Imperial College London – Department of Computing

MSc in Advanced Computing
MSc in Computing Science (Specialism)

531: Prolog
'Prison' — Assessed

Issued: 7 November 2018 **Due:** 15 November 2018

Introduction

The high security prison, San Parkmoor Scrubs, has 100 cells, one prisoner in each cell. There are 100 warders.

In high spirits following a drinking session at the Christmas party, the warders play a game. They line up in the gymnasium. The first warder runs through the prison unlocking every cell. The second warder follows him, turning the key in every second cell door. Thus all even numbered cells, which were unlocked after the first warder's run, are locked after the second warder has passed. The third warder follows the second, turning the key in every third cell door. The fourth warder follows the third, and so on, the Nth warder turning the key in every Nth cell door. The last warder therefore turns the key in the last cell door only. Some of the prisoners escape, but the warders are too excited by their game to notice exactly how many. How many prisoners escaped?

The next day, the Department of Justice hears of the escapade and sees it as a way of reducing overcrowding in prisons. Before issuing the appropriate regulations, however, its officials wish to estimate how many prisoners will escape from their prisons.

Charlotte Ann Software is commissioned to write a program that will calculate how many prisoners escape from a prison with M cells and N warders. (Most prisons do not have the same number of warders as cells.) You are the technical director (and only programmer) of Charlotte Ann Software.

You start to point out to the Department of Justice that the number of escaped prisoners given any M and N can be calculated by a simple formula. However, when you are told what the fee is, you decide to keep quiet and write the program.

Two Alternative Programs

You will write **two versions** of the program in this exercise. The first ('Problem 1') is a kind of simple simulation. Given a list representing the state of all M cell doors in the prison, it will simulate the running of N warders through the prison. The second version ('Problem 2') uses a different approach. It is a very simplified version of a common technique in temporal reasoning in AI, though you do not need to know that to solve the problem. Problem 2 also has a number of extensions to make the problem (slightly) more interesting.

You are provided with a skeleton solution in the file `prison.pl`. You should complete both problems by adding to this file.

This is essentially an exercise in writing list processing programs. For that reason do *not* use the Sicstus `library(lists)`. You do not need it for this exercise.

Obtaining the Exercise Files

You will need to clone the skeleton respository to your DoC home directory in order to work on it. Later you will need to *push* your changes back to the server.

- You can get your skeleton repository by issuing the following command (all on one line, replacing the occurrence of *`login`* with your DoC login):

  `git clone https://gitlab.doc.ic.ac.uk/lab1819_autumn/531_prison_`*`login`*`.git`

  Or, if you have set up ssh key access you can use:

  `git clone git@gitlab.doc.ic.ac.uk:lab1819_autumn/531_prison_`*`login`*`.git`

- This will create a new directory called `531_prison_`*`login`*. Inside you will find the following files / directories:
  - `prison.pl` — this is the source file you should edit to implement the procedures required for this exercise.
  - `prisonDb.pl` — contains a Prolog database to be used in Problem 2.
  - `.git` and `.gitignore`

Problem 1

You will complete the Prolog program for

`prison_game(+C, +W, -Escaped)`

which, given `C` cells and `W` warders (both positive integers), will return `Escaped`, the list of cell numbers of prisoners whose cell doors are unlocked after all warders have made their runs. The definition of `prison_game/3` itself is given to you in the skeleton file. (Note that `prison_game` requires there to be at least one cell and at least one warder, and does not simulate the run of the first warder since he will simply unlock all the doors.)

What To Do

Task 1.1 In `prison.pl`, write a Prolog program

```
make_list(+N, +Item, -List)
```

which given a (non-negative) integer `N` and item `Item` will construct a list `List` of `N` elements each of which is `Item`. For example, `make_list(5, a, List)` should return `List = [a,a,a,a,a]`.

Task 1.2 In `prison.pl`, write a program

```
extract_indices(+List, +Item, -Indices)
```

which given a list `List` and item `Item` will compute the list `Indices`, the list of integers N such that `Item` is the Nth item of the list `List`. For example, your program should return the following answers for `Indices` in the queries below:

```
?- extract_indices([m,a,n,d,e,l,a], a, Indices).  Indices = [2,7]
?- extract_indices([m,a,n,d,e,l,a], n, Indices).  Indices = [3]
?- extract_indices([m,a,n,d,e,l,a], k, Indices).  Indices = []
```

You can assume that `List` and `Item` are ground when `extract_indices` is called.

Task 1.3 In `prison.pl`, write a program

```
run_warders(+N, +W, +Initial, -Final)
```

which, given next warder `N` and total warders `W` (both positive integers), and current door states `Initial` (a list of the constants `locked` and `unlocked`) will return `Final`, the list of door states after all warders have completed their runs. For example, your program should return the following answers for `F` in the queries below:

```
?- run_warders(2,2,[unlocked,unlocked],F).      F = [unlocked,locked]
?- run_warders(2,3,[locked,locked,locked],F).  F = [locked,unlocked,unlocked]
```

Problem 2

In this exercise you will solve the problem in a *different* way. (You should not use any of the parts of your answer to Problem 1 in Problem 2 — it does not use the model of the list of cell states.)

