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

Advanced Logic Programming
Summer Semester 2010
Dr. Günter Kniesel

Assignment 11

Due: Sunday, 11.07.2010, 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!

Task 1. *Update Semantics* (5 Points)

Consider the following program:

```
paint_stairway(Stairway) :-
    stair(Stairway, Step,Color),
    replacement_color_for(Color,NewColor),
    paint_step(Stairway, Step, NewColor).

paint_step(Stairway, Step, NewColor) :-
    retract(stair(Stairway, Step,_)),
    assert(stair(Stairway, Step, NewColor) ).


:- dynamic step/2.
stair(s1, 1, grey).
stair(s1, 2, grey).
stair(s1, 3, grey).
stair(s1, 4, grey).

replacement_color_for(grey,green).
replacement_color_for(green,blue).
replacement_color_for(blue,yellow).
```

a) Describe the final state of the `stair/3` predicate after running the query

```
?- paint_stairway(s1), fail.
```

on the program using

1. immediate update semantics.
2. deferred update semantics.
3. really deferred update semantics.

b) Consider the following change to replacement_color_for/3. Assume it is called correctly in paint_stairway/1.

```
/**
 * Stairs beyond nr 100 get a special treatment.
 */
replacement_color_for(Stairway, Stair ,New) :-
    stair(Stairway, Stair ,OldColor),
     ( Step <100
       -> replacement_color(OldColor,New)
        ; New=blue
     ).

replacement_color(grey,green).
replacement_color(green,blue).
replacement_color(blue,yellow).
```

Describe again the final state of above query for each semantic.

Task 2. *Modules I* (3 Points)

Define in two different modules two different graphs that contain cycles. Use `strongly_connected/1` to detect them. You may do this via the console or by writing a predicate.

You can find `strongly_connected/1` in the prolog file on the website. It contains a description what you have to do, to use it. You may use but not change this file.

Give also a copy of your console output.

Task 3. *Modules II* (3 Points)

Assume, that two graphs have been dynamically generated as `graph_node/1` and `graph_edge/2` facts. Each graph would be stored in a different module.

Write a predicate, that takes two module names as parameters and compares their contents. The result of the comparision shall be stored in a third module, that is defined as third argument:

```
compare_graphs( + Module1, + Module2, +OutputModule)
```

The facts that should be generated are:

```
node_only_in_first/1
node_only_in_second/1
edge_only_in_first/1
edge_only_in_second/1
```