My Coq Reference Sheet

2021

1 Nat

1.1 Definition

```
Inductive nat : Type :=
  | O
  | S (n : nat).
```

1.2 Important Functions

- eqb (n m : nat) : bool := returns true iff $n = m$. Notation is $x =?y$
- plus (n : nat) (m : nat) : nat := $n + m$
- mult (n m : nat) : nat := $n * m$
- minus (n m:nat) : nat := $n - m$
- exp (base