Artificial intelligence - Project 2
- Constraint Satisfaction Problems. Propositional Logic -

Alexandru Madalina-Elena

13/12/2021

1 First Problem - Who Is Mad?

Shortly after the trial, Alice met the Duchess, and they had the following remarkable conversation.

"The Cheshire Cat says that everyone here is mad," said Alice. "Is that really true?"

"Of course not," replied the Duchess. "If that were really true, then the Cat would also be mad, hence you could not rely on what it said."

This sounded perfectly logical to Alice.

"But I'll tell you a great secret, my dear," continued the Duchess. "Half the creatures around here are mad—totally mad!"

"That doesn't surprise me," said Alice, "many have seemed quite mad to me!"

"When I said totally mad," continued the Duchess, quite ignoring Alice's remark, "I meant exactly what I said: They are completely deluded! All their beliefs are wrong—not just some, but all. Everything true they believe to be false and everything false they believe to be true."

Alice thought awhile about this very queer state of affairs. "Does a mad person or creature believe that two plus two equals five?" asked Alice.

"Why, of course, child! Since two plus two doesn't equal five, then a mad person naturally believes that it does."

"And does a mad person also believe that two plus two equals six?"

"Of course," replied the Duchess, "since it doesn't, then the mad one believes it does."

"But it can't both equal five and six!" exclaimed Alice.

"Of course not," agreed the Duchess, "you know that and I know that, but a mad person doesn't. And the moral of that is—"

"What about the sane people around here?" interrupted Alice (who had heard quite enough morals for the day). "I guess most of their beliefs are right but some of them are wrong?"

"Oh, no, no!" said the Duchess most emphatically. "That may be true where you come from, but around here the sane people are one hundred percent accurate in their beliefs! Everything true they know to be true and everything false they know to be false."

Alice thought this over. "Which ones around here are sane and which ones are mad?" asked Alice. "I would very much like to know this."

1.1 Problems and solutions

1.1.1 Code implementation

Code:

```
assign(domain_size, 31).

list(distinct).
        [catterpillar, lizard, cook, cat, fish_footman, frog_footman, king_of_diamonds,
        queen_of_diamonds, hatter, hare, dormouse, lobster, mock_turtle, gryphon,
        king_of_spades, queen_of_spades, king_of_clubs, queen_of_clubs, queen_of_hearts,
        king_of_hearts, dodo, lory, eaglet, knave_of_hearts, one, two, three, four, five,
        six, seven].
end_of_list.

formulas(assumptions).
        sane(x) | mad(x).
        sane(x) <-> -mad(x).
end_of_list.
```

Explanation:

- I established two rules by which every other solved exercise will abide.
- The domain contains the 31 characters involved in these problems.

Commands:

- mace4 -c -m -1 -f who_is_mad.in | interpformat
- mace4 -c -f who_is_mad.in | interpformat

1.2 Question 1 - The Caterpillar and the Lizard

"Well," replied the Duchess, "take, for example, the Caterpillar and Bill the Lizard. The Caterpillar believes that both of them are mad."

"Which of them is really mad?" asked Alice.

"I shouldn't have to tell you that!" replied the Duchess. "I have given you enough information for you to deduce the answer."

What is the solution? Is the Caterpillar mad or sane? And what about the Lizard?

1.2.1 Code implementation

Code:

```
sane(catterpillar) -> mad(catterpillar) & mad(lizard).
mad(catterpillar)  -> sane(catterpillar) | sane(lizard).
```

Explanation:

- The statement as it was told by the Caterpillar for the case in which the Caterpillar is sane, and its opposite for the case in which the Caterpillar is mad.
- The Caterpillar is mad and the Lizard is sane.

1.3 Question 2 - The Cook and the Cat

"Then there's my cook and the Cheshire Cat," continued the Duchess. "The Cook believes that at least one of the two is mad."

What can you deduce about the Cook and the Cat?

1.3.1 Code implementation

Code:

```
sane(cook) -> mad(cat).
mad(cook) -> sane(cook) & sane(cat).
```

Explanation:

- The statement as it was told by the Cook for the case in which the Cook is sane, and its opposite for the case in which the Cook is mad.
- The Cook is sane and the Cheshire Cat is mad.

1.4 Question 3 - The Fish-Footman and the Frog-Footman

"That was very interesting," said Alice. "The two cases are quite different."

"Of course they are, my dear! And the moral of that is—to be or not to be is not the same as to be and not to be."

Alice tried to figure out just what the Duchess had meant, when the Duchess interrupted her thoughts.

"Then there are my two footmen, the Fish-Footman and the Frog-Footman. You've met them?"

"Oh, yes, indeed!" said Alice, remembering the latter's unspeakable rudeness.

"Well, the Fish-Footman believes that he and the Frog-Footman are alike—in other words that they are either both sane or both mad. And now, my dear, it is up to you to tell me which ones are mad."

