Declarative Programming and (Co)Induction
Module 2
Prolog lab 2

Davide Ancona and Elena Zucca
University of Genova

PhD Course, DIBRIS, June 26-27, 2014

Easy exercises

1. Try out non ground queries, with the predicates defined in exercise 3 of Prolog lab 1. Consider both inductive and coinductive predicates.

2. Define the predicate $add/3$ s.t. $add(t_1, t_2, t_3)$ holds iff t_1, t_2, and t_3 are natural numbers and $t_3 = t_1 + t_2$.

   Try out the goal ?- $add(N, M, s(s(z)))$ with both the inductive and coinductive interpretations.

3. Implement the typechecking rules of the simply typed lambda-calculus as defined on slide 30, Module 1, "Small Step Semantics, Lambda Calculus and Type Systems".

   Hints: Define the predicate *typeof/2* for ground terms (that is, where the type environment is implicitly empty), based on the auxiliary predicate *typeof/3* that takes also a type environment.

   To implement the type environment you may use the library *assoc* (with :- use_module(library(assoc)).) and then the three predicates *empty_assoc/1* (to return an empty environment), *get_assoc/3* (to check the type of a variable), and *put_assoc/4* to update an environment (see the on-line documentation at `http://www.swi-prolog.org/`).

   For representing the terms of the language, see the suggested syntax in the queries below.

   ?- $E = fun(x : bool- > x), typeof(E, RT)$.
   ?- $E = fun(x : T- > x), typeof(E, RT)$.
   ?- $E = fun(f1 : T1- > fun(f2 : T2- > fun(x : T- > app(f1, app(f2, x))))), typeof(E, RT)$.
   ?- $E = fun(x : T- > app(x, x)), typeof(E, RT)$.
   ?- $E = fun(x : T- > app(x, x)), typeof(app(E, E), RT)$.
   ?- $E = fun(x : T- > app(x, x)), typeof(app(app(app(E, E), true), false), RT)$.
   ?- $E = fun(x : X- > fun(y : Y- > if(x, y, x))), typeof(E, RT)$.
   ?- $E1 = fun(x : X- > app(f, app(x, x))), E = fun(f : F- > app(E1, E1)), typeof(E, RT)$.
   ?- $F = fun(x : T- > x), E = fun(f : FT- > if(true, app(f, true), app(f, false))), typeof(E, RT)$.
   ?- $F = fun(x : T- > x), E = fun(f : FT- > if(true, app(f, true), app(f, F))), typeof(E, RT)$.

   Try out the queries with both inductive and coinductive interpretation, and motivate the computed answers.

4. Define the coinductive predicate *add/3* which computes addition between repeating decimals.

   Hints: use built-in numbers to represent digits, and built-in predicates to compute addition, and integer division with remainder (example $X\ is\ 3 + 5, Y\ is\ 5//10, Z\ is\ 5\ mod\ 10$).