Unit 3 Activities

Activity 1- Chapter 2, Question 4 – The Barber's Paradox

In a certain town, there are the following regulations concerning the town barber: Anyone who does not shave himself must be shaved by the barber. Whomever the barber shaves, must not shave himself. Show that no barber can fulfil these requirements. That is, formulate the requirements as sentences of FOL and show that in any interpretation where the first regulation is true, the second one must be false. (This is called the barber's paradox and was formulated by Bertrand Russell.)

1. Define Predicates:

- Let shaves(x, y) represent the predicate "x shaves y".

2. Define Constant:

- Let barber represent the barber in the town.

3. Formalize the Regulations:

- **Regulation 1:** "Anyone who does not shave himself must be shaved by the barber." ∀x (¬shaves (x, x) → shaves (barber, x))
- **Regulation 2:** "Whomever the barber shaves, must not shave himself." ∀x (shaves (barber, x) → ¬shaves (x, x))

4. Show the Contradiction:

To show the contradiction, we substitute barber for x in both formulas:

- From Regulation 1: ¬shaves (barber, barber) → shaves (barber, barber)
- From Regulation 2: shaves(barber, barber) → ¬shaves(barber, barber)

Now, let's consider the possibilities:

- **Case 1: Assume shaves(barber, barber) is true.**
  - Then, from Regulation 2, ¬shaves(barber, barber) must also be true, which is a contradiction.
- **Case 2: Assume ¬shaves(barber, barber) is true.**
  - Then, from Regulation 1, shaves(barber, barber) must be true, which is again a contradiction.

In both cases, we arrive at a contradiction. Therefore, there is no interpretation where both Regulation 1 and Regulation 2 can be true. This demonstrates that no barber can fulfil these requirements.

Activity 1- Chapter 3 Question 4 - A Canadian variant of an old puzzle.

A Canadian variant of an old puzzle:

A traveller in remote Quebec comes to a fork in the road and does not know which way to go to get to Chicoutimi. Henri and Pierre are two local inhabitants nearby who do know the way. One of them always tells the truth, and the other one never does, but the traveller does not know which is which. Is there a single question the traveller can ask Henri (in French, of course) that will be sure to tell him which way to go?

We will formalize this problem in FOL. Assume there are only two sorts of objects in our domain: inhabitants, denoted by the constants henri and pierre; and French questions, which Henri and Pierre can answer. These questions are denoted by the following terms:

gauche, which asks if the traveller should take the left branch of the fork to get to Chicoutimi;

dit_oui(x, q), which asks if inhabitant x would answer yes to the French question q;

dit_non(x, q), which asks if inhabitant x would answer no to the French question q.

Obviously, this is a somewhat impoverished dialect of French, although a philosophically interesting one. For example, the term

dit_non (henri, dit_oui (pierre, gauche))

represents a French question that might be translated as, “Would Henri answer no if I asked him if Pierre would say yes, I should go to the left to get to Chicoutimi?” The predicate symbols of our language are the following:

Truth_teller(x), which holds when inhabitant x is a truth teller.

Answer_yes (x, q), which holds when inhabitant x will answer yes to French question q;

True(q), which holds when the correct answer to the question q is yes.

Go_left, which holds if the direction to get to Chicoutimi is to go left.

For purposes of this puzzle, these are the only constant, function, and predicate symbols.

(a) Write FOL sentences for each of the following:

One of Henri or Pierre is a truth teller, and one is not.

An inhabitant will answer yes to a question if and only if he is a truth teller and the correct answer is yes, or he is not a truth teller, and the correct answer is not yes.

The gauche question is correctly answered yes if and only if the proper direction is to go is left.

A dit_oui(x, q) question is correctly answered yes if and only if x will answer yes to question q.

A dit_non(x, q) question is correctly answered yes if and only if x will not answer yes to q.

Imagine that these facts make up the entire KB of the traveller.

(b) Show that there is a ground term t such that in other words, there is a question t that can be asked to Henri (and there is an analogous one for Pierre) that will be answered yes if and only if the proper direction to get to Chicoutimi is to go left.