Alice did not quite see why it should be up to her. Still, the puzzle interested her, so she worked on it quite a while.

"I'm afraid I can't solve it," said Alice, "I know what one of the footmen is, but I cannot figure out the other."

"Why, you have solved it, you dear thing!" said the Duchess, giving Alice a hug. "The other footman can't be figured out from what I have told you. In fact, even I don't know what the other one is."

Which footman do you know to be sane or mad, and what is he?

1.4.1 Code implementation

Code:

```
sane(fish_footman) -> (sane(fish_footman) & sane(frog_footman)) |
                                            (mad(fish_footman) & mad(frog_footman)).
mad(fish_footman) -> (mad(fish_footman) | mad(frog_footman)) &
                                           (sane(fish_footman) | sane(frog_footman)).
```

Explanation:

- The statement as it was told by the Fish-Footman for the case in which the Fish-Footman is sane, and its opposite for the case in which the Fish-Footman is mad.
- The Fish-Footman can be either sane or mad, the Frog-Footman is sane.

1.5 Question 4 - The King and Queen of Diamonds

"Then there's the King and Queen of Diamonds," began the Duchess.

"The King and Queen of Diamonds?" said Alice. "I don't believe I've met them—in fact I did not know they were around here."

"All the cards are around here," said the Duchess. "Anyway, I heard a rumor that the Queen of Diamonds was mad. However, I was not sure whether the person who told me this was mad or sane, so I decided to find out for myself.

"Well, one day I met the King of Diamonds without his Queen. I knew him to be absolutely honest, though of doubtful sanity, hence whatever he would say he would at least believe to be true.

'"Is your poor dear wife really mad?' I asked sympathetically."

'"She believes she is,' replied the King."

What can be deduced about the King and Queen of Diamonds?

1.5.1 Code implementation

Code:

```
mad(king_of_diamonds) -> sane(queen_of_diamonds) |
                         mad(queen_of_diamonds).
sane(king_of_diamonds) -> -sane(queen_of_diamonds) &
                          -mad(queen_of_diamonds).
```

Explanation:

- The statement as it was told by the King of Diamonds for the case in which the king is sane, and its opposite for the case in which the king is mad.
- The Queen of Diamonds can be wither mad or sane, the King of Diamonds is mad.

1.6 Question 5 - What About These Three?

"I've always wondered about the March Hare, the Hatter, and the Dormouse," said Alice. "The Hatter is called the Mad Hatter, but is he really mad? And what about the March Hare and the Dormouse?"

"Well," replied the Duchess, "the Hatter once expressed the belief that the March Hare does not believe that all three of them are sane. Also, the Dormouse believes that the March Hare is sane."

What can you deduce about these three?

1.6.1 Code implementation

Code:

```
mad(hatter) -> mad(hare).
sane(hatter) -> sane(hare).

sane(hare) -> mad(hatter) | mad(dormouse).
mad(hare) -> (mad(hare) & mad(hatter) & mad(dormouse)).

sane(dormouse) -> sane(hare).
mad(dormouse) -> mad(hare).
```

Explanation:

- I included both cases for each character, sane or mad.
- The sane case is represented by the statement as the character tells it, and the mad case represents the opposite.
- All three are mad.

1.7 Question 6 - And These Three?

"Then there's the Gryphon, the Mock Turtle, and the Lobster," the Duchess began.
"I didn't know there was a real lobster around here," replied Alice. "I only know him in a poem."
"Oh, yes, there is a real lobster, and he is as big as the Mock Turtle," replied the Duchess. "Anyway, the Lobster once expressed the belief that the Gryphon believes that exactly one of the three is sane. The Mock Turtle believes that the Gryphon is sane."
What can you deduce about these three?

1.7.1 Code implementation

Code:

```
sane(lobster) -> sane(gryphon).
mad(lobster) -> sane(gryphon).

(sane(gryphon) & mad(lobster) & mad(mock_turtle)) |
(mad(gryphon) & sane(lobster) & mad(mock_turtle)) |
(mad(gryphon) & mad(lobster) & sane(mock_turtle)) -> sane(lobster).
(mad(gryphon) & mad(lobster) & mad(mock_turtle)) |
(sane(gryphon) & sane(lobster) & sane(mock_turtle)) |
(mad(gryphon) & sane(lobster) & sane(mock_turtle)) |
(sane(gryphon) & mad(lobster) & sane(mock_turtle)) |
(sane(gryphon) & sane(lobster) & mad(mock_turtle)) -> mad(lobster).

sane(mock_turtle) -> sane(gryphon).
mad(mock_turtle) -> mad(gryphon).
```

Explanation:

- The Lobster and the Gryphon must be of the same type. (mad/sane)
- The Mock Turtle and the Gryphon must be of the same type. (mad/sane)
- If exactly one of the three characters is sane, then the Lobster must be sane. In any other case, the Lobster is mad.
- The Lobster is mad, the Gryphon is sane, and the Mock Turtle is sane.

