Report: The River Crossing Problem

By Enrique Perez Soler

1. **Introduction of the project**

   The purpose of this project is to make use of the skills learnt through the lectures of Logic Programming and to practise the implementation of a program in the Prolog language. After searching for an interesting topic to address in my project for this course i was referred to everytime to singular mathematical and logical problems. Or rather, paradigms of puzzles that play with the concepts of logic above all. After a deep search i finally settled for *the River Crossing puzzle,* or the *problem of the Fox, the Goose and the bag of Beans*. I've tried to use the knowledge i have learn during the course and tried to elaborate a complex problem to showcase the concepts of this subject. To make matters easier i have modified the problem and used instead Foxes, Sheeps and Beans.

2. Description of the project

The problem goes like this a farmer went to a market and purchased a fox, a sheep and a bag of beans. On his way home, he approached the bank of a river and rented a boat to cross it. Unfortunately the boat had only capacity for him and only one of his purchases: the fox, the sheep or the bag of beans.

At first it wouldn't be a problem as he could make 3 voyages (6 trips) but unfortunately nature has its rules and so if the farmer would have taken first the bag of beans by the time he would have gotten back to take another of his purchases he would have found only one left, the fox, as this one would have eaten the sheep while the farmer was gone. Same issue would have happened if he had taken the fox, the sheep would have eaten the beans. So then, how did he do it?

SOLUTION: In the first trip he'd take the sheep (as the fox doesn't eat the beans). Once he has left the sheep on the other shore, he returns to pick another of his purchases. Now we have the same problem, if he takes the fox to the other side, while he goes back to pick the beans the fox would have eaten the sheep when he came back. And if he takes the beans when he had come back to pick the fox, the sheep would have eaten the beans.
The dilemma is solved by taking the fox (or the beans) over and bringing the sheep back. Now he can take the beans (or the fox) over, and finally return to fetch the sheep.

His actions in the solution are summarised in the following steps:

1. Take the sheep over

2. Return
3. Take the fox/beans over
4. Return with the sheep
5. Take the beans/fox over
6. Return
7. Take the sheep over

As we can see there are seven crossings, four forward and three back and regardless of which is taken secondly after the sheep, the fox or the beans.

3. Difficulties and evolution of the project

From the beginning i had quite clear how i was going to handle the problem with the basic parameters (1 fox, 1 sheep, 1 bag of beans and 2 seats in the boat). But because before i started programming i had to sit, take a piece of paper and start to write down every possible scenario. I began to realise and understand the logical background behind it all. And therefore, at that moment I decided to write a program that could work with any possible amount in each parameter.

The program is divided into **4 parts:**

- **The Status Declaration:** where i have implemented the methods:
  - **init(Status):** defines the initial status based on the input parameters.
  - **end(Status):** defines the final status that is no purchases on the initial shore.
- **Safe statuses:** here i declared all the possible statuses in which the scenario would be safe:
  - Scenario in which there are no Sheeps but there're Foxes and Beans in the shore
  - Scenario in which there are no Sheeps but there are Foxes in the shore
  - Scenario in which there are no Sheeps but there are Beans in the shore
  - Scenario in which there are Sheeps but no Foxes nor Beans in the shore
  - Scenario in which the Farmer and the Sheep are in the same shore
  - Scenario in which there are no purchases left on the shore.

- **Available Movements:** the available movements i have developed a system to establish them based on the status (before and after such movement) and the values of each parameter in each shore, here i show you a schematic table summarizing how it works:

Shore:	**Shore**	**1**				**Shore**	**2**		
param:	**Fa1**	**Fox1**	**Shep1**	**Bean1**	**---**	**Fa2**	**Fox2**	**Shep2**	**Bean2**
Status (i)	Fai1	Foxi 1	Shepi1	Beani1	---	Fai2	Foxi2	Shepi2	Beani2
Status (f)	Faf1	Foxf 1	Shepf1	Beanf1	---	Faf2	Foxf2	Shepf2	Beanf2

So for instance, the first movement in the base case (farmer, 1 fox, 1 sheep and 1 bag of beans) is the one where the farmer takes the sheep first, the result would be:

Shore:	**Shore**	**1**				**Shore**	**2**		
param:	**Fa1**	**Fox1**	**Shep1**	**Bean1**	**---**	**Fa2**	**Fox2**	**Shep2**	**Bean2**
Status (i)	1	1	1	1	---	0	0	0	0
Status (f)	0	1	0	1	---	1	0	1	0

Added to the status changes, there are certain conditions that need to be contemplated as the problem itself has certain restrictions that makes the movements allowed or not in order to have a successful outcome.

