ASSIGNMENT 3

SUBJECT CODE: CS 6300

SUBJECT: ARTIFICIAL INTELLIGENCE

LEENA KORA
EMAIL:leenak@cs.utah.edu

Unid: u0527667

IMPLEMENTATION OF INFERENCE METHODS

Documentation and Discussion

1 . A short problem description

Explanation: The problem in hand is regarding the implemenation of two inference methods. There are three major inference algorithms. They are forward chaining, backward chaining and resolution-based theorem-proving systems. Reasoning using forward or backward chaining is much more efficient than reasoning using resolution.

Forward chaining is a very simple algorithm. Forward chaining algorithm determines whether a single proposition symbol(query) is entailed by a knowledge base. Foward chaining is a form of data-driven reasoning. It starts from known facts(positive literals) in the knowledge base and apply Modus Ponens in the forward direction, adding new facts , until no further inferences can be made. In order words if all the premises of an implication are known, then its conclusion

is added to the database. And a fact should not be added if it is just a renaming of a known fact.
Forward chaining is sound, since every inference is an application of Modus Ponens. Forward chaining is complete since every entailed proposition will be derived. Forward chaining is very easy to understand and it just needs pattern matching. As Forward chaining needs pattern matching, it can very expensive. It migth generate many facts that are irrelevant to the goal.

Backward chaining is a form of goal-directed reasoning. Backward chaining works backwards from the goal. The algorithm first finds all the implications in the knowledge base that concludes the query. Then if all the premises of one of those implications can be proved true then the query can be stated as true.
Backward chaining algorithm states from the goal and reaches a set of relevant known facts. Backward chaining is an efficient algorithm for propositional inference based models. Backward chaining suffers from problems with repeated states and incompleteness.

For this assignment, we need to implement forward and backward chaining methods applicable to propositional logic. Both of these algorithms are very natural, in that the inference steps are obvious and easy to follow. The knowledge base consists of assertions and implications. Assertions are propositions that are true and implications involve one or more premises and a single consequent. Both premises and consequent will be propositions. There is an implicit 'and' connective between each proposition in the premise. There is no explicit representation of an 'or' connective in this system. Negation operation is not supported in this system.

Implementation of the two inference algorithms, which are forward and backward chaining, is somewhat very interesting to deal with. We come to know how they work and what advantages and disadvantages each have.

As Forward chaining algorithm is a form of data-driven reasoning, we can use it within an agent to make it derive new facts from the percepts and from the facts in the knowledge base. While implementing machine learning concepts, an agent can also infer facts whenever its needed. As Backward chaining algorithm is a form of goal-directed reasoning, we can use it within an agent to make it answer specific questions.

2. A description of the solution.

Explanation: I first created an Input_processing class. That class has functions as 'get_validstring()' to accept input from the user. For simplicity, the input line entered using 'A:' command is considered as assertions and the input line entered using 'I:' command is considered as implications. I decided to use vectors and vector of vectors as the major data structures as the user can enter any number of facts or implications respectively. Input_Process class has two vectors of type string as 'facts' and 'derived_facts' as data members to store facts(assertions) and derived facts. It also has a data member as 'implications' which is a vector of vector of strings to store implications(inferences). Input_Process class has functions such as process_facts() and process_implications() to take the input line and store the entered facts and implications into the respective data structures.

That class also has data members like Input_line to accept the user input. It also has facts_count, premises_count, implication_count, input_line_count, proposition_count to keep track of whether the system has valid number of facts, premises, implications, characters in the input line and proposition respectively. It also mainatains a string as check_string to store the query.

Input_Process class has a function called get_user_input() which calls all the functions such as check_for_quit(), check_for_summary(), check_for_print(), check_for_assertions(), check_for_implications to check which command the user has entered and calls the respective appropriate functions such as Print_facts(), Print_implications(), process_for_fc(), process_for_bc(), Do_forward_chaining() and

Do_backward_chaining() which in turn calls Backward_chaining(). Some constants are also defined as Max_Facts(100), Max_Premises(20), Max_Implications(100), MaxProposition_length(20), MaxInput_length(99) for checking the validity of the input line by comparing them with the respective counters.