1.8 Question 7 - And Now, What About These Two?

"You know," said Alice in a very low voice, looking around to see that the Queen of Hearts was not within hearing distance, "I am particularly interested in knowing about the King and Queen of Hearts. What are they?"

"Ah," said the Duchess, "this is an interesting story indeed! The Queen believes that the King believes that the Queen believes that the King believes that the Queen is mad."

"Now that's too much!" cried Alice. "I think I'll go mad if I try to puzzle that one out!"

"Very well," said the Duchess good-naturedly, "let's try an easier one first. For example, take the King and Queen of Spades."

There was a long pause.

"What about the King and Queen of Spades?" asked Alice.

"Well, the Queen believes that the King believes that she is mad.

What can you tell me about the King and Queen of Spades?"

1.8.1 Code implementation

Code:

```
sane(queen_of_spades) -> mad(king_of_spades).
mad(queen_of_spades) -> mad(king_of_spades).

sane(king_of_spades) -> mad(queen_of_spades).
```

Explanation:

- Whether the Queen of Spades is mad or sane, the king must be mad.
- If the King of Spades is sane, then the queen must be mad.
- The Queen of Spades can be either sane or mad, but the King is mad.

1.9 Question 8 - The King and Queen of Clubs

"You got that one pretty easily," said the Duchess. "Now, what would you say if I told you that the King of Clubs believes that the Queen of Clubs believes that the King of Clubs believes that the Queen of Clubs is mad?"

1.9.1 Code implementation

Code:

```
mad(queen_of_clubs) -> mad(king_of_clubs).
sane(queen_of_clubs) -> mad(king_of_clubs).

mad(king_of_clubs) -> sane(queen_of_clubs).
```

Explanation:

- Wheter the Queen of Clubs is sane or mad, the King must be mad.
- If the King of Clubs is mad, then the queen must be sane.
- The Queen of Clubs and the King of Clubs are both mad.

1.10 Question 9 - And Now, What About the Queen of Hearts?

Alice thought this last puzzle over and said, "If you had told me that (which of course you didn't) then I'm afraid I would have had to conclude that you must be mad!"

"And quite right you would be!" cried the Duchess. "But of course I would never tell you any such

impossible thing!

"And now," continued the Duchess, "you should be able to solve the puzzle about the King and Queen of Hearts. Remember what I told you: The Queen believes that the King believes that the Queen believes that the King believes that she is mad. The question is, Is the Queen of Hearts mad or sane?"

1.10.1 Code implementation

Code:

```
sane(king_of_hearts) | mad(king_of_hearts) -> mad(queen_of_hearts).
```

Explanation:

- Whether the king is mad or sane, the queen must be mad.
- The Queen of Hearts is mad, and the King of hearts can be mad or sane.

1.11 Question 10 - The Dodo, the Lory, and the Eaglet

"Then there's the Dodo, the Lory, and the Eaglet," said the Duchess. "The Dodo believes that the Lory believes that the Eaglet is mad. The Lory believes that the Dodo is mad, and the Eaglet believes that the Dodo is sane.

"Can you puzzle that out?" asked the Duchess.

1.11.1 Code implementation

Code:

```
mad(eaglet) -> sane(dodo) & sane(lory).
sane(eaglet) -> sane(dodo) & mad(lory).

sane(lory) -> mad(dodo).
mad(lory) -> sane(dodo).

sane(eaglet) -> sane(dodo).
mad(eaglet) -> mad(dodo).
```

Explanation:

- If the Eaglet is sane, then his statement must be true. Otherwise, the opposite must be true.
- The Eaglet and the Dodo must be of the same type.
- The Lory and the Dodo must be of opposite types.
- The Eaglet is sane, the Dodo is sane, and the Lory is mad.

1.12 Question 12 - The Knave of Hearts

Alice solved this last puzzle.

"I think I know why half the people around here are mad," said Alice.

"Why?" asked the Duchess.

"I think they went mad trying to work out puzzles like these. They're dreadfully confusing!"

"As to confusing puzzles," replied the Duchess, "these are nothing compared to some I could tell you if I chose!"

"Oh, you needn't choose!" said Alice as politely as she could.

"For example, there's the Knave of Hearts," the Duchess went on, "he keeps company with the Spade-Gardeners, One, Two, Three, Four, Five, Six, and Seven. I believe you've met Two, Five, and Seven?"

"Oh, yes," remembered Alice, "they were having a terrible time trying to paint the white roses red, because they had by mistake planted a white rose tree in the garden instead of a red rose tree as the Queen had ordered."

"Well," said the Duchess, "Three believes that One is mad. Four believes that Three and Two are not both mad. Five believes that One and Four are either both mad or both sane. Six believes that One and Two are both sane. Seven believes that Five is mad. As for the Knave of Hearts, he believes that Six and Seven are not both mad.

