CMPT310
ARTIFICIAL INTELLIGENCE SURVEY

Assignment 3: Backward Chaining - Report

Krystle Bulalakaw
301134933

Outline of Procedure

This program solves a list of rules within a knowledge base in order to prove a query, *q*. The algorithm used is the inference method: backward chaining. The entire program is contained within BackwardChaining.java, with 2 methods: *init()* which reads the input file given by the user in command prompt and sets up the initial values to be used by *backwardChaining()*, which tries to prove the query *q* (the first line in the input file). It does this by trying to prove all the premises of some rule concluding *q,* recursively by backward chaining. It essentially follows the pseudocode provided in the assignment:

```
solve(goals):
    if goals = () then return true
    let a = first(goals); goals = rest(goals)
    for each r ∈ rules where head(r) = a
        if solve(append(body(r), goals)) = true
            return(true)
    return(fail)
```

The program starts with *q* and iterates through the knowledge base to find a clause that has a body/conclusion that matches *q*'s head. If a clause is found, its predicates are added to a list of subgoals to be evaluated. Then backward chaining is performed on each of these subgoals until the list of goals is empty, at which point it exits from recursion and can evaluate if the main query can be proven true or not.

For example, trying to solve the sample data provided in the assignment:

```
r
p
q p
r p q
r s
```

In this case, since **r** is the goal, it looks for a clause like **r⇒p^q**. If the head of that clause (in this case, p^q) is not known to be true, it is added to a list of goals. Since **p^q** is a conjunction of two statements, they are broken down into two subgoals which are both added to the list of goals. So p and q are the new goals. To prove these subgoals, the program may find that p and was given as a fact (simply, **p**).

So, by this method, it finds p is true. p implies q is true. p and q implies r, therefore r is true.

The program makes an effort to avoid useless work and repetition, for example:

- It avoid loops by checking if a new subgoal is already in the list of goals
- It avoids repeating work by checking if the new subgoal has already been proven true
- A clause which is not useful to solving the problem is not stored or output in the result (for example, **r⇒s**

Output

As the program evaluates each subgoal, the following diagnostics are printed: the list of goals, the current goal being evaluated, whether or not the current goal has been found as a conclusion/head within a clause/body in the KB, and goals visited so far. Finally, if the result of proving the ultimate goal is successful, it will print TRUE and the order of entailments to reach the query. If the result is unsuccessful, it will print FALSE.

Success

Here is sample output generated by the problem provided above, which can be proven true:

```
Filename: src/data1.txt

RULES:
fact: p is TRUE
implication: IF p THEN q
conjunction: IF p AND q THEN r
implication: IF s THEN r

BACKWARD CHAINING DIAGNOSTICS:
First query: r

Goals to evaluate: [r]
Evaluating r...
Found r as conclusion in (r p q)
Truth table: [r p q]

Goals to evaluate: [p, q]
Evaluating q...
Found q as conclusion in (q p)
Truth table: [r p q, q p]

Goals to evaluate: [p]
Evaluating p...
Found p as conclusion in (p)
Truth table: [r p q, q p, p]

RESULT:
p is true
p implies q
p and q implies r
Therefore r is TRUE
```

Failure

On the other hand, for input of a problem that cannot be proven true, such as:

```
p
q p
d
g q j
e f
c f d
c d g
c e d
b j
a b c
j
```

(This was also provided in the assignment). A sample of the output generated is as follows:

```
Filename: src/data2.txt

RULES:
implication: IF p THEN q
fact: d is TRUE
```

```
conjunction: IF q AND j THEN g
implication: IF f THEN e
conjunction: IF f AND d THEN c
conjunction: IF d AND g THEN c
conjunction: IF e AND d THEN c
implication: IF j THEN b
conjunction: IF b AND c THEN a
fact: j is TRUE

BACKWARD CHAINING DIAGNOSTICS:
First query: p

Goals to evaluate: [p]
Evaluating p...

RESULT:
p could not be proven in the knowledge base.
Therefore p is FALSE
```

Other Discussion

In regards to Note #5 in the assignment, if this program were to encounter rules such as **p^q⇒p** or the pair of rules **q⇒p** and **p⇒q**, it should make inferences about these statements, store those inferences in the knowledge base, and print them to the user.

Finding the rule **p^q⇒p** should produce the implication that **p⇒q**, because q must be true in order to make **p^q** true. If not q, then not p (by modus tollens).

Finding the pair of rules **q⇒p** and **p⇒q** should produce the biconditional rule **p↔q** (p if and only if q), as they are logically equivalent. It is also equivalent to “both or neither”, “(not p or q) and (not q or p)”, and “(p and q) or (not p and not q)”.