My Do_forward_chaining() function implements the forward chaining algorithm. It first checks whether the query is in the knowledge base or not and if not then it displays the appropriate message. Then it checks whether it is in the vector of facts and if so then it displays the appropriate message. And then it loops through all the implications to derive new facts untill it derives the required fact(query) or no further facts can be inferred. It also keeps track of the facts generated during the process.

My Do_backward_chaining() function implements the backward chaining algorithm. It first checks whether the query is in the knowledge base or not and if not then it displays the appropriate message. It then calls Backward_chaining(), which is a recursive function, with the query. This function first checks whether the query is in the vector of facts and if so then it returns true. It then loops through all the implications to find out any implication that has the query as the consequent. Then for each of the premises of that implication, it calls itself to check for the target implication to infer each premises. If each of the permises are true then it returns true. It also keeps track of the number of facts verified during the process and number of facts actually present in the vector of facts.

3. A discussion of the problems I encountered in solving the problem and how I overcame them, the weaknesses and strengths of your solution, and any other interesting observations.

Explanation: Processing the input line was straight forward but required lot of coding to check for validity and character count and other counts limitations. My forward and backward chaining are implemented well and they give the correct output.I tried to check for each all most all invalid inputs and made my program to flag appropriate error messages when they are encountered.

As per the code to accept the propositions from the user which have digits, lower and upper case character and '_' symbol. But I did not enforce that the propositions should not begin with digits or '_' symbol because it was not mentioned in the assignment to do it.

The only problem I faced was, I was using separate vector to store derived facts and for each run of the program it was the derived facts and was displaying inappropriate number of derived facts in the output. I then solved the problem by clearing the vector of derived facts each time the forward chaining algorithm is being called.

Testing information

Explanation: I tested my program with various test cases. Following are the test cases for my program.

I also added error messages for flagging when my program reads invalid input from the user. In below test cases, I purposely entered arbitrary number of blank spaces to show that it was being supported by my program.

- My first test case is as follows

```
-> A: !tyy

 Invalid proposition
 So enter again
```

The purpose of the test was to check whether my program accepts only valid propositions that is propositions having digits, upper andd lower alphabetic characters and underscore symbol.

- My second test case is as follows

```
-> A:                                   rr r

   Invalid proposition
   So enter again
```

The purpose of the test was to see whether the input line with "A:" at the begin takes only valid proposition or not. As a valid propostion does not have blank spaces in it, my program flagged an error message for it.

- My third test case is as follows

```
->I:        r

  Invalid implication, so enter again
```

The purpose of the test was to see whether my program takes only valid implications. As valid implications should have atleast two implications, my program displayed an error message for it.

- My four test case is as follows

```
-> A: leena

-> p
 Facts:
 leena
 Implications:

 A: leena
```

The assertion that is being enter is already in the database

The purpose of the test was to see whether my program checks all the assertions in the database to see if the given assertion is already in the database. As assertion "leena" was already in the database, it flagged a message.

- My fifth test case is as follows

```
-> A:

Invalid Syntax....No string was being entered
```

The purpose of the test was to see whether it checks for the correct syntax of the given input line or not.

- My sixth test case is as follows

```
-> errr
```

```
Invalid Input..Type '?' on the command line to know the right syntax
```

The purpose of the test was to see whether it detects an invalid entry and guides the user in a better way by telling them to type '?' on the command line .

- My seventh test case is as follows

```
I:      1 2
A:    2
F:      2

Attempting to verify 2 using forward chaining.
forward chaining: searched for fact is found!
2 in assertion database
2 is true
0 facts generated
```

The purpose of the test was to check whether forward chaining implementation checks the given proposition in the database and displays the message accordingly.

- My next test case is as follows

```
I:  1  2
I:  2  3
A:       1
F:    3
```

```
Attempting to verify 3 using forward chaining
forward chaining: derived 2
forward chaining: derived 3
forward chaining: searched for fact is found!
3 is true
2 facts generated
```

The purpose of the test was to see whether it generates correct number of facts or not.

