test_HornKB.txt FC
Forward Chaining Search result:
YES: a, b, p2, p3, p1, d
```

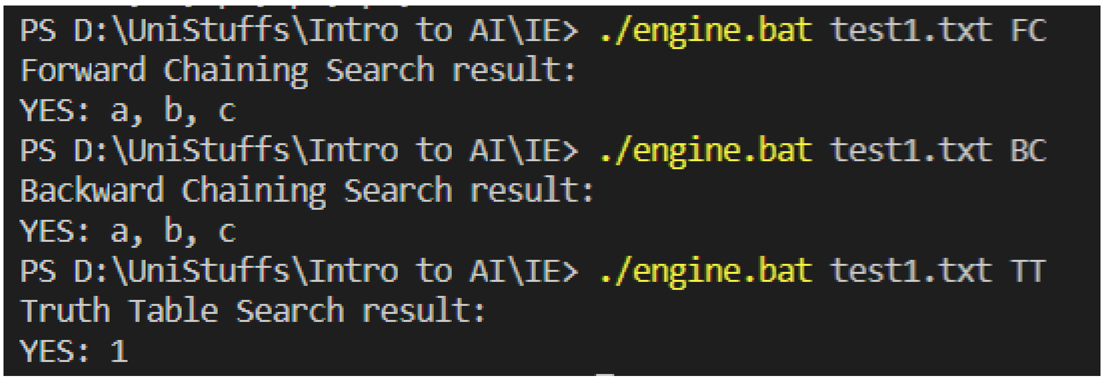
No bugs found.

2. Test case 1: Simple, straight-forward clauses



```
TELL
a => b; b =>c; a;
ASK
c
```

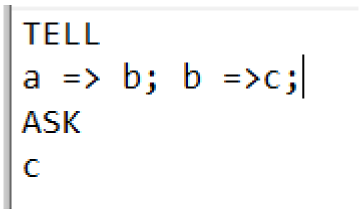
Output:



```
PS D:\UniStuffs\Intro to AI\IE> ./engine.bat test1.txt FC
Forward Chaining Search result:
YES: a, b, c
PS D:\UniStuffs\Intro to AI\IE> ./engine.bat test1.txt BC
Backward Chaining Search result:
YES: a, b, c
PS D:\UniStuffs\Intro to AI\IE> ./engine.bat test1.txt TT
Truth Table Search result:
YES: 1
```

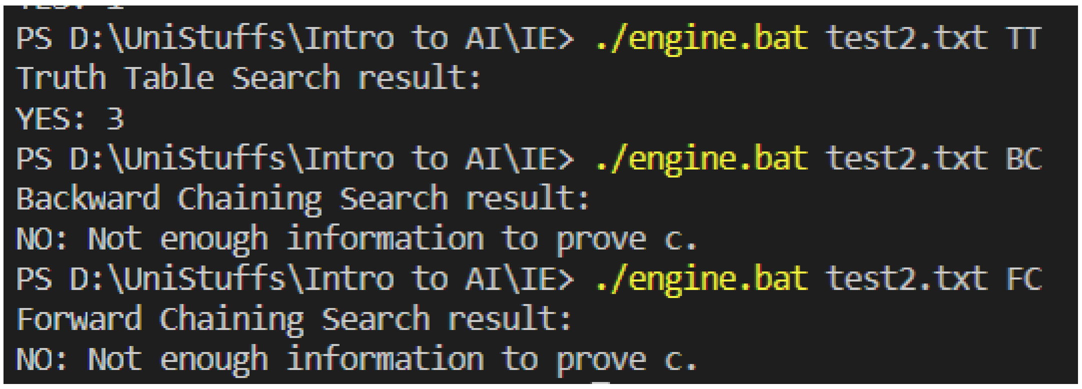
No bugs found.

3. Test case 2: No facts given. Should output NO