(c) Show that this KB does not entail which direction to go, that is, show that there is an interpretation satisfying the KB where Go_left is true, and another one where it is false.

(a) FOL Sentences:

- **One of Henri or Pierre is a truth teller, and one is not:**
  - `(Truth_teller(henri) ∧ ¬Truth_teller(pierre)) ∨ (¬Truth_teller(henri) ∧ Truth_teller(pierre))`
- **An inhabitant will answer yes to a question if and only if he is a truth teller and the correct answer is yes, or he is not a truth teller, and the correct answer is not yes:**
  - `∀x ∀q (Answer_yes (x, q) ↔ ((Truth_teller(x) ∧ True(q)) ∨ (¬Truth_teller(x) ∧ ¬True(q))))`
- ***The gauche question is correctly answered yes if and only if the proper direction is to go is left:***
  - `True(gauche) ↔ Go_left`
- ***A dit_oui (x, q) question is correctly answered yes if and only if x will answer yes to question q:***
  - `∀x ∀q (True (dit_oui (x, q)) ↔ Answer_yes (x, q))`

- ***A dit_non (x, q) question is correctly answered yes if and only if x will not answer yes to q:***
    - `∀x ∀q (True(dit_non(x, q)) ↔ ¬Answer_yes(x, q))`

(b) The Ground Term t

The question the traveller can ask Henri is: "Would Pierre say yes to the question *gauche*?". In FOL, this is represented by the ground term dit_oui (pierre, gauche).

Let's show that Answer_yes (henri, dit_oui(pierre, gauche)) is true if and only if go_left is true.

- We want to prove: Answer_yes (henri, dit_oui (pierre, gauche)) ↔ Go_left

Here's the proof:

1. Answer_yes (henri, dit_oui(pierre, gauche)) ↔ ((Truth_teller(henri) ∧ True (dit_oui(pierre, gauche))) ∨ (¬Truth_teller(henri) ∧ ¬True(dit_oui(pierre, gauche)))) (Applying the 2nd FOL sentence where x is henri and q is dit_oui(pierre, gauche))
2. True (dit_oui(pierre, gauche)) ↔ Answer_yes(pierre, gauche) (Applying the 4th FOL sentence where x is pierre and q is gauche)
3. ¬True (dit_oui(pierre, gauche)) ↔ ¬Answer_yes(pierre, gauche) (Negating both sides of 2)
4. Substituting 2 and 3 into 1: Answer_yes(henri, dit_oui(pierre, gauche)) ↔ ((Truth_teller(henri) ∧ Answer_yes(pierre, gauche)) ∨ (¬Truth_teller(henri) ∧ ¬Answer_yes (pierre, gauche))))
5. Answer_yes(pierre, gauche) ↔ ((Truth_teller(pierre) ∧ True(gauche)) ∨ (¬Truth_teller(pierre) ∧ ¬True(gauche))) (Applying the 2nd FOL sentence where x is pierre and q is gauche)
6. Substituting 5 into 4: Answer_yes(henri, dit_oui(pierre, gauche)) ↔ ((Truth_teller(henri) ∧ ((Truth_teller(pierre) ∧ True(gauche)) ∨ (¬Truth_teller(pierre) ∧ ¬True(gauche)))) ∨ (¬Truth_teller(henri) ∧ ¬((Truth_teller(pierre) ∧ True(gauche)) ∨ (¬Truth_teller(pierre) ∧ ¬True(gauche))))))
7. Using the fact that either Henri or Pierre is the truth teller, we simplify the above expression. Case 1: Truth_teller(henri) Answer_yes (henri, dit_oui (pierre, gauche)) ↔ ((True(gauche)) ∨ (False)) ↔ True(gauche) Case 2: ¬Truth_teller(henri) Answer_yes (henri, dit_oui(pierre, gauche)) ↔ ((False) ∨ (¬((¬True(gauche)))) ↔ True(gauche)
8. From the 3rd FOL sentence: True(gauche) ↔ Go_left
9. Therefore, Answer_yes (henri, dit_oui(pierre, gauche)) ↔ Go_left

Therefore, the traveller can ask Henri the question dit_oui (pierre, gauche) to determine if Go_left is true.