- My tenth test case is as follows

```
I:    1    2
I:    2     3
A:                 1
F:          2
```

```
Attempting to verify 2 using forward chaining.
forward chaining: derived 2
forward chaining: searched for fact is found!
2 is true
1 fact generated
```

The purpose of the test was to see whether my implementation of forward chaining algorithm works properly or not.

- My Eleventh test case is as follows

```
->A   : ff
Invalid Input line, there should be ':' as a second character in the string
So try again
->p
Facts:
Implications:
```

The purpose of the test was to see whether it detects the above invalid input and does not store in the database.

- My next test case is as follows

```
->  N  :   ghgh
 Invalid Input..Type '?' on the command line to know the right syntax
```

The purpose of the test was to see whether it detects another invalid input or not.

- My next test case is as follows

```
-> P:  gdh hdh
Invalid Input..Type '?' on the command line to know the right syntax

-> q:  dhgsfd  sgdhgh
```

```
Invalid Input..Type '?' on the command line to know the right syntax

-> ?  :  hjhhd dj
Invalid Input..Type '?' on the command line to know the right syntax
```

The purpose of the test was to see whether it detects the above invalid inputs or not.

- My next test case is as follows.

```
->?(some blank spaces here)
Invalid Input..Type '?' on the command line to know the right syntax

?
A: P                  [P is true]
I: P1 P2 ... PN Q     [P1 & P2 & ... & PN ==> Q]
B: P                  [is P true (backward chaining)]
F: P                  [is P true (forward chaining)]
P                     [print facts and implications]
?                     [print summary of commands]
q                     [quit]
```

The purpose of the test was to see whether it takes arbitrary number of blank spaces after the valid input line i.e '?' for tha above case or not. My program gave an error message for that, so I correct that bug in my code.

- My next test case is as follows.

```
->(type nothing)
 Invalid Input..Type '?' on the command line to know the right syntax
```

The purpose of the test was to see whether it displays a useful message when the user does not type anything on the command line.

- My next test case is as follows.

```
-> I:      :    1    2

Invalid proposition
So try again
```

The purpose of the test was to see whether it detects the an invalid proposition when entered for the implications.

- My next test case is as follows.

```
 A:
ghjhhhhhhhhhhhhhhhhhhhhh
Input line exceeds the maximum limit, so try again
```

The purpose of the test was to see whether it detects that the entered input line exceeds the length limit i.e 100 characters and whether it displays the appropriate error message or not.

- My next test case is as follows.

```
-> A:   eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee

     Invalid proposition because it exceeded the limit of 20 characters
     Try  again
```

The purpose of the test was to see whether it detects that the entered proposition exceeds the length limit i.e 20 characters and whether it displays the appropriate error message or not.

- My next test case is as follows.

```
->   I: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
P
Facts:
Implications:
     Premises:
               1
               2
               3
               4
               5
```

```
                    6
                    7
                    8
                    9
                    10
                    11
                    12
                    13
                    14
                    15
                    16
                    17
                    18
                    19
                    20
              consequent:
                    21
         -> I: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
          The number of premises have exceeded the count i.e 20...So the entered implication will not be added
         to the knowledge base
```

In the above test case, I first entered 20 premises for an implication and my program accepted it and stored it in the data base and when I entered 21 then it did not. The purpose of the test was to see whether it detects that the entered implication is invalid and whether it displays the appropriate error message.

- My next test case is as follows.

```
Trace flag is set
Stats flag is set

 A: s
 I: s f
 F: f

 Attempting to verify f using forward chaining.
 forward chaining: derived f
 forward chaining: searched for fact is found!
 f is true
 1 fact generated

 B: f
 Attempting to verify f using backward chaining.
 Attempting to verify s using backward chaining.
 backward chaining: s in assertion database
 f is true
 2 attempted fact verifications, 1 actually verified
```

The purpose of the test was to see whether my program works correctly for the above test case or not.