"And now," continued the Duchess, "would you care to figure out whether the Knave is mad or sane, or would you prefer a more confusing puzzle?"

"Oh, no," replied poor Alice, "this one is quite confusing enough, thank you!"

Is the Knave of Hearts mad or sane?

1.12.1 Code implementation

Code:

```
sane(three) -> mad(one) & mad(four).
mad(three) -> sane(one) & sane(four).

sane(four) -> sane(three) | sane(two).
mad(four) -> (mad(three) & mad(two)).

sane(five) -> (mad(one) & mad(four)) |
               (sane(one) & sane(four)).
mad(five) -> (sane(one) & mad(four)) |
              (mad(one) & sane(four)).

sane(six) -> sane(one) & sane(two).
mad(six) -> (sane(one) & mad(two)) |
            (mad(one) & mad(two)) |
                        (mad(one) & sane(two)).

sane(seven) -> mad(five).
mad(seven) -> sane(five).

sane(knave_of_hearts) -> sane(six) | sane(seven).
mad(knave_of_hearts) -> mad(six) & mad(seven).
```

Explanation:

- Analyzed each character's statement for both cases.
- The knave of hearts is sane.

1.12.2 Personal observations and notes

- I added the second half of the first two conditions. Otherwise I would have had 32 final models with mace4.

1.13 Conclusion and Observations

- I chose to create various models with mace4, so that I can track the values for each character introduced in the story.
- Therefore, there are 16 resulted models, because of the 4 characters that can be both mad or sane. Of course, in order to visualize only one model you can just run the file who_is_mad.in with the second command mentioned previously.

1.14 References

- https://biblioteca.utcluj.ro/files/carti-online-cu-coperta/290-8.pdf
- *Alice in Puzzle-land*, RAYMOND SMULLYAN

2 Second Problem - Isle of Dreams

I once dreamed that there was a certain island called the Isle of Dreams. The inhabitants of this island dream quite vividly; indeed, their thoughts while asleep are as vivid as while awake. Moreover, their dream life has the same continuity from night to night as their waking life has from day to day. As a result, some of the inhabitants sometimes have difficulty in knowing whether they are awake or asleep at a given time.

Now, it so happens that each inhabitant is of one of two types: diurnal or nocturnal. A diurnal inhabitant is characterized by the fact that everything he believes while he is awake is true, and everything he believes while he is asleep is false. A nocturnal inhabitant is the opposite: everything a nocturnal person believes while asleep is true, and everything he believes while awake is false.

2.1 Problems and solutions

2.1.1 Code implementation

Code:

```
formulas(assumptions).
        diurnal(x) | nocturnal(x).
        diurnal(x) <-> -nocturnal(x).
        noctural(x) <-> -diurnal(x).

        awake(x) | asleep(x).
        awake(x) <-> -asleep(x).
        asleep(x) <-> -awake(x).
end_of_list.
```

Explanation:

- This time I chose Prover9 in order to solve separate tasks. I will include for each problem the rest of the assumptions and the goals in the code.
- The general rules are that every person must be either diurnal or nocturnal and cannot be both at the same time, and that the same goes for the awake/asleep states.
- For every problem the code usually takes into consideration all of the four possibilities for the state of an inhabitant:
  - diurnal and awake
  - diurnal and asleep
  - nocturnal and awake
  - nocturnal and asleep

Commands:

- prover9 -f isle_of_dreams.in > isle_of_dreams.out

2.2 Question 1

At one particular time, one of the inhabitants believed that he was of the diurnal type.

Can it be determined whether his belief was correct? Can it be determined whether he was awake or asleep at the time?

2.2.1 Code implementation

Code:

```
formulas(assumptions).
        diurnal(i1) & awake(i1) -> diurnal(i1).
        nocturnal(i1) & awake(i1) -> nocturnal(i1).
        diurnal(i1) & asleep(i1) -> nocturnal(i1).
        nocturnal(i1) & asleep(i1) -> diurnal(i1).
end_of_list.

formulas(goals).
        diurnal(i1).
        nocturnal(i1).
        awake(i1).
        asleep(i1).
end_of_list.
```

Explanation:

- The type of the inhabitant cannot be determined.
- He must be awake.

2.3 Question 2

On another occasion, one of the natives believed he was asleep at the time. Can it be determined whether his belief was correct? Can it be determined what type he is?

2.3.1 Code implementation

Code:

```
formulas(assumptions).
        nocturnal(i2) & asleep(i2) -> asleep(i2).
        nocturnal(i2) & awake(i2) -> awake(i2).
        diurnal(i2) & awake(i2) -> asleep(i2).
        diurnal(i2) & asleep(i2) -> awake(i2).
end_of_list.

formulas(goals).
        diurnal(i2).
        nocturnal(i2).
        awake(i2).
        asleep(i2).
end_of_list.
```

Explanation:

- He is nocturnal.
- Whether he is asleep or not cannot be determined.