(c) KB does not entail which direction to go:

We need to show two interpretations: one where Go_left is true and another where Go_left is false, both satisfying the KB.

- **Interpretation 1: Go_left is True**
  - Domain: {henri, pierre, gauche}
  - Truth_teller(henri) = True
  - Truth_teller(pierre) = False
  - True(gauche) = True
  - Answer_yes(henri, gauche) = True
  - Answer_yes(pierre, gauche) = False
  - Answer_yes(henri, dit_oui(pierre, gauche)) = True
  - Answer_yes(pierre, dit_oui (pierre, gauche)) = False
- **Interpretation 2: Go_left is False**
  - Domain: {henri, pierre, gauche}
  - Truth_teller(henri) = False
  - Truth_teller(pierre) = True
  - True(gauche) = False
  - Answer_yes(henri, gauche) = False
  - Answer_yes(pierre, gauche) = False
  - Answer_yes(henri, dit_oui(pierre, gauche)) = False
  - Answer_yes(pierre, dit_oui(pierre, gauche)) = False

It can be verified that both interpretations satisfy all the FOL sentences in the KB. Since we have interpretations where Go_left is both true and false, the KB does not entail which direction to go.

Activity 2

Read the paper by Palomino et al (2005) and review the 'crossing problem' diagram provided in the Lecturecast.

Create a set of statements in first order logic (FOL) that represent the states shown – for example you may define two functions left and right and therefore the first state could be represented as: Left(F) and left(W) and Left(G) and left(C). Define your own set of FOL statements for the entire diagram.

You have been provided with solutions to the 'crossing problem' written in Lisp (here), Prolog (here) and Maude (here). Compare your FOL clauses with the various implementations – which of the implementations provides the closest match to your FOL version?

Here's a representation of the "crossing problem" states in FOL.

1. Define Objects and Functions:

- Objects: farmer, wolf, goat, cabbage (f, w, g, c)
- Functions:
  - `left(x)`: Represents that object x is on the left bank.
  - `right(x)`: Represents that object x is on the right bank.

2. Define Predicates:

- `State(f, w, g, c)`: Represents the state where the farmer, wolf, goat, and cabbage are on respective banks.

3. FOL State Representation:

Here's how I'd represent the states in FOL:

- **Initial State:** State(left(f), left(w), left(g), left(c))
- **State 1:** State(right(f), left(w), left(g), left(c))
- **State 2:** State(left(f), left(w), right(g), left(c))
- **State 3:** State(right(f), left(w), right(g), left(c))
- **State 4:** State(left(f), right(w), right(g), left(c))
- **State 5:** State(left(f), right(w), left(g), left(c))
- **State 6:** State(right(f), right(w), right(g), left(c))
- **Final State:** State(right(f), right(w), right(g), right(c))

4. Comparison with Implementations:

- **Lisp:** The Lisp implementation uses a list-based representation of the states, which is similar to the FOL representation in terms of explicitly defining the location of each object in each state. However, Lisp uses lists and predicates like `(left f)` while FOL uses functions like `left(f)` within a predicate.
- **Prolog:** The Prolog implementation is very close to the FOL representation. Prolog facts like `state(left,left,left,left,left)` directly correspond to the `State(left(f), left(w), left(g), left(c))` FOL representation. Prolog's declarative style aligns well with the declarative nature of FOL.
- **Maude:** Maude uses a more algebraic approach, defining states as objects with attributes. While it captures the state information, it's structurally different from the FOL representation. Maude is based on rewriting logic, which is different from FOL.

Conclusion:

The Prolog implementation provides the closest match to the FOL representation in this case. Both Prolog and FOL emphasize a declarative approach, representing the states and relationships in a clear and logical manner.