COMP24412
Academic Session: 2018-19

Lab Exercise 2A: More Prolog

For this lab exercise you should do all your work in your COMP24412/ex2 directory.

This lab exercise will be marked along with the lab exercise 2b with both due in week 8. Lab exercise 2b will extend what you do in this lab exercise by replacing Prolog with first-order logic and a reasoning tool called Vampire.

If you did not do all of the warmup in the first lab then you should consider finishing this now.

Learning Objectives

At the end of this lab you should be able to:

1. Model more complex constraint solving problems requiring arithmetic in Prolog
2. Express the same relations in Prolog and first-order logic and give examples of when Prolog is less expressive than first-order logic
3. Use Prolog to model real world situations
4. Explain and demonstrate the relationship between Prolog, Datalog, and standard relational databases

Part 1: Solving Arithmetic Constraints

In the first lab we saw how to introduce constraints and use Prolog to solve them. In lectures we saw how we can use the `clpfd` library to solve constraints involving arithmetic. To use this library you should include the following at the start of your Prolog program:

```
:- use_module(library(clpfd)).
```

You may also find this website helpful.

Reminder: Please solve the problems using pure Prolog. Solutions containing non-logical constructs (cut, explicit disjunction, if then else, negation) will most likely lead to errors in your predicates.

In this part of the exercise you are a busy farmer with two important tasks to handle. You have chosen to tackle these using constraints in Prolog.

Paying Taxes

The government has just introduced a new tax on cows and chickens. You check your records and realise that you've not been recording how many of each animal you have, only how many heads and feet all of your animals have in total.

Write a Prolog program to work out how many cows and chickens you have given that the sum of heads is 65, the sum of feet is 226. You should assume that all animals have the normal number of heads and feet.

Use arithmetic constraints to implement the core relation `cows_chicken_eqs(Cows,Chicken)` that only expresses the two equations about the numbers of heads and feet. Write a second predicate `cows_chicken(Cows, Chicken)` that assigns appropriate domains for Cows and Chicken and labels the constraints obtained from the core relation to obtain a solution to the puzzle.

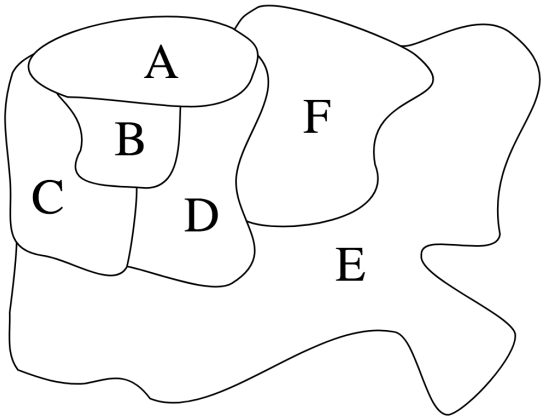
Please save your program in a file `farmer_tax.pl`.

Hint: The query `cows_chicken_eqs(Cows,Chicken)` of the core relation must always succeed and show the (simplified) constraints but it may not unify the variables with concrete solutions. The query `cows_chicken(Cows,Chicken)` to the final predicate must unify the variables with concrete solutions but it may not show any constraints.

Planting Crops

Next you need to decide which crops to plant in each of your fields. You have three crops: *cabbages*, *potatoes*, and *carrots*. However, the government department for 'picturesque fields from above' does not allow you to plant the same crop in two adjacent fields.

Your fields are laid out as shown in the below map:



You are doubtful if this is even possible but to prove it you formulate the situation as a constraint satisfaction problem where each crop is represented by a number. Use a list of six elements representing the fields A to F. First, implement a core predicate `crops_eqs(Fields)` that describes which neighbouring fields can not have the same crops.

Then implement a predicate `crops3(Fields)` that assigns domains to the variables in `Fields` and labels the constraints obtained from the core predicate. Convince yourself that the requirements are impossible to fulfill by checking that the query `crops3(Fields)` does not find an answer.

