Lab Session 2
Proof by Induction

Construction and Verification of Software

Nova School of Science and Technology
Mário Pereira   mjp.pereira@fct.unl.pt

Version of September 16, 2025

Preface

From this point on, we will start using some auxiliary definitions from the Coq standard library. You shall start your Coq developments by the following lines:

```coq
From Coq Require Import Nat. (* Library on natural numbers *)
From Coq Require Import Bool.Bool. (* Library on Boolean values *)
From Coq Require Import Lia. (* Linear integer arithmetic decision
    procedure *)
From Coq Require Import Lists.List. (* Library on lists *)
Import ListNotations. (* Syntactic-sugar for list values *)
```

1 Proof by Induction

1.1 Natural Numbers

Exercise 1. Prove the following lemma:

```coq
Lemma add_0_r : ∀ n: nat,
  n + 0 = n.
```

□

Exercise 2. Prove the following lemma:

```coq
Lemma mult_0_r : ∀ n: nat,
  n × 0 = 0.
```

□

Exercise 3. Prove the following lemma:

```coq
Lemma plus_n_Sm : ∀ n m: nat,
  S (n + m) = n + S m.
```

□

Exercise 4. Prove the following lemma:

```coq
Lemma add_comm : ∀ n m: nat,
  n + m = m + n.
```
Coq

□

Exercise 5. Prove the following lemma:

```coq
Lemma add_assoc : ∀ n m p: nat,
  n + (m + p) = (n + m) + p.
```
Coq

□

Exercise 6. Given the following definitions

```ocaml
let rec power (n: nat) (m: nat) : nat =
  match m with
  | O => S O
  | S m' => mult n (power n m')

let rec pow (n: nat) (m: nat) (r: nat) : nat :=
  match m with
  | O => r
  | S m' => pow n m' (r * n)

let power_alt (n: nat) (m: nat) : nat =
  pow n m 1
```
OCaml

show that the following Lemma holds:

```coq
Lemma power_alt_correct : ∀ n m: nat,
  power n m = power_alt n m.
```
Coq

Hint: you might want to prove an auxiliary Lemma about the result of `pow p q`, for any natural numbers `p` and `q`.

□

Exercise 7. Provide a definition for the function `fact`, of type `nat` → `nat`, which computes the factorial of its argument. □

Exercise 8. Consider the following alternative formulation of the factorial function:

```coq
Fixpoint fact_acc (n: nat) (a: nat) : nat :=
  match n with
  | O ⇒ a
  | S n' ⇒ fact_acc n' (n × a)
  end.

Definition fact_alt (n: nat) : nat :=
  fact_acc n 1.
```
Coq

Prove that `fact_alt` is a correct implementation of the factorial function, *i.e.*, prove the following statement:

```coq
Lemma fact_alt_correct : ∀ n: nat,
  fact_alt n = fact n.
```
Coq

□

1.2 Lists

Exercise 9. Prove the following lemma:

```coq
Lemma nil_app: ∀ l : list nat,
  [ ] ++ l = l.
```
Coq

□

Exercise 10. Prove the following lemma:

```coq
Lemma app_assoc : ∀ l1 l2 l3: list nat,
  (l1 ++ l2) ++ l3 = l1 ++ (l2 ++ l3).
```
Coq

□

Consider the following definition of a function that reverses a list of natural numbers:

```coq
Fixpoint rev (l: list nat) : list nat :=
  match l with
  | [ ] ⇒ [ ]
  | h :: t ⇒ rev t ++ [h]
  end.
```
Coq

Exercise 11. Prove the following lemma:

```coq
Lemma app_nil_r : ∀ l : list nat,
  l ++ [ ] = l.
```
Coq

□

Exercise 12. Prove the following lemma:

```coq
Lemma rev_length : ∀ l: list nat,
  length (rev l) = length l.
```
Coq

□

Exercise 13. Prove the following lemma:

```coq
Lemma rev_app_distr : ∀ l1 l2 : list,
  rev (l1 ++ l2) = rev l2 ++ rev l1.
```
Coq

□

Exercise 14. Prove the following lemma:

```coq
Lemma rev_involutive : ∀ l : list,
  rev (rev l) = l.
```
Coq

□

1.3 Trees

Consider the following definition for a binary tree of natural numbers:

```coq
Inductive tree : Type :=
| Leaf : tree
| Node (l: tree) (v: nat) (r: tree).
```
Coq

Exercise 15. Provide a definition for the function `mirror`, of type `tree` $\rightarrow$ `tree`, which "mirrors" its argument.

Your `mirror` check should check the following examples:

Coq
```
Example mirror1 : mirror Leaf = Leaf.
Proof. reflexivity. Qed.

Example mirror2 :
  mirror (Node (Node Leaf 0 Leaf) 1 (Node Leaf 2 Leaf)) =
  Node (Node Leaf 2 Leaf) 1 (Node Leaf 0 Leaf).
Proof. reflexivity. Qed.
```

□

Exercise 16. Prove the following lemma:

Coq
```
Lemma mirror_involutive: ∀ t: tree,
  mirror (mirror t) = t.
```

□