- **Print Status Table:** in this part of the program i have created a method that prints every status the problem goes through to see the result more visually. For instance, here we see the result of the original problem:

```
[1]  ?-
|    solution.

******* PROBLEM OF THE RIVER CROSSING *******

Enter initial parameters:

        * Number of Foxes: 1.
        * Number of Sheeps: |: 1.
        * Number of Bean bags: |: 1.
        * Boat capacity: |: 2.

Solution:

        Shore 1                                                 Shore 2
Farmer  Fox     Sheep   Beans           Farmer  Fox     Sheep   Beans
1       1       1       1       |~ ~ ~|   0       0       0       0
                                |  ~ ~ |
0       1       0       1       |~ ~ ~|   1       0       1       0
                                |  ~ ~ |
1       1       0       1       |~ ~ ~|   0       0       1       0
                                |  ~ ~ |
0       0       0       1       |~ ~ ~|   1       1       1       0
                                |  ~ ~ |
1       0       1       1       |~ ~ ~|   0       1       0       0
                                |  ~ ~ |
0       0       1       0       |~ ~ ~|   1       1       0       1
                                |  ~ ~ |
1       0       1       0       |~ ~ ~|   0       1       0       1
                                |  ~ ~ |
0       0       0       0       |~ ~ ~|   1       1       1       1
                                |  ~ ~ |
true .
```

The reading of the table is based on the available movements' structure. Each line represents the current status of the two shores and the next one is the following status after a certain allowed movement. Each column represents the evolution of the parameters through the program's execution.

- **Depth Search:** formed by a method that makes a depth search to find the possible and optimal movement options. This method allows me to find the best path and if found a more optimal one backtracking and choosing a new path, with the final goal of reaching the final standard status that is nothing left on the initial shore and with all the purchases in the other shore.

*** The Farmer:** i could have counted out the parameter of the farmer, establishing the boat capacity parameter to actually be the "*amount of seats remaining in the boat*" without counting the one the farmer has, so i didn't have to manage a binary value going back and forth. But when i tried to take him out of the equation the results didn't add up, so then i realised that the farmer is basic and necessary in order to evaluate the conditions and therefore it is a must-have parameter.

TEST PHASE:

- **Test1:** we try adding 1 more purchase of any kind but the same boat capacity
    - Result: FAIL.

```
[1]  ?- solution.

*** PROBLEM OF THE RIVER CROSSING ***

Enter initial parameters:

        Number of Foxes: 2.
        Number of Sheeps: |: 1.
        Number of Bean bags: |: 1.
        Boat capacity: |: 2.

false.

[1]  ?- solution.

*** PROBLEM OF THE RIVER CROSSING ***

Enter initial parameters:

        Number of Foxes: 1.
        Number of Sheeps: |: 2.
        Number of Bean bags: |: 1.
        Boat capacity: |: 2.

false.

[1]  ?- solution.

*** PROBLEM OF THE RIVER CROSSING ***

Enter initial parameters:

        Number of Foxes: 1.
        Number of Sheeps: |: 1.
        Number of Bean bags: |: 2.
        Boat capacity: |: 2.

false.
```