2.4 Question 3

At one time, an inhabitant believed that she was either asleep or of the nocturnal type, or both. (Or means at least one or possibly both.)

Can it be determined whether she was awake or asleep at the time? Can it be determined what type she is?

2.4.1 Code implementation

Code:

```
formulas(assumptions).
        nocturnal(i3) & asleep(i3) -> asleep(i3) | nocturnal(i3).
        diurnal(i3) & awake(i3) -> asleep(i3) | nocturnal(i3).
        nocturnal(i3) & awake(i3) -> awake(i3) & diurnal(i3).
        diurnal(i3) & asleep(i3) -> awake(i3) & diurnal(i3).
end_of_list.

formulas(goals).
        diurnal(i3).
        nocturnal(i3).
        awake(i3).
        asleep(i3).
end_of_list.
```

Explanation:

- The inhabitant was asleep and nocturnal.

2.5 Question 4

At one time, an inhabitant believed that he was both asleep and diurnal. What was he really?

2.5.1 Code implementation

Code:

```
formulas(assumptions).
        nocturnal(i4) & asleep(i4) -> asleep(i4) & diurnal(i4).
        diurnal(i4) & awake(i4) -> asleep(i4) & diurnal(i4).
        nocturnal(i4) & awake(i4) -> awake(i4) | nocturnal(i4).
        diurnal(i4) & asleep(i4) -> awake(i4) | nocturnal(i4).
end_of_list.

formulas(goals).
        diurnal(i4).
        nocturnal(i4).
        awake(i4).
        asleep(i4).
end_of_list.
```

Explanation:

- The inhabitant was awake and nocturnal.

2.6 Question 5

There is a married couple on this island whose last name is Kulp. At one point Mr. Kulp believed that he and his wife were both nocturnal. As the same instant, Mrs. Kulp believed that they were not both nocturnal. As it happened, one of them was awake and one of them was asleep at the time. Which one of them was awake?

2.6.1 Code implementation

Code:

```
formulas(assumptions).
        nocturnal(mr_kulp) & asleep(mr_kulp) -> nocturnal(mr_kulp) & nocturnal(mrs_kulp).
        diurnal(mr_kulp) & awake(mr_kulp) -> nocturnal(mr_kulp) & nocturnal(mrs_kulp).
        nocturnal(mr_kulp) & awake(mr_kulp) -> diurnal(mr_kulp) | diurnal(mrs_kulp).
        diurnal(mr_kulp) & asleep(mr_kulp) -> diurnal(mr_kulp) | diurnal(mrs_kulp).

        nocturnal(mrs_kulp) & asleep(mrs_kulp) -> (nocturnal(mr_kulp) & diurnal(mrs_kulp)) |
                                                  (nocturnal(mrs_kulp) & diurnal(mr_kulp)) |
                                                  (diurnal(mr_kulp) & diurnal(mrs_kulp)).
        diurnal(mrs_kulp) & awake(mrs_kulp) -> (nocturnal(mr_kulp) & diurnal(mrs_kulp)) |
                                               (nocturnal(mrs_kulp) & diurnal(mr_kulp)) |
                                               (diurnal(mr_kulp) & diurnal(mrs_kulp)).
        nocturnal(mrs_kulp) & awake(mrs_kulp) -> nocturnal(mr_kulp) & nocturnal(mrs_kulp).
        diurnal(mrs_kulp) & asleep(mrs_kulp) -> nocturnal(mr_kulp) & nocturnal(mrs_kulp).

        (awake(mrs_kulp) & asleep(mr_kulp)) | (awake(mr_kulp) & asleep(mrs_kulp)).
end_of_list.

formulas(goals).
        diurnal(mr_kulp).
        diurnal(mrs_kulp).

        nocturnal(mr_kulp).
        nocturnal(mrs_kulp).

        awake(mr_kulp).
        awake(mrs_kulp).

        asleep(mr_kulp).
        asleep(mrs_kulp).
end_of_list.
```

Explanation:

- Besides the four cases mentioned for both Mr. and Mrs. Kulp, there is another condition: one of them was asleep and one of them was awake.
- They are either both nocturnal, or both diurnal. Mr. Kulp was asleep and Mrs. Kulp was awake.

2.7 Question 6

There is another married couple on this isle whose last name is Byron. One of them is nocturnal and the other is diurnal. At one point the wife believed that they were either both asleep or both awake. At the same instant, the husband believed that they were neither both asleep not both awake.

Which one was right?