This part of the exercise uses a database of facts about a (fictional, unnamed) prison. Such a database is provided in the file `prisonDb.pl`. The database details which prisoners are in which cell, meaning you can compute the list of names of all escaped prisoners.

Extended Scenario

In Problem 2:

- There may be *more than one prisoner in a cell*, and there may be some *empty* cells.

- Some prisoners are psychopaths. No prison warder will *ever* unlock the cell door of a psychopath.
- A prisoner who has one year or less to serve *will not run away* even if his/her cell door is left unlocked.

The Database

The file `prisonDb.pl` defines the following predicates:

`cells/1`, `warders/1`, `prisoner/6`, `psychopath/2`

- `cells/1` and `warders/1` give the number of cells and number of warders in the prison, respectively.
- `prisoner(Surname, FirstName, Cell, Crime, Sentence, ToServe)` represents data about the prisoners. A prisoner is uniquely identified by his or her first name and surname. The other arguments represent the following information for each prisoner.

`Cell`	the prisoner's cell number (a positive integer)
`Crime`	the crime for which the prisoner was convicted
`Sentence`	the number of years (positive integer) for which the prisoner was originally convicted
`ToServe`	the number of years (positive integer) left to serve of the prisoner's orginal sentence

- `psychopath(Surname,FirstName)` holds when the prisoner with that name is a psychopath. (Psychopaths are not necessarily kept in cells by themselves.)

What To Do

Task 2.1 Write a program defining `cell_status(+Cell, +N, ?Status)`. Given `Cell` (a positive integer), `N` (a non-negative integer) and `Status` (one of `locked` or `unlocked`), a call `cell_status(Cell, N, Status)` should succeed if the status of cell `Cell` is `Status` after `N` warders have made their runs. If `Status` is a variable, then the correct value should be returned.

Don't forget about the possibility of psychopaths in some cells.

Task 2.2 Write Prolog programs for

`escaped(?Surname, ?FirstName)` and `escapers(-List)`

`escaped(Surname, FirstName)` holds when the prisoner with that name escapes (i.e., occupies a cell which is unlocked after the last warder has made his run, but bearing in mind that prisoners with a year or less left to serve will not escape). `escapers(List)` is the list of escaped prisoners. `List` should be a list of terms of the form `(Surname, FirstName)`, and sorted in ascending alphabetical order according to `Surname`. Since prisoners are uniquely identified by `(Surname, FirstName)` you do not need to worry about removing duplicates.

Testing

You should test your program on a range of suitable examples before submission. You can automate your testing using the `plunit` Sicstus library. See the Sicstus documentation for details.

Your submitted work will be automatically tested. You must ensure that:

- Your Prolog program is written in `prison.pl`.
- Your program *COMPILES WITHOUT ERRORS* on the **Linux Sicstus** system installed on the lab machines.
- You have *not* included any print or write statements in your submitted code. Use them by all means for tracing/debugging but do not include them in your submitted program.
- You do *not* load (consult, compile etc.) `prisonDb.pl` from `prison.pl`, nor define any of the database predicates in `prison.pl`. Your programs will be evaluated on a different database.
- You do *not* use the Sicstus `library(lists)`.

You can verify that your code runs and produces sensible output by requesting an autotest online via `https://teaching.doc.ic.ac.uk/labts`. This is NOT the full set of tests that will be used in assessing your work, so you cannot assume that your program is correct if all the tests pass.

Submission

Submit By 15 November 2018

Submission is a two stage process.

1. **Push To Gitlab.** Use `git add`, `git commit` and `git push` to update the Gitlab server with the changes you have made to the skeleton repository. Use `git status` to confirm that you have no local changes you have not pushed, and then inspect the files on Gitlab:

   `https://gitlab.doc.ic.ac.uk`

2. **Submit directly to CATE.** Go to LabTS (`https://teaching.doc.ic.ac.uk/labts`), find your list of commits for this exercise and click the **Submit to CATe** button for the commit you want to submit.

Assessment

This exercise is worth 15% of the marks allocated for coursework on this module.

Your solutions will be marked on correctness, code design and readability. You should test your program on a range of suitable examples before submission. Do not overcomplicate the solution!

Grades

`F-E:` The submission shows a clear lack of understanding of how to write Prolog code. The behaviour of the majority of the predicates is likely to be incorrect.

`D-C:` The submission shows that the student can write Prolog, but the behaviour of several predicates is incorrect. Solutions may have serious efficiency problems. Solutions may be grossly overcomplicated, or otherwise poorly designed. Code style may be poor.

`B-A:` The behaviour of most or all of the predicates is correct. Solutions may have efficiency problems, and/or some design flaws. Code style is good.

`A+:` The behaviour of all of the predicates is correct. Some minor efficiency problems may exist in some predicates, but otherwise code is well designed, simple and elegant. Code style is good.

`A*:` There are no obvious deficiencies in the solution, including efficiency of the program, program design and the coding style.

Return of Work and Feedback

This exercise will be marked and returned by **30th November 2018**. Feedback on your solution will be given on the returned copy, and feedback will be given to the class in the support lectures.