TDT4165 PROGRAMMING LANGUAGES

Assignment 5

Relational and Constraint Programming

Autumn 2024

Preliminaries

This exercise is about relational and constraint programming, and will be performed in Prolog, as it provides better support for constraint programming.

Relevant reading: Chapters 9 and Chapter 12 in the CTMCP book, and the "Learn Prolog Now!" material, available at http://www.let.rug.nl/bos/lpn/.

Delivered code must be runnable in the Prolog environment accessible on https://clp-tdt4165.idi.ntnu.no/.

Please deliver the code as a single code file (typically, the extension `.pl` is used for Prolog code). The delivery should also include a `.pdf` file containing a section for each task. For each task, the PDF should describe the implementation, or include a screenshot of the code, as well as answer any theoretical questions. You can use the template found on BlackBoard, under "Coursework" / "Latex template for PDFs", to generate your PDF file.

Evaluation

This assignment is graded as Approved/Not approved.

The requirements to get this exercise approved are as follows:

- Task 1 implemented
- Task 2.1 attempted
- Task 2.2 attempted

Task 1: Constraint programming

In the Prolog environment found at https://clp-tdt4165.idi.ntnu.no/, click the CLPFD tab, and then "Notebook".

Read through the `Constraint Programming` chapter, and do the tasks under the `Exercise` section. You will find three Prolog code blocks: one empty, and two pre-filled with queries. The task is to write the appropriate predicate in the first block, so that queries in the other two succeed.

Task 2: Relational programming

Introduction

In this task you're going to create a relational program to solve tasks involving pairs of cabins. For this task, use the "Empty" tab in the aforementioned Prolog environment. From there, add the initial program by selecting the "+" button, and choosing "Program". In this cell, add the following code.

```
:- use_module(library(clpfd)).
distance(c1, c2, 10, 1). distance(c1, c3, 0, 0). distance(c1, c4, 7, 1).
distance(c1, c5, 5, 1). distance(c2, c3, 4, 1). distance(c2, c4, 12, 1).
distance(c2, c5, 20, 1). distance(c3, c4, 0, 0). distance(c3, c5, 0, 0).
distance(c4, c5, 0, 0). distance(c2, c1, 10, 1). distance(c3, c1, 0, 0).
distance(c4, c1, 7, 1). distance(c5, c1, 5, 1). distance(c3, c2, 4, 1).
distance(c4, c2, 12, 1). distance(c5, c2, 20, 1). distance(c4, c3, 0, 0).
distance(c5, c3, 0, 0). distance(c5, c4, 0, 0).
```

This code defines the *facts* of our problem, that is, the list of cabins that will be used to test your implementation and the distances between them.

More precisely, you will be using the `distance(Cabin1, Cabin2, Distance, Connected)` predicate to denote a relation between two cabins, describing the distance between them, and whether there exist a direct path between them or not.

Task 2.1: Create a planner

Implement a planner `plan(Cabin1, Cabin2, Path, TotalDistance)`, where `Cabin1` and `Cabin2` are two cabins, `Path` is a path encoded as a list of the cabins that are visited, and `TotalDistance` is a number.

The predicate holds if it is possible to reach `Cabin2` from `Cabin1`, going through `Path` and covering a total distance of `TotalDistance`. You should also make sure that the path does not contain closed loops. You can also think of the predicate as a function that, given two cabins, calculates the `Path` and the `TotalDistance`.

Running the query "`plan(c1, c2, Path, TotalDistance).`", should result in these three instances:

- `Path = [c1, c2], TotalDistance = 10`
- `Path = [c1, c4, c2], TotalDistance = 19`
- `Path = [c1, c5, c2], TotalDistance = 25`

Note that in the SWISH notebook you can create a new permanent block holding query, by pressing the "+" at the top, and then selecting "Query".

- *Hint 1*: You can define multiple rules for the same predicate, which are the Prolog way to realize conditional statements.
- *Hint 2*: Use the `not(X)` predicate to define rules that cannot hold, e.g., `not(Cabin1 = Cabin2)` as part of the rule would mean that the path cannot be from a cabin to itself.
- *Hint 3*: Consider implementing additional help-predicates when implementing the `plan` predicate, and then wrapping the `plan` predicate around them.
- *Hint 4*: Consider using the `append([Head1|Tail1],[Head2|Tail2], List)` predicate to append elements to a list. Lists in Prolog work very similar to what you have already experienced in Oz. See also Chapters 4, 5, and 6 in "Learn Prolog Now!" for more details.

Task 2.2: Create the planner for the shortest path

In the previous task you have defined a predicate that defines *a* path between two cabins. Here, you are going to implement the predicate `bestplan(Cabin1, Cabin2, Path, Distance)`, which "finds" the minimal path between two cabins.

More precisely, `bestplan(Cabin1, Cabin2, Path, Distance)` holds if `Path` is the shortest path between cabins `Cabin1` and `Cabin2`, and its total length is `Distance`.

- *Hint*: To compare results from the same predicate, you can use more instances of it, bind the arguments to different variables, and then make comparisons between those variables. For example, `employee(Company, Name1, Age1), employee(Company, Name2, Age2), Age2>Age1.` defines that the rule holds if the second employee is older than the first.