Artificial intelligence - Project 2
- Propositional Logic -

Suster Elena - Roxana

10/12/2020

1 Truth and Lie

A week after these last adventures, Craig was preparing to return to London when he suddenly received a wire from the Transylvania government, urgently requesting him to come to Transylvania to help solve some baffling cases of vampirism. Transylvania is inhabited by both vampires and humans, the vampires always lie and the humans always tell the truth. However, half the inhabitant, both vampires and humans, are insane and totally deluded by their beliefs - all true propositions they believe false and all false propositions they believe true. The other half of inhabitants are completely sane and totally accurate in their judgement - all true statements they know to be true and all false statements they know to be false. So, when a Transylvanian makes a false statement, it could be either out of delusion or out of malice. Sane humans and insane vampires both make true statements, insane humans and sane vampires make only false statements. It was fortunate that Craig was as well versed in vampirism as in logic (the general range of Craig's interest and knowledge was quite remarkable altogether). When he arrived in Transylvania, he was informed by all authorities (all of whom were sane humans) that there were ten cases which they need help, and he was requested to take charge of the investigations. One of them is : "The case of Karl and Martha Dracula". This case involves a pair of twins, Karl and Martha Dracula (no relation to the count, I can assure you!). The interesting think about this case is that not only it was it already known that one of them was human and one vampire, but it was also known that one of the two was sane and the other insane, although Craig had no idea which was which. Here is what they said: Karl : " My sister is a vampire". Martha: "My brother is insane!". Which one is the vampire ?

1.0.1 Code implementation

This sub-section is dedicated to showcasing your own solution that you came up with for solving the above question. One has to put here any **code** that has been used for solving the above task, along with **comments** that explain every design decision made. To reference the code, please make use of the *code lines number*. Additionally, complete this sub-section with any **command configurations** that you may have used during the implementation or testing process (please fill in *just the arguments*).

Code:

```
%sane humans tell the truth
%insane humans tell a lie
%sane vampires tell a lie
%insane vampire tell the truth
%Karl says "My sister is a vampire"
%Martha says " My brother is insane"

formulas(assumptions).

(human(Karl) & -human(Martha)) | (-human(Karl) & human(Martha)).
(sane(Karl) & -sane(Martha)) | ( -sane(Karl) & sane(Martha)).
liar(x) <-> (-sane(x) & human(x)) | (sane(x) & -human(x)).
-liar(x) <-> (sane(x) & human(x)) | (-sane(x) & -human(x)).

-liar(Martha) <-> (-sane(Karl) & sane(Martha)).
-liar(Karl) <-> (-human(Martha)) | (human(Karl)).

end_of_list.

formulas(goals).
-human(Karl).
end_of_list.
```

Explanation:

- We start by supposing that Martha is a vampire. Then Karl is human, and also Karl has made a true statement; hence Karl in this case has to be a sane human. This would make Martha an insane vampire, since, as we have been told, Karl and Martha are different as regards their sanity. But then Martha an insane vampire, since, would have made a false statement - that Karl is insane - which insane vampires cannot do. Therefore, the assumption that Martha is a vampire leads to a contradiction. So it is Karl who is a vampire. We can also determine their sanity or lack of it: Karl has made a false statement; hence, being a vampire, he is sane. But then Martha has also made a false statement; hence, being a human, she is insane. So the complete answer ts that Karl is a sane vampire and Martha an insane human; Karl is lying when he says that his sister is a vampire, and Martha is deluded when she says that her brother is insane.

Commands:

```
prover9 -f transylvania.in
```

2 Puzzles and Metapuzzles

The Island of Questioners
Somewhere in the vast reaches the ocean, there is the Island of Questioners; it name derives from the fact that its inhabitants never make statements; they only ask questions. The inhabitants ask only questions answerable by yes or no. Each inhabitant is one of two types, A and B. Those of type A ask only questions whose correct answer is yes; those of type B ask only questions whose correct answer is no.
One time I met two brothers whose first names were Arthur and Robert. Arthur once asked Robert " Is at least one of us of type B?" What types are Arthur and Robert?