```
TELL
a => b; b =>c;
ASK
c
```

Output:

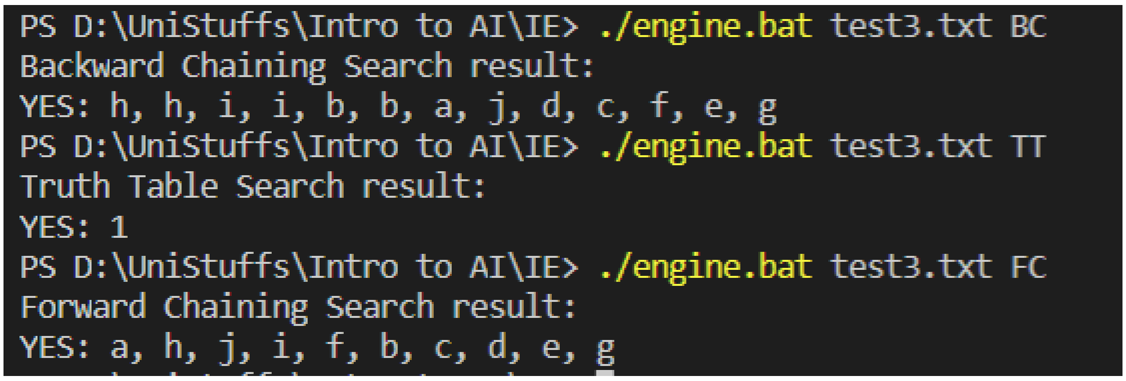


```
PS D:\UniStuffs\Intro to AI\IE> ./engine.bat test2.txt TT
Truth Table Search result:
YES: 3
PS D:\UniStuffs\Intro to AI\IE> ./engine.bat test2.txt BC
Backward Chaining Search result:
NO: Not enough information to prove c.
PS D:\UniStuffs\Intro to AI\IE> ./engine.bat test2.txt FC
Forward Chaining Search result:
NO: Not enough information to prove c.
```

Bug: Truth Table still output YES because the condition to ‘c’ still satisfied *NOT RESOLVED*

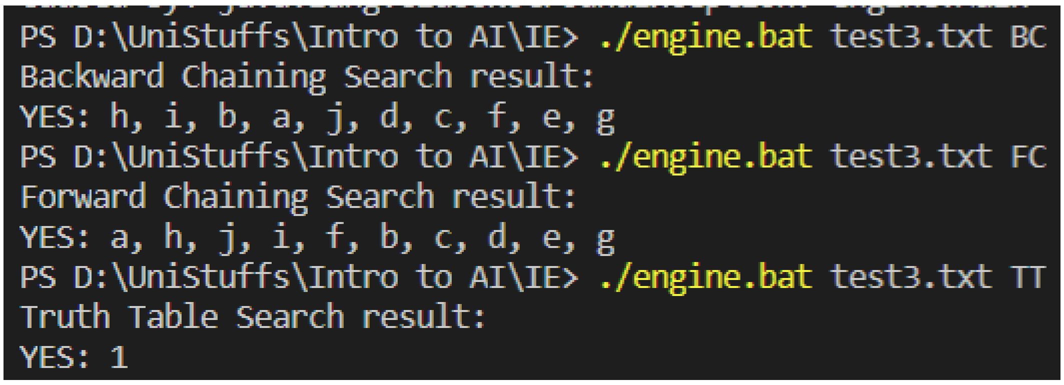
4. Test case 3: Complex clauses which have both 2 left variables and 1 left variables

Output before:



```
PS D:\UniStuffs\Intro to AI\IE> ./engine.bat test3.txt BC
Backward Chaining Search result:
YES: h, h, i, i, b, b, a, j, d, c, f, e, g
PS D:\UniStuffs\Intro to AI\IE> ./engine.bat test3.txt TT
Truth Table Search result:
YES: 1
PS D:\UniStuffs\Intro to AI\IE> ./engine.bat test3.txt FC
Forward Chaining Search result:
YES: a, h, j, i, f, b, c, d, e, g
```

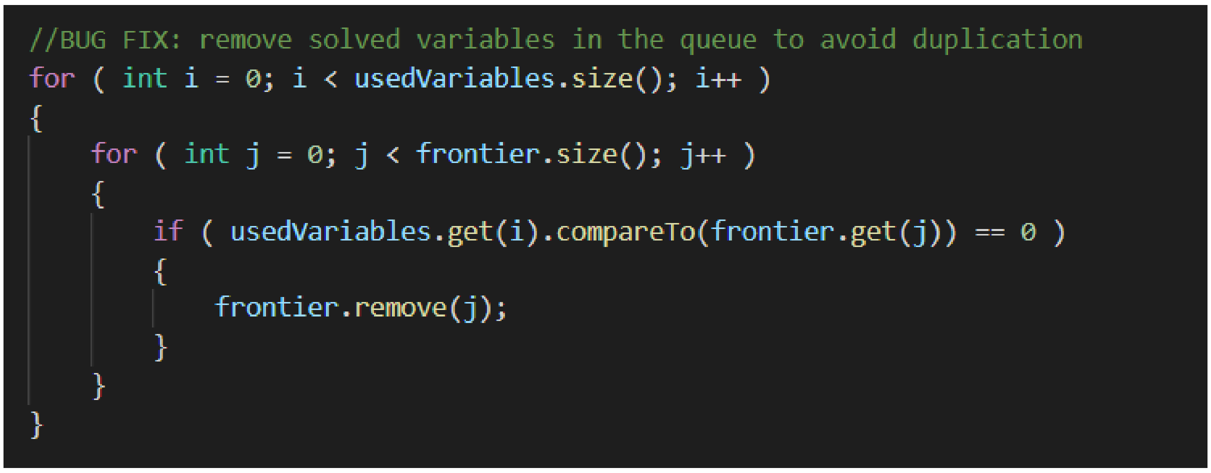
Output after:



```
PS D:\UniStuffs\Intro to AI\IE> ./engine.bat test3.txt BC
Backward Chaining Search result:
YES: h, i, b, a, j, d, c, f, e, g
PS D:\UniStuffs\Intro to AI\IE> ./engine.bat test3.txt FC
Forward Chaining Search result:
YES: a, h, j, i, f, b, c, d, e, g
PS D:\UniStuffs\Intro to AI\IE> ./engine.bat test3.txt TT
Truth Table Search result:
YES: 1
```

Bugs: The test case had some duplicate outputs. This was due to the array was not cleaned which allowed elements duplicates.

Solution:



```java
//BUG FIX: remove solved variables in the queue to avoid duplication
for ( int i = 0; i < usedVariables.size(); i++ )
{
    for ( int j = 0; j < frontier.size(); j++ )
    {
        if ( usedVariables.get(i).compareTo(frontier.get(j)) == 0 )
        {
            frontier.remove(j);
        }
    }
}
```

IV. Resources

1. Forward and Backward Chaining algorithm: https://www.javatpoint.com/forward-chaining-and-backward-chaining-in-ai
   This website helped me to get a better understanding of how the A* Algorithm works in a coding environment.
2. Remove array duplicates: https://stackoverflow.com/questions/22352934/remove-duplicates-from-an-array-in-java
   This website provided useful information about removing duplicates in array, which helped me in bug fixing
3. Multi-dimensional array:
   - https://careercup.com/question?id=17632666
   - https://stackoverflow.com/questions/10723168/generating-truth-tables-in-java/10761325

   These websites contain useful information Truth table in Java and how to populate them in coding.
4. Lecture notes, slides and videos was the thing that helped me the most by providing detailed information about each methods, clear examples and easy-to-understand lecture videos.

References

Anon., n.d. *JavaTPoint.* [Online]
Available at: https://www.javatpoint.com/forward-chaining-and-backward-chaining-in-ai

Anon., n.d. *Wikipedia.* [Online]
Available at: https://en.wikipedia.org/wiki/Truth_table