After reporting your findings, the department just reiterates the requirements above. However, you think you could get away with planting lavender as an additional crop. Similar to `crops3(Fields)`, implement a predicate `crops4(Fields)` that uses a four element domain instead and convince yourself that a query to the predicate now yields a solution.

Please save your program in a file `farmer_crops.pl`.

Hint: To find a satisfying assignment of crops to fields you should number the crops and capture the adjacency of fields as disequality constraints. Keep in mind that constraints have their own predicates for equality (`#=`) and disequality (`#\=`).

Extension (not marked): You notice that the fields `[1, 2, 3, 4, 1, 2]` and `[2, 1, 3, 4, 2, 1]` describe the same solution where the names of crop one and two are just switched. Implement a core predicate `crops_asym(Fields)` that adds asymmetry constraints on the field.

Part 2: Prolog and FOL

For each of the following statements write (i) a Prolog relation , and (ii) a first-order formula, or if you cannot say why. In both cases you can assume relevant extensional facts e.g. $\mathsf{man}(x)$ but you should assume the same facts in both definitions.

1. If something has an engine and four wheels it is a car
2. A good meal is one that contains three different kinds of food
3. Every pet is either a dog or a cat
4. An object A is part of another object B either directly or due to the fact that A is part of an object C, which is itself part of B.
5. Hate is always mutual
6. For every action, there is an equal and opposite reaction
7. There are only ever exactly two Sith, a master and an apprentice
8. Everybody was Kung Fu fighting, those cats were fast as lightning
9. Nobody expects the Spanish Inquisition
10. It is a truth universally acknowledged, that a single man in possession of a good fortune, must be in want of a wife.

Like in English, there are many ways to express the same sentiment using language. Both Prolog and first-order logic are unambiguous in what they mean but this does not remove the ambiguity in the original statement. Here you will have to make decisions about how you're going to interpret the English and how you want to model things. You should justify your decisions. There will be an opportunity during the feedback process to adjust your definitions.

You should write these definitions and your explanations in a file named `part2.txt`.

Part 3: Modelling Something You Know

In this part you will pick your own domain to model. For inspiration, here are some examples of things you might choose to model:

- The Premier League
- The Human Body
- The School of Computer Science
- The Music Industry

You **do not** need to choose one of these and we strongly encourage you not to - choose something you know well. The domain can be big (like our examples) or small but it needs to be complex enough for you to complete the following tasks.

Step 1: The initial knowledge base

Create a knowledge base as a Prolog program modelling your chosen domain. Your knowledge base should contain at least 5 extensional relations (defined by facts) and at least 10 intensional relations (defined by rules). At least two of your rules must be recursive. If you cannot manage this with your chosen domain then you should pick a different domain.

Save your knowledge base as a Prolog program called `model.pl`.

Step 2: Relation to Relational

Investigate the relationship between Prolog and languages used for relational databases e.g. SQL or relational algebra. Describe how your extensional relations could be represented as tables and how your intensional relations could be represented as queries, or describe why they cannot if not.

Write up what you have found in `modelling_relational.txt`

Step 3: Now to First-Order Logic

Update: Please read the relevant part in Lab 2B before completing this step as further hints and details are provided there.

The final step for this first half of the exercise is to translate the above knowledge base and queries to first-order logic. For the most part this should be straightforward. But, are there any assumptions that you were making that you cannot make anymore?

Now write at least two queries that cannot easily be asked in Prolog. Typical examples of such queries are looking at the properties of relations themselves e.g. is one relation a sub-relation of another, is a relation symmetric etc.

In the next half of the exercise you will be using Vampire to reason over this FOL knowledge base. Remember that both halves will be marked together so if you struggle with this step then the next steps may help you debug things.

You should save your working in a file `modelling_fol.txt` to be used as a starting point for the next step in the second half.