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

Advanced Logic Programming
Summer Semester 2010
Dr. Günter Kniesel

Assignment 7

Due: Sunday, 13.06.2010, 23:59:59 via SVN

For help, contact alp-staff@lists.iai.uni-bonn.de (staff only) or alp-course@lists.iai.uni-bonn.de (staff and participants).

Please start working on the exercises early enough so that you can contact us in time in case of problems. Don't expect us to be available during weekend!

As an assignment you will be implementing Conway's Game of Life in Prolog. The Game of Life was devised by John Conway in 1970. It is a two dimensional cellular automaton with two states and a Moore neighborhood on an orthogonal grid of square cells. Every cell can either be in the alive or dead state. Also every cell interacts with its eight neighbours, which are the cells directly horizontal, vertically, or diagonally adjacent.

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 (see Assignment0):

https://svn.iai.uni-bonn.de/repos/IAI_Software/se/alp2010/group/all/game_of_life/game_of_life_template.pl

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

For the following tasks 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 cells directly horizontally, vertically, or diagonally adjacent. Implement a predicate `neighbour(+X,+Y,-NX,-NY)` which succeeds if (NX,NY) is a neighbour of (X,Y). For instance

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

should produce all possible neighbourhood coordinates via backtracking, eight in total:

```
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 `call/1` predicate.

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 also use the fact alive/2 to see which cell 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:-
    candidate(X,Y),
    retract(candidate(X,Y)),

            << Insert Code here >>

    fail.
next_state:-
    retractall(alive(_,_)),
    next_alive(X,Y),
      assert(alive(X,Y)),
    fail.
next_state:-
    retractall(next_alive(_,_)).
```

Implement the missing code lines:

1. During each step for a candidate the number of neighbours is calculated
2. The candidate is checked if it is born or if it survives.
3. This coordinate if the candidate is then asserted as a `next_alive/2` fact.