2.7.1 Code implementation

Code:

```
formulas(assumptions).
        nocturnal(mr_byron) & asleep(mr_byron) -> (asleep(mrs_byron) & awake(mr_byron)) |
                                                  (awake(mrs_byron) & asleep(mr_byron)).
```

```
        diurnal(mr_byron) & awake(mr_byron) -> (asleep(mrs_byron) & awake(mr_byron)) |
                                               (awake(mrs_byron) & asleep(mr_byron)).
        nocturnal(mr_byron) & awake(mr_byron) -> (asleep(mrs_byron) & asleep(mr_byron)) |
                                                 (awake(mrs_byron) & awake(mr_byron)).
        diurnal(mr_byron) & asleep(mr_byron) -> (asleep(mrs_byron) & asleep(mr_byron)) |
                                                (awake(mrs_byron) & awake(mr_byron)).

        nocturnal(mrs_byron) & asleep(mrs_byron) -> (asleep(mrs_byron) & asleep(mr_byron)) |
                                                    (awake(mrs_byron) & awake(mr_byron)).
        diurnal(mrs_byron) & awake(mrs_byron) -> (asleep(mrs_byron) & asleep(mr_byron)) |
                                                 (awake(mrs_byron) & awake(mr_byron)).
        nocturnal(mrs_byron) & awake(mrs_byron) -> (asleep(mrs_byron) & awake(mr_byron)) |
                                                   (awake(mrs_byron) & asleep(mr_byron)).
        diurnal(mrs_byron) & asleep(mrs_byron) -> (asleep(mrs_byron) & awake(mr_byron)) |
                                                  (awake(mrs_byron) & asleep(mr_byron)).

        (diurnal(mr_byron) & nocturnal(mrs_byron)) | (diurnal(mrs_byron) & nocturnal(mr_byron)).
end_of_list.

formulas(goals).
        diurnal(mr_byron).
        diurnal(mrs_byron).

        nocturnal(mr_byron).
        nocturnal(mrs_byron).

        awake(mr_byron).
        awake(mrs_byron).

        asleep(mr_byron).
        asleep(mrs_byron).
end_of_list.
```

Explanation:

- We know the four cases for each Byroun spouse, but we also know that one of them must be nocturnal and the other diurnal.
- They were both asleep or both awake. Therefore, Mrs. Byron was right.

2.8 Question 7

Here is a particularly interesting case; at one time an inhabitant named Edward believed amazingly that he and his sister Elaine were both nocturnal, and at the same time that he was not nocturnal.

How is this possible? Is he nocturnal or diurnal? What about his sister? Was he awake or asleep at the time?

2.8.1 Code implementation

Code:

```
formulas(assumptions).
        nocturnal(edward) & asleep(edward) -> (nocturnal(edward) & nocturnal(elaine)) &
                                              (diurnal(edward)).
        diurnal(edward) & awake(edward) -> (nocturnal(edward) & nocturnal(elaine)) &
                                           (diurnal(edward)).
```

```
        nocturnal(edward) & awake(edward) -> (diurnal(edward) | diurnal(elaine)) &
                                             (nocturnal(edward)).
        diurnal(edward) & asleep(edward) -> (diurnal(edward) | diurnal(elaine)) &
                                            (nocturnal(edward)).
end_of_list.

formulas(goals).
        nocturnal(edward).
        diurnal(edward).
        awake(edward).
        asleep(edward).

        diurnal(elaine).
        nocturnal(elaine).
end_of_list.
```

Explanation:

- Edward is nocturnal and awake.
- Elaine is diurnal.

2.9 Question 8 - The Royal Family

The isle has a king and a queen and also a princess. At one point the princess believed that her parents were of different types. Twelve hours later, she changed her state(either from sleeping to waking or from waking to sleeping), and she then believed that her father was diurnal and her mother was nocturnal.

What type is the king and what type is the queen?

2.9.1 Code implementation

Code:

```
formulas(assumptions).
        nocturnal(princess1) & asleep(princess1) -> (nocturnal(queen) & diurnal(king)) |
                                                    (nocturnal(king) & diurnal(queen)).
        diurnal(princess1) & awake(princess1) -> (nocturnal(queen) & diurnal(king)) |
                                                 (nocturnal(king) & diurnal(queen)).
        nocturnal(princess1) & awake(princess1) -> (nocturnal(queen) & nocturnal(king)) |
                                                   (diurnal(king) & diurnal(queen)).
        diurnal(princess1) & asleep(princess1) -> (nocturnal(queen) & nocturnal(king)) |
                                                  (diurnal(king) & diurnal(queen)).

        nocturnal(princess2) & asleep(princess2) -> diurnal(king) & nocturnal(queen).
        diurnal(princess2) & awake(princess2) -> diurnal(king) & nocturnal(queen).
        nocturnal(princess2) & awake(princess2) -> nocturnal(king) | diurnal(queen).
        diurnal(princess2) & asleep(princess2) -> nocturnal(king) | diurnal(queen).

        (nocturnal(princess1) & asleep(princess1)) |
        (nocturnal(princess2) & asleep(princess2)) |
        (diurnal(princess1) & awake(princess1)) |
        (diurnal(princess2) & awake(princess2)).

        (nocturnal(princess1) & awake(princess1)) |
        (nocturnal(princess2) & awake(princess2)) |
        (diurnal(princess1) & asleep(princess1)) |
```

```
        (diurnal(princess2) & asleep(princess2)).

        (asleep(princess1) & awake(princess2)) | (asleep(princess2) & awake(princess1)).
end_of_list.

formulas(goals).
        nocturnal(king).
        diurnal(king).

        nocturnal(queen).
        diurnal(queen).
end_of_list.
```