- **Test2:** we try adding 1 more purchase of any kind and 1 more seat in the boat
    - Result: SUCCESS. There's a solution

```
[1]  ?- solution.

*** PROBLEM OF THE RIVER CROSSING ***

Enter initial parameters:

        Number of Foxes: 2.
        Number of Sheeps: |: 1.
        Number of Bean bags: |: 1.
        Boat capacity: |: 3.

Solution:

        Shore 1                                         Shore 2
Farmer  Fox     Sheep   Beans           Farmer  Fox     Sheep    Beans
1       2       1       1       |~ ~ ~|   0       0       0       0
                                |  ~ ~ |
0       2       0       0       |~ ~ ~|   1       0       1       1
                                |  ~ ~ |
1       2       1       0       |~ ~ ~|   0       0       0       1
                                |  ~ ~ |
0       0       1       0       |~ ~ ~|   1       2       0       1
                                |  ~ ~ |
1       0       1       0       |~ ~ ~|   0       2       0       1
                                |  ~ ~ |
0       0       0       0       |~ ~ ~|   1       2       1       1
                                |  ~ ~ |
true .

[1]  ?- solution.

*** PROBLEM OF THE RIVER CROSSING ***

Enter initial parameters:

        Number of Foxes: 1.
        Number of Sheeps: |: 2.
        Number of Bean bags: |: 1.
        Boat capacity: |: 3.

Solution:

        Shore 1                                         Shore 2
Farmer  Fox     Sheep   Beans           Farmer  Fox     Sheep    Beans
1       1       2       1       |~ ~ ~|   0       0       0       0
                                |  ~ ~ |
0       1       0       1       |~ ~ ~|   1       0       2       0
                                |  ~ ~ |
1       1       0       1       |~ ~ ~|   0       0       2       0
                                |  ~ ~ |
0       0       0       0       |~ ~ ~|   1       1       2       1
                                |  ~ ~ |
true .


[1]  ?- solution.

*** PROBLEM OF THE RIVER CROSSING ***

Enter initial parameters:

        Number of Foxes: 1.
        Number of Sheeps: |: 1.
        Number of Bean bags: |: 2.
        Boat capacity: |: 3.

Solution:

        Shore 1                                         Shore 2
Farmer  Fox     Sheep   Beans           Farmer  Fox     Sheep    Beans
1       1       1       2       |~ ~ ~|   0       0       0       0
                                |  ~ ~ |
0       0       0       2       |~ ~ ~|   1       1       1       0
                                |  ~ ~ |
1       0       1       2       |~ ~ ~|   0       1       0       0
                                |  ~ ~ |
0       0       1       0       |~ ~ ~|   1       1       0       2
                                |  ~ ~ |
1       0       1       0       |~ ~ ~|   0       1       0       2
                                |  ~ ~ |
0       0       0       0       |~ ~ ~|   1       1       1       2
                                |  ~ ~ |
true .
```

As we can see, there's a certain correlation between the capacity of the boat and the number of extra purchases. If there's one addition to any of the purchases (one more fox, one more sheep, one more bag of beans) then in order for the

problem to be solved there must be an additional seat in the boat. Nevertheless, due to the conditions that establish the restrictions in the statuses of the shores, an additional seat in the boat may let way to a wider range of possible amounts f purchases as seen in the following example:

```
[1]  ?- solution.

******* PROBLEM OF THE RIVER CROSSING *******

Enter initial parameters:

        * Number of Foxes: 2.
        * Number of Sheeps: |: 2.
        * Number of Bean bags: |: 1.
        * Boat capacity: |: 3.

Solution:

        Shore 1                                         Shore 2
Farmer  Fox     Sheep   Beans           Farmer  Fox     Sheep    Beans
1       2       2       1       |~ ~ ~|   0       0       0       0
                                |  ~ ~ |
0       2       0       1       |~ ~ ~|   1       0       2       0
                                |  ~ ~ |
1       2       0       1       |~ ~ ~|   0       0       2       0
                                |  ~ ~ |
0       0       0       1       |~ ~ ~|   1       2       2       0
                                |  ~ ~ |
1       0       2       1       |~ ~ ~|   0       2       0       0
                                |  ~ ~ |
0       0       2       0       |~ ~ ~|   1       2       0       1
                                |  ~ ~ |
1       0       2       0       |~ ~ ~|   0       2       0       1
                                |  ~ ~ |
0       0       0       0       |~ ~ ~|   1       2       2       1
                                |  ~ ~ |
true .
```

4. Final Conclusions

To conclude i must say the effort put in this project was great and i've learned to find out the tricks and shortcuts to the Prolog language. The fact that the problem itself was interesting and could have a visual representation had made the work a lot easier. The code might be rusty and messy or even hard to understand, at first i was confused when i wrote everything, especially the part of the available movements, i had to study the real algorithm behind the mathematical problem and it had a lot to do with graphs. Summarizing, it was a thrill ride and even though it was tough and overwhelming on me sometimes i enjoyed both the researching and the coding part.