COURSEWORK ASSIGNMENT UNIVERSITY OF EAST ANGLIA
School of Computing Sciences

Unit : CMP-6040A
Assignment Title : Prolog coursework: analysis of a two-player game

Date Set	: Wednesday week 5 (25/10/17)	
Date & Time of Submission	: Wednesday week 10 (29/11/17)	
Return Date	: Week 13	
Assignment Value	: 40%	
Set By	: Dr P. Chardaire	**Signed:**
Checked By	: Dr W. Wang	**Signed:**

Aim:

To write a Prolog program to implement an algorithm for finding the best move in a two-player game, and play the game interactively as well as possible.

Learning outcomes:

This coursework should:

- Develop students' understanding of logic programming.
- Develop students' understanding of search problems.
- Develop students' ability in programming.

Description of assignment:

A two-person game is played with a set of identical stones arranged into a number of heaps. There may be any number of stones and any number of heaps. A move in this game consists of either removing any number of stones from one heap, or removing an equal number of stones from each of two heaps. The loser of this game is the player who picks up the last stone.

For example with three heaps, of sizes 3, 2, and 1, there are ten possible moves, leading to these states (where we use the obvious representation of game states in Prolog as a list of the sizes of the heaps):

Taking from the first heap only: [2,2,1], [1,2,1], [2,1]
Taking from the second heap only: [3,1,1], [3,1]
Taking from the third heap only: [3,2]
Taking from the first and second heaps: [2,1,1], [1,1]
Taking from the first and third heaps: [2,2]
Taking from the second and third heaps: [3,1]

[3,1] occurs twice because there are two different ways of reaching it in one move.

Write a Prolog program which includes predicates to perform the following tasks:

1. A non deterministic predicate `move(S1,S2)` that upon backtracking returns all states `S2` reachable from `S1` in one move. 20 marks

2. A predicate `win(S)` that succeed if `S` is a winning position for the player whose turn it is to play and fails otherwise. 20 marks
   Besides giving the correct answers, your code for this should avoid evaluating the same position or equivalent positions more than once. For example, there are only 930 positions that can be reached from `[30,30]`, but more than one hundred octillion games that could be played starting from there.

3. A predicate `analyse(S)` that determines, for any game state `S`, whether it is a win or a loss for the player whose turn it is, given best play on both sides. When it is a win for that player, it should print out all of the winning moves that can be made in that state, otherwise it should announce that there are no winning moves. For example, the query `analyse([3,2])` should report two winning moves: to `[1]` and to `[2,2]`, while `analyse([3,5])` should report that there are no winning moves. 20 marks

4. A predicate `analyseall(N)`, where `N` is an integer, which will perform the above analysis for every game state with one or two heaps of every possible size up to `N`. For example, `analyseall(3)` should report something like this (the precise layout is not important): 20 marks

```
[1]:    no winning moves.
[2]:    [[1]]
[3]:    [[1]]
[1,1]:  [[1]]
[1,2]:  [[1]]
[1,3]:  [[1]]
[2,2]:  no winning moves.
[2,3]:  [[1],[2,2]]
[3,3]:  [[2,2]]
```

   (Since e.g. `[2,3]` is effectively the same as `[3,2]`, there is no need to print results for both states separately.)

5. A predicate `play(S)`, which given any game state `S`, will play the game interactively, as well as possible. It should do all of the following: 20 marks

   (a) It should detect when the game is over and announce who has won.

   (b) If the game is not over, it should choose a winning move if possible, and otherwise select any legal move.
   Then it should ask the user for her move, and continue to play from the resulting state.

   (c) It should detect when the user tries to make an illegal move, and ask her to try again.

Assessment criteria:

Your answers will be assessed not only on the correctness of your code, but also for clarity of programming, documentation, presentation, and efficiency.

Required:

Your answer should consist of a single Prolog file which can be loaded into SWI-Prolog and tested. The code should include comments to explain what each predicate does and how it is implemented.

Handing in procedure:

Your work should be submitted electronically through the Blackboard assignment system.

Deadlines: Coursework should be submitted before 23:59 on the deadline day.

If coursework is handed in after the deadline day or an agreed extension:

Work submitted	**Marks deducted**
On the day following the due date	10 marks
On either the 2nd or 3rd day after the due date	20 marks
On the 4th day after the due date and before the 20th day after the due date	All the marks the work merits if submitted on time (ie no marks awarded)
After 20 working days	Work will not be marked and a mark of zero will be entered

All extension requests will be managed through the LTS Hub. A request for an extension to a deadline for the submission of work for assessment should be submitted by the student to the appropriate Learning and Teaching Service Hub, prior to the deadline, on a University Extension Request Form accompanied by appropriate evidence.

Plagiarism:

Plagiarism is the copying or close paraphrasing of published or unpublished work, including the work of another student; without due acknowledgement. Plagiarism is regarded a serious offence by the University, and all cases will be investigated. Possible consequences of plagiarism include deduction of marks and disciplinary action, as detailed by UEA's Policy on Plagiarism and Collusion.