2.0.1 Code implementation

This sub-section is dedicated to showcasing your own solution that you came up with for solving the above question. One has to put here any **code** that has been used for solving the above task, along with **comments** that explain every design decision made. To reference the code, please make use of the *code lines number*. Additionally, complete this sub-section with any **command configurations** that you may have used during the implementation or testing process (please fill in *just the arguments*).

Code:

```
% type A can only ask questions whose answ are YES
% type B can only ask questions whose answ are NO
% Arthur asks Robert "Is at least one of us of type B? "

formulas (assumptions).
(typeA(Arthur) & typeA(Robert)) | (typeA(Arthur) & -typeA(Robert)) | (-typeA(Arthur) & typeA(Robert)) |

typeA(Arthur) -> couldAsk(Robert).
-typeA(Arthur) -> -couldAsk(Robert).

typeA(Robert) & couldAsk(Robert) -> ((-typeA(Arthur) & typeA(Robert)) | (typeA(Arthur) & -typeA(Robert))
typeA(Robert) & -couldAsk(Robert) -> (typeA(Arthur) & -typeA(Robert)).
-typeA(Robert) & couldAsk(Robert) -> (typeA(Arthur) & -typeA(Robert)).
-typeA(Robert) & -couldAsk(Robert) -> ((-typeA(Arthur) & typeA(Robert)) | (typeA(Arthur) & -typeA(Robert
end_of_list.

formulas(goals).
typeA(Arthur) & -typeA(Robert).
end_of_list.
```

Explanation:

-
- We will start by supposing that Arthur were of type B. Then it would be true that at least one of the brothers is of type B, which would make yes the correct way to answer to his question, which would mean he is of type A. This is a contradiction; hence Arthur cannot be of type B, he must be of type A. From this it follows that the correct answer to his question is yes, which means that at least one of the two is of type B. Since Arthur is not of type B, this must be Robert. So Arthur is of type A and Robert is of type B.

Commands:

-
- prover9 -f metapuzzles.in

3 FOL

Ships
Having 5 ships, we know that:
Greek ship leaves at 6 and carries coffee.
The ship in the middle has a black chimney.
The English ship leaves at 9.
The French ship with blue chimney is to the left of the ship that carries coffee.
To the right of the ship carrying cocoa is a ship going to Marseille.
The Brazilian ship is heading to Manila.
Next to the ship carrying rice is a ship with a green chimney.
A ship going to Genova leaves at 5.
The Spanish ship leaves at 7 and is to the right of the ship going to Marseille.
The ship with a red chimney goes to Hamburg.
Next to the ship that leaves at 7 is a ship with white chimney.
The ship on the border carries corn.
The ship with the black chimney leaves at 8.
The ship that carries corn is anchored next to the one carrying rice.
The ship to Hamburg leaves at 6.
Find out which ship carries Tea and which one has the destination Port Said.

3.0.1 Code implementation

This sub-section is dedicated to showcasing your own solution that you came up with for solving the above question. One has to put here any **code** that has been used for solving the above task, along with **comments** that explain every design decision made. To reference the code, please make use of the *code lines number*. Additionally, complete this sub-section with any **command configurations** that you may have used during the implementation or testing process (please fill in *just the arguments*).