Explanation:

- Princess1 and princess2 are variables used to depict the two instances in time of the princess thinking about her parents type.
- Since one of her thoughts must be true, and the other must be false, one of the instances of princess must be telling the truth, therefore she must be nocturnal and asleep or diurnal and wake, and one of the ininstances of princess must be lying, therfore she must be nocturnal and awake or diurnal and asleep.
- Another thing I specified is that they must be of different states: one awake and one asleep.
- The king is nocturnal and the queen is diurnal.

2.10 Conclusion and Observations

- I chose to test the state and type of every character separately using Prover9.
- Everything from the goal list can be proven to be true or untrue because of search failure.

2.11 References

- https://biblioteca.utcluj.ro/files/carti-online-cu-coperta/290-8.pdf
- *The Lady or the Tiger And Other Logic Puzzles*, RAYMOND SMULLYAN

3 Third Problem - The King's Story

"Very well then," said the King. "All the cases take place in a land far, far away—a very strange land inhabited exclusively by knights who always tell the truth, and knaves who always lie—
" "Oh, I know those puzzles!" said Alice.
"Now, I declare," said the King quite angrily, "you should never say you know those puzzles until you have heard what the puzzles are! There are countless puzzles about liars and truth-tellers, child, and the odds are a million to one you don't know these puzzles!

3.1 Problems and solutions

3.1.1 Code implementation

Code:

```
assign(domain_size, 18).

list(distinct).
        [a, b, c, a1, b1, c1, a2, b2, c2, a3, b3, c3, a4, b4,
        c4, a5, b5, c5].
end_of_list.

formulas(assumptions).
        knave(x) -> -knight(x) & -spy(x).
        knight(x) -> -knave(x) & -spy(x).
          spy(x) -> -knight(x) & -knave(x).

        knave(x) | knight(x) | spy(x).
end_of_list.
```

Explanation:

- There are some general rules to all of the problems: if someone is a knight, they cannot be a spy or a knave, and so on.
- I designed the solution to these problems using mace4.
- The result by running the code is one unique model.
- Each of the implementations are based on the initial rules: knights are truth-tellers, knaves are liars and spies sometimes lie and sometimes tell the truth.

Commands:

- mace4 -c -m -1 -f the_kings_story.in | interpformat

3.2 Question 1 - Enter the First Spy

"At the interrogation, A claimed that C was a knave, and B claimed that A was a knight. Then C was asked what he was, and C replied, 'I am the spy.'"
Which one was the spy, which one was the knight, and which one was the knave?

3.2.1 Code implementation

Code:

```
formulas(assumptions).
        (knave(a) & knight(b) & spy(c)) |
        (knave(a) & spy(b) & knight(c)) |
```

```
        (knight(a) & knave(b) & spy(c)) |
        (knight(a) & spy(b) & knave(c)) |
        (spy(a) & knight(b) & knave(c)) |
        (spy(a) & knave(b) & knight(c)).

        knight(a) -> knave(c).
        knave(a) -> knight(c) | spy(c).
        spy(a) -> knave(c) | knight(c) | spy(c).

        knight(b) -> knight(a).
        knave(b) -> knave(a) | spy(a).
        spy(b) -> knight(a) | knave(a) | spy(a).

        spy(c) | knave(c).
end_of_list.
```

Explanation:

- A, B and C must be different.
- A is the knight, B is the spy, and C is the knave.

3.3 Question 2 - The Case of the Foxy Spy

"First A said, 'I am not a spy.' Then B said, 'I am a spy.' Then C was asked, 'Is B really a spy?'
"Now, it so happened that C was the spy. Being a spy, he can either lie or tell the truth as he chooses. Well, he did the foxiest thing possible and answered in such a way as not to convict himself."

3.3.1 Code implementation

Code:

```
formulas(assumptions).
        (knave(a1) & knight(b1) & spy(c1)) |
        (knave(a1) & spy(b1) & knight(c1)) |
        (knight(a1) & knave(b1) & spy(c1)) |
        (knight(a1) & spy(b1) & knave(c1)) |
        (spy(a1) & knight(b1) & knave(c1)) |
        (spy(a1) & knave(b1) & knight(c1)).

        knight(a1) | spy(a1).

        knave(b1) | spy(b1).

        spy(c1).
end_of_list.
```

Explanation:

- A, B and C must be different.
- C is the spy, B is the knight and A is the knave.