- My next test case is as follows.

```
A: smart
A: money
```

```
I: smart money college_degree
I: college_degree good_job
I:    college_degree big_house
I:  good_job big_house happy
B: happy
Attempting to verify happy using backward chaining.
Attempting to verify good_job using backward chaining.
Attempting to verify college_degree using backward chaining.
Attempting to verify smart using backward chaining.
backward chaining: smart in assertion database
Attempting to verify money using backward chaining.
backward chaining: money in assertion database
Attempting to verify big_house using backward chaining.
Attempting to verify college_degree using backward chaining.
Attempting to verify smart using backward chaining.
backward chaining: smart in assertion database
Attempting to verify money using backward chaining.
backward chaining: money in assertion database
happy is true
9 attempted fact verifications, 4 actually verified
```

The purpose of the test was to see whether it shows correct trace and stats information for the above example.

Answers to additional discussion questions

1. What would be the effect on performance of alternate ways of ordering the search for asserted propositions and potentially relevant implications?

Explanation: If ordering the search for asserted propositions and relevant implications is altered, then that surely effects the performance.

As the inference algorithms often loop through each of the implications and try to process each proposition to infer new facts or required query, order of the facts and implications is very important in terms of performance.

If the assertions are properly ordered with respect to the relevant implications then the program will be more faster and performs well. In some cases the order can even effects the performance in terms of memory. Consider the following example. to illustrate it.

```
I: a b
I: c a
I: a e
I: e a
A: c
```

If we pass a query 'b' for backward chaining function then, it returns true.

```
B: b
b is true
```

If we alter the order of the implications in the data base as below, then backward chaining function goes into infinite loop if some sort of book keeping is not maintained.

```
    I: a b
    I: a e
    I: e a
    I: c a
    A: c

   ->B: b
(infinite loop)
```

2. Give an example of a problem for which forward chaining inference would likely be more appropriate than backward chaining inference.

Explanation: When the knowlegde base has circular implications or repeated states(propositions) forward chaining inference would be more appropriate than backward chaining inference.

Forward chaining algorithm is a form of data-driven reasoning, so we can use it within an agent to make it derive new facts from the percepts and knowledge base. Actually forward chaining inference is used for the problems whose final states are unknown. Hence it is more appropriate for problems that need planning, design or process monitoring. One Such example of a problem is suggesting next steps in the diligence investigation.

3. Give an example of a problem for which backward chaining inference would likely be more appropriate than forward chaining inference.

Explanation: In case of problems having knowledge bases with less rules(implications) and huge number of facts, backward chaining inference will be better to use than forward chaining inference so that we can process only those facts that are relevant to fulfilling goals.

Backward chaining algorithm is a form of goal-directed reasoning, we can use it within an agent to make it answer specific questions and have better justification or explanation mechanisms. Hence backward chaining inferences are more appropriate for diagnostic and classification tasks.

4. Given an identical set of asserted propositions (`A:` command) and implications (I: command), under what circumstances would forward chaining (`F:` command) and backward chaining (`B:` command) give different results? Consider only the answer to whether or not the proposition is true, not the output of the -trace command or purely performance-related issues.

Explanation: I think this question was a tricky one.

I would like to specify the above example here.

```
I: a b
I: a e
I: e a
I: c a
A: c

->B: b
(infinite loop)
```

I: a b
I: a e
I: e a
I: c a
A: c

->F: b

b is true.

From the above example it is quit clear that forward chaining inference proved 'b' to be true but backward chaining inference was not able to infer it.

5. How can the backward chaining algorithm be modified to avoid looping on circular inferences of the form:

```
-> I: chicken egg
-> I: egg chicken
-> B: egg
```

Is this also a problem for forward chaining inference?

Explanation: One way to avoid looping on circular inference is to keep track of all the propositions that we were trying to prove and if we encounter any proposition which was previously tried to prove, then backward chaining function should return false, displaying an appropriate message as "Encountered circular inferences....So the query can not be supported".

