Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik III
Prof. Dr. A. B. Cremers

Advanced Logic Programming
Summer Semester 2011
Dr. Günter Kniesel

Assignment 6

Due: Sunday, 22.05.2011, 23:59:59 via SVN

Submit your solution for this assignment into your group's SVN as an
Eclipse Project named "assignment06".
Programming exercises should be submitted as **Prolog files** in this project.
Theoretical exercises should be submitted as **PDF files** in this project.

As an assignment you will be implementing the Game of Life in Prolog. The Game of Life was devised by John Conway in 1970. On a two-dimensional grid of square cells, every cell can either be "alive" or "dead". Also every cell interacts with its eight neighbours, which are the directly adjacent cells (horizontaly, vertically, or diagonally).

For further background information on the Game of Life see for example:

http://en.wikipedia.org/wiki/Conway's_Game_of_Life

There is a template file of Prolog code available in the "all" folder in the SVN:

https://svn.iai.uni-bonn.de/repos/IAI_Software/se/alp2011/group/all/game_of_life/game_of_life_template.pl

This template also includes a starting generation and textual console output. The code of your tasks should be inserted at the appropriate positions.

Here is a short excerpt from the template file. The `game_of_life/0` predicate simulates one step in the Game of Life. The `next_candidates/0` predicate creates `candidate/2` facts for all coordinates of cells that may be alive in the next generation:

```
game_of_life:-
    alive(_,_),
    next_candidates,
    next_state,
    print_alive.

next_candidates:-
    alive(X,Y),
    assert_once(candidate(X,Y)),
    neighbour(X,Y,NX,NY),
      assert_once(candidate(NX,NY)),
    fail.
next_candidates.
```

Task 1. *The neighbourhood of a cell* (2Points)

Every cell interacts with its eight neighbours, which are the directly adjacent cells (horizontaly, vertically, or diagonally). Implement a predicate `neighbour(+X,+Y,-NX,-NY)` which succeeds if the cell at position (NX,NY) is a neighbour of the one at position (X,Y). For instance

```
?- neighbour(1,2,NX,NY).
```

should produce the coordinates of all 8 cells neighbouring (1,2):

```
NX = 0, NY = 1;
NX = 1, NY = 1;
NX = 2, NY = 1;
     ...
NX = 2, NY = 3;
```

Task 2. *Assert only something that isn't already there* (2 Points)

In the lecture you have learned about the `assert/1` predicate for asserting facts. The predicate `next_candidates/0` in the above excerpt makes use of a predicate `assert_once/1`.
This predicate tests whether the fact already exists; if this is not the case, the fact is asserted. `assert_once/1` should always succeed. Implement the `assert_once/1` predicate.

Tip: Lookup the documentation of the `call/1` predicate (?- help.).

Task 3. *Counting number of alive neighbours* (6 Points)

Implement a predicate `nr_of_neighbours(+X,+Y,-Nr)` that succeeds if Nr is the number of alive cells in the neighbourhood of the cell at coordinate (X,Y).

Tip: Think about the generate and test principle. You can use the predicate alive(X,Y) to check whether the cell at position (X,Y) is alive.

Task 4. *Update rules for a cell* (2 Points)

Game of Life has three rules:

1. Loneliness: Any live cell with fewer than two live neighbours dies.
2. Overpopulation: Any live cell with more than three live neighbours dies.
3. Birth: Any dead cell with exactly three live neighbours becomes a live cell.

During each step the rules are applied for all cells at the same time. The implementation checks which cells are born or surive, therefore as a simplification you should implement two predicates:

a) Implement a predicate `birth(+X,+Y,+Nr)` which succeeds if the Birth rule applies and a cell will be born in the next step.

b) Implement a predicate `survival(+X,+Y,+Nr)` which succeeds if the cell stays alive in the next generation. For a surviving cell neither the Overpopulation nor the Loneliness rule applies.

Task 5. *Next state* (3 Points)

Given the following template (this can also be found in the complete Prolog template file):

```
next_state:-                          % Compute next_alive/2
    candidate(X,Y),
    retract(candidate(X,Y)),

            << Insert Code here >>

    fail.
next_state:-                          % Turn next_alive/2 into alive/2
    retractall(alive(_,_)),
    next_alive(X,Y),
      assert(alive(X,Y)),
    fail.
next_state:-
    retractall(next_alive(_,_)).      % Delete temporary next_alive/2
```

Implement the missing code lines so that:

1. For each candidate. the number of neighbours is calculated.
2. It is then determined whether the candidate is born or survives.
3. If yes, the coordinates of the candidate are asserted as a `next_alive/2` fact.