3.4 Question 3 - Who Is Murdoch?

"Another spy by the name of Murdoch entered the land.
He is one of A, B, C, and one of the three is a knight and the other a knave. The spy is the only one of the three named Murdoch.

The three made the following statements in court:
A: My name is Murdoch.
B:That is true.
C: I am Murdoch.
Which one is the spy?

3.4.1 Code implementation

Code:

```
formulas(assumptions).
        (knave(a2) & knight(b2) & spy(c2)) |
        (knave(a2) & spy(b2) & knight(c2)) |
        (knight(a2) & knave(b2) & spy(c2)) |
        (knight(a2) & spy(b2) & knave(c2)) |
        (spy(a2) & knight(b2) & knave(c2)) |
        (spy(a2) & knave(b2) & knight(c2)).

        spy(a2) | knave(a2).

        knight(b2).
        knight(b2) -> spy(a2).

        spy(c2) | knave(c2).
end_of_list.
```

Explanation:

- A is the spy, B is the knight, and C is the knave.

3.5 Question 4 - The Return of Murdoch

"Well," continued the King, "Murdoch was sent to prison, but soon after, he escaped and fled the land. He then came back well disguised, so no one could recognize him. Again, he was arrested in the company of a knight and a knave, and the three—call them A, B, C—made the following statements at the trial:
A: My name is Murdoch.
B: That is true.
C: I am not Murdoch.
Which one is Murdoch this time?

3.5.1 Code implementation

Code:

```
formulas(assumptions).
        (knave(a3) & knight(b3) & spy(c3)) |
        (knave(a3) & spy(b3) & knight(c3)) |
        (knight(a3) & knave(b3) & spy(c3)) |
        (knight(a3) & spy(b3) & knave(c3)) |
        (spy(a3) & knight(b3) & knave(c3)) |
        (spy(a3) & knave(b3) & knight(c3)).

        spy(a3) | knave(a3).

        knight(b3) -> spy(a3).
        knave(b3) | spy(b3) -> knave(a3).
```

```
13
14          knight(c3) | spy(c3).
15  end_of_list.
```

Explanation:

- B is Murdoch.

3.6 Question 5 - A More Interesting Case

"Well," began the King, "in this trial we again have three defendants—A, B, C. The court knew that one was a knight, one a knave, and the other the spy, but it was not known who was which. First A accused B of being the spy; then B accused C of being the spy; and then C pointed to one of the other two defendants and said, 'He is really the spy!' The judge then convicted the spy. Which one did he convict?"

3.6.1 Code implementation

Code:

```
1   formulas(assumptions).
2           (knave(a4) & knight(b4) & spy(c4)) |
3           (knave(a4) & spy(b4) & knight(c4)) |
4           (knight(a4) & knave(b4) & spy(c4)) |
5           (knight(a4) & spy(b4) & knave(c4)) |
6           (spy(a4) & knight(b4) & knave(c4)) |
7           (spy(a4) & knave(b4) & knight(c4)).
8
9           knight(a4) -> spy(b4).
10          knave(a4) -> knight(b4).
11          spy(a4) -> knight(b4) | knave(b4).
12
13          knight(b4) -> spy(c4).
14          knave(b4) -> knight(c4).
15          spy(b4) -> knight(c4) | knave(c4).
16
17          knight(c4) -> spy(a4) | spy(b4).
18          knave(c4) -> knight(a4) | knight(b4).
19          spy(c4).
20  end_of_list.
```

Explanation:

- For this problem I reduced the number of cases for the last three propositions on paper.
- C is the spy, B is the knight and A is the knave.

3.7 Question 6 - A Still More Interesting Case

First the judge asked A, 'Are you the spy?' A answered (yes or no). Then the judge asked B, 'Did A tell the truth?' B answered (again either yes or no).

"At this point, A said, 'C is not the spy.' The judge replied, 'I already knew that. And now I know who the spy is!'

3.7.1 Code implementation

Code:

```
formulas(assumptions).
        (knave(a5) & knight(b5) & spy(c5)) |
        (knave(a5) & spy(b5) & knight(c5)) |
        (knight(a5) & knave(b5) & spy(c5)) |
        (knight(a5) & spy(b5) & knave(c5)) |
        (spy(a5) & knight(b5) & knave(c5)) |
        (spy(a5) & knave(b5) & knight(c5)).

        knight(c5) | knave(c5).
        -spy(c5).
        knight(a5) | spy(a5).
        -knave(a5).

        ((knave(c5) & spy(a5) & knight(b5))).
end_of_list.
```

Explanation:

- I once again minimized the possible cases on paper.
- A is the spy, C knave, and B is the knight.

3.8 Conclusion and Observations

- One model results after running mace4 on the input file.

3.9 References

- https://biblioteca.utcluj.ro/files/carti-online-cu-coperta/290-8.pdf
- *Alice in Puzzle-land*, RAYMOND SMULLYAN