Code:

```
%FInding models FOL - EInstein's problem
% Which ship carries tea ? WHich has the destination Port_Said?

%We know that :
% Greek ship leaves at 6 and carries coffee.
% The ship in the middle has a  black chimney.
% The English ship leaves at 9.
% The French ship with blue chimney is to the left of the ship that carries coffee.
% To the right of the ship carrying cocoa is a ship going to Marseille.
% The Brazilian ship is heading to Manila.
% Next to the ship carrying rice is a ship with a green chimney.
% A ship going to Genova leaves at 5.
% The Spanish ship leaves at 7 and is to the right of the ship going to Marseille
% The ship with a red chimney goes to Hamburg.
% Next to the ship that leaves at 7 is a ship woth white chimney.
% The ship on the border carries corn
% The ship with the black chimney leaves at 8
% The ship that carries corn is anchored next to the one carrying rice
% The ship to Hamburg leaves at 6

set (arithmetic). % for "right neighbour"/ "left neighbour" relation
assign (domain_size, 5). % there are 5 ships {0, 1, 2, 3, 4}

list(distinct). %Objects in each list are distinct
```

```
25
26  [Greek, English, Spanish, French, Brazilian]. % nationalities are distinct
27  [Black, Blue, Green, White, Red]. % chimney colour is distinct
28  [Six, Nine, Five, Seven, Eight]. %depature hour is distinct
29  [Hamburg, Manila, Genova, Marseille, Port_Said]. %destination is distinct
30  [Tea, Coffee, Cocoa, Rice, Corn]. %cargo is distinct
31  end_of_list.
32
33  formulas(assumptions).
34
35  right_neighbour(x,y) <-> x < y.
36  left_neighbour(x,y) <-> x > y.
37  middle(x) <-> x = 2.
38  border(x) <-> x = 0 | x = 4.
39  neighbour(x,y) <-> right_neighbour(x,y) | left_neighbour(x,y).
40
41  Greek = Six.
42  Greek = Coffee.
43  middle(Black).
44  English = Nine.
45  French = Blue.
46  left_neighbour(Coffee, French).
47  right_neighbour(Cocoa, Marseille).
48  Brazilian = Manila.
49  neighbour(Rice, Green).
50  Genova = Five.
51  Spanish = Seven.
52  right_neighbour(Marseille, Spanish).
53  Hamburg = Red.
54  neighbour(Seven, White).
55  border(Corn).
56  Black = Eight.
57  neighbour(Corn, Rice).
58  Hamburg = Six.
59
60  end_of_list.
```

Explanation:

-
- We start by declaring our list of elements which compound the identification elements for each of our ships. The identification elements are : the nationality of the ship (where the ship is from), the destination (where does it go), the departure hour, chimney colour and what it carries. The clues also gives us some arithmetic relations regarding the neighbour of one specific ship, such that in the assumptions we will formulate the neighbours relationships using the domain size (in our case being 5 -> 5 ships).
  After declaring all the relations needed we are going to associate to them the relations presented in the text regarding the position of each ship.

Commands:

-
- mace4 -c -f ship.in | interpformat

4 Skolemization

The following puzzle is one which will include the key words "all" and "exists", which would categorise it in the same class with the "Aunt Agatha" problem or "Curiosity cat" problem. The text of this puzzle will be: Everyone who loves dogs is lucky.
Anyone who hurts a dog cannot be lucky
Vlad loves all dogs.
Either Vlad or Maria hurts the German Shepherd called Irod.
The German Shepherd is a dog.
Did Maria hurt the German Shepherd?

4.0.1 Code implementation

This sub-section is dedicated to showcasing your own solution that you came up with for solving the above question. One has to put here any **code** that has been used for solving the above task, along with **comments** that explain every design decision made. To reference the code, please make use of the *code lines number*. Additionally, complete this sub-section with any **command configurations** that you may have used during the implementation or testing process (please fill in *just the arguments*).

Code:

```
% Everyone who loves dogs is  lucky.
% Anyone who hurts a dog cannot be luck.y
% Vlad loves all dogs.
% Either Vlad or Maria hurts the German Shepherd called Irod.
% The German Shepherd is a dog.
% Did Maria hurt the German Shepherd?

set (binary_resolution).

formulas (assumptions).

all x (all y (Dog(y) -> Loves(x,y)) -> Lucky(x)).
all x (exists z (Dog(z) & Hurt(x,z)) -> -Lucky(x)).
all x (Dog(x) -> Loves(Vlad,x)).
Hurt(Vlad, Irod) | Hurt(Maria, Irod).
German_Shepherd(Irod).
all x (German_Shepherd(x) -> Dog(x)).
end_of_list.

formulas(goals).
Hurt(Maria, Irod).
end_of_list.
```

Explanation:

First, we will identify the constants of our problem, which are : Vlad, Maria and Irod; and the predicates: Dog(x) : x is a dog, Love(x,y) : x loves y, Hurt(x,y) : x hurts y and Lucky(x) : x is lucky.
After identifying them, we will rewrite the relations which constitute the problems, taking into consideration the fact that prover9 uses all and exists for the logical and respectively. By default, variables in a clause are universally quantified, so all could be omitted.

Commands:

-
- prover9 -f dogs.in