# Assignment 2
# Introduction to Computational Logic, SS 2011

Prof. Dr. Gert Smolka, Dr. Chad Brown
www.ps.uni-saarland.de/courses/cl-ss11/

Read in the lecture notes: Chapter 2

**Note:** It is very important to do all the examples in the lecture notes and the exercises below in the system Coq.

**Exercise 2.1** An operation taking two arguments can be represented either as a function taking its arguments one by one (**cascaded representation**) or as a function taking both arguments bundled in one pair (**cartesian representation**). While the cascaded representation is natural in Coq, the cartesian representation is commonly used in mathematics. Define functions

$$car : \; forall\, X\; Y\; Z : Type,\; (X \to Y \to Z) \to (pair\; X\; Y \to Z)$$
$$cas : \; forall\, X\; Y\; Z : Type,\; (pair\; X\; Y \to Z) \to (X \to Y \to Z)$$

that translate between the cascaded and cartesian representation and prove the following lemmas.

```
Lemma car_P (X Y Z :Type)  (f : X -> Y -> Z) (x :X) (y :Y) :  car f (P x y) = f x y.
Lemma cas_P (X Y Z :Type)  (f : pair X Y -> Z) (x :X) (y :Y) :  cas f x y = f  (P x y).
```

The type arguments of *car* and *cas* are assumed to be implicit.

**Exercise 2.2** Prove $mul\; x\; y = iter\; x\; (add\; y)\; O$ for all numbers $x$ and $y$.

**Exercise 2.3** Prove the following lemma.

```
Lemma iter_move (X : Type) (f : X -> X) (x : X) (n : nat) :
 iter  (S n)  f  x = iter  n f  (f  x).
```

**Exercise 2.4** Prove the following lemmas.

```
Lemma app_asso (X : Type) (xs ys zs : list X) :
app (app xs ys) zs = app xs (app ys zs).

Lemma length_app (X : Type) (xs ys : list X) :
length (app xs ys) = add (length xs) (length ys).

Lemma rev_app (X : Type) (xs ys : list X) :
rev (app xs ys) = app (rev ys) (rev xs).

Lemma rev_rev (X : Type) (xs : list X) :
rev (rev xs) = xs.
```

**Exercise 2.5** Here is a tail-recursive function that obtains the length of a list with an accumulator argument.

```
Fixpoint lengthi {X : Type} (xs : list X) (a : nat) :=
match xs with
| nil => a
| cons _ xr => lengthi xr (S a)
end.
```

Proof the following lemmas.

```
Lemma lengthi_length {X : Type} (xs : list X) (a : nat) :
lengthi xs a = add (length xs) a.
```

```
Lemma length_lengthi {X : Type} (xs : list X) :
length xs = lengthi xs O.
```