Declarative Programming, (Co)Induction and Monads
Module 2
Prolog lab 1

Davide Ancona, Giovanni Lagorio, Eugenio Moggi and Elena Zucca
University of Genova

PhD Course, DISI, June 29, 2012

Easy exercises Consider natural numbers defined by the functors $z/0$ and $s/1$, and list defined by the standard Prolog functors.
Define the Horn clauses for the following predicates.

1. $is\_nat/1$ s.t. $is\_nat(t)$ holds iff t is a natural number defined either inductively or coinductively. The same definition should work for both interpretations. Verify that the query $I = s(I), is\_nat(I)$ does not terminate, whereas the query $cosld((I = s(I), is\_nat(I)))$ (**beware** of the initial double parentheses!) succeeds.

2. $geq/2$ s.t. $geq(t1, t2)$ holds iff $t1$ is a natural number greater or equal than the natural number t_2. The predicate must not hold if t_1 or t_2 is not a natural number. The same definition should work for the inductive and coinductive interpretation. Verify that the query $I = s(I), geq(I, s(z))$ does not terminate, whereas the query $cosld((I = s(I), geq(I, s(z))))$ succeeds.

3. Same as point 2, but with the predicate $leq/2$ (less or equal than).

4. $gth/2$ s.t. $gth(t1, t2)$ holds iff $t1$ is a natural number greater than the natural number t_2 (only in the inductive case). The predicate must not hold if t_1 or t_2 is not a natural number.

   Change the definition to accommodate the coinductive case.

5. Same as point 4, but with predicate $lth/2$ (less than).

6. $eq/2$ s.t. $eq(t1, t2)$ holds iff $t1$ is a natural number equal to the natural number t_2. Try to define the predicate directly, without using other predicates. The predicate must not hold if t_1 or t_2 is not a natural number. The same definition should work for the inductive and coinductive interpretation.

7. $odd/1$ s.t. $odd(t)$ holds iff t is an odd natural number (only for the inductive case).

   Change the definition to accommodate the coinductive case.

8. $even/1$ s.t. $even(t)$ holds iff t is an even natural number (only for the inductive case).

   Change the definition to accommodate the coinductive case.

9. $parent/2$ s.t. $parent(t_1, t_2)$ holds iff t_1 is the parent node of t_2; assume that there is a predefined predicate $child/2$ s.t. $child(t_1, t_2)$ holds iff t_1 is a child node of t_2.

   Compare the inductive and coinductive interpretations for the same definition.

10. $eq\_list/2$ s.t. $eq\_list(t_1, t_2)$ holds iff t_1 and t_2 are two identical lists. The same definition should work for the inductive and coinductive interpretation.

11. $all\_even/1$ s.t. $all\_even(t_1)$ holds iff t_1 is a list containing just even natural numbers.

    The same definition should work for the inductive and coinductive interpretation. For instance, the query $cosld((L = [z, s(z)|L], all\_even(L)$ must succeed. Verify that in Haskell you cannot easily define such a predicate by using the built-in all function if you want it to work correctly on infinite regular lists.

12. $is\_in/2$ s.t. $is\_in(t_1, t_2)$ holds iff t_1 is an element of the list t_2.

    Verify that such a definition works properly only for the inductive case.

Less easy exercises

1. $descendant/2$ which is the transitive closure of $child/2$; assume that there is a predefined predicate $child/2$ s.t. $child(t_1, t_2)$ holds iff t_1 is a child node of t_2.

   Compare the inductive and coinductive interpretations for the same definition.

2. $ancestor/2$ which is the transitive closure of $parent/2$; assume that there is a predefined predicate $parent/2$ s.t. $parent(t_1, t_2)$ holds iff t_1 is the parent node of t_2.

   Compare the inductive and coinductive interpretations for the same definition.

3. $div/2$ s.t. $div(t_1, t_2)$ holds iff the natural number t_1 divides the natural number t_2 (only for the inductive case).

   Hint: define first the predicate $sub/3$ s.t. $sub(t_1, t_2, t_3)$ holds iff t_1, t_2, and t_3 are natural numbers s.t. $t_3 = t_1 - t_2$.