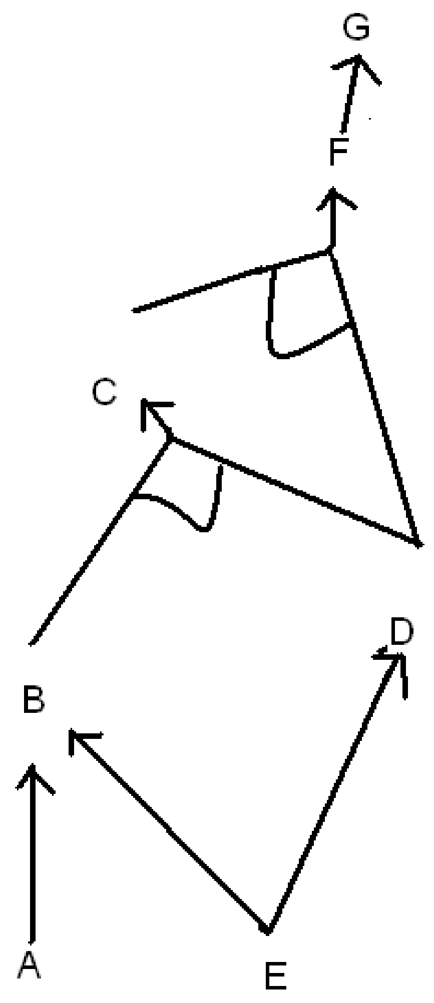
No, circular inference will not be a problem for forward chaining inference. It always begins with the known facts and if the query doen not match with any of the assertions, then it processes all implications one by one to derive new facts if all of the premises of an implication in hand are true. In the above example it first processes the first implication and it tries to infer "egg" by checking whether "chicken" is in the facts list. As it is not in the knowledge base, it moves to next impication. As "egg" is not in the knowledge base, it can infer "chicken", so it concludes that query(egg) can not be supported.

6. Show that your backward chaining algorithm does a search of an and/or graph.

Explanation: Consider the below Knowledge base

F=>G
C and D=>F
D and B =>C
A =>B
E=>B
E=>D
A
E

The AND-OR graph for the above case



When the backward chaining algorithm is given the query 'G', it processes from top to the bottom of the graph to infer 'G' reaching the known facts at the end. To infer 'G', it has to prove 'F' and to infer 'F' it has to prove both(and) 'C' and 'D'. Then to infer 'C', it has to process both 'B' and 'D'. And to infer 'B' it has to infer any one(or) of 'A' or 'E'. And to infer 'D', it has to prove 'E'. To infer necessary propositions it has to search them then process them. In this way the Backward chaining does a search of an And-Or graph.

7. What is the computational (time) complexity of the forward and backward search procedures implemented in this assignment, considering both searches through the set of propositions known to be true and searches for relevant implications? Note that this question asks about computational complexity, and cannot be answered with an empirical investigation of run times.

Explanation: In case of forward chaining inference method, the computational(time) complexity is as follows

Let 'A' be the number of assertions, 'I' be the number of implications and 'P' be the number of premises per implication in the working memory. Then the worst case time complexity for the forward chaining will be O(A*I*P).

In case of backward chaining inference method, the computational(time) complexity is as follows

Let 'A' be the number of assertions, 'I' be the number of implications, 'P' be the number of premises per implications in the working memory and 'D' be the depth of the recursion which depends on the given problem. Then the worst case time complexity for the backward chaining will be O((A*I*P)^D)

8. What would be involved in adding negation of propositions to the two inference methods?

Explanation: Adding negation of propositions to the two inference methods requires some modifications.

For this assignment, symbol " ! " can be used at the beginning of the proposition to indicate at that it is a negative form of it. If any proposition has '!' at the beginning of it, then it should interpret as false. The implementation should have some kind of method which takes a proposition and returns whether it is true or not. If the knowledge base contains positive proposition and the premise being proved is just a negation of that proposition then that methods must conclude the given proposition as 'false' and vise versa.