# Declarative Programming and (Co)Induction
# **Prolog exercises, part 2**

Davide Ancona, Giovanni Lagorio and Elena Zucca
University of Genova

PhD Course, DISI, June 13-17, 2011

**Easy exercises**

1. try out the following non ground goals, with the predicates defined in the previous lab experience. For instance

   ?- $is\_nat(N)$.
   ?- $odd(N)$.
   ?- $even(N)$.
   ?- $div(s(s(z))), N)$.
   ?- $div(N, s(s(s(s(s(z)))))))$.

   Consider both inductive and coinductive cases.

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.

**Less easy exercise**

1. Implement the typechecking rules of the simply typed lambda-calculus as defined on pag.52 in the slides on functional programming.

   **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/pldoc/refman/`).

   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, and motivate the computed answers.

   **Very challenging exercise**

   (a) Rational numbers in the interval $[0, 1]$ can be rapresented by regular lists of digits. For instance the list $L$ s.t. $L = [1|L]$ corresponds to the repeating decimal $0.\hat{1}$, that is, $\frac{1}{9}$ (by the way, recall that $0.\hat{9} = 1$, see `http://en.wikipedia.org/wiki/0.999...`). Note also that finite decimal numbers can be seen as a specific case of repeating decimals; for instance, $0.1 = 0.1\hat{0}$.
   Using regular lists to represent decimals in $[0.1]$, implements the predicate *addrat/3* s.t. $addrat(t_1, t_2, t_3)$ holds iff $t_1$, $t_2$, and $t_3$ are repeating decimals and $t_1 + t_2 = t_3$.

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

Use an auxiliary predicates that computes also the regular list of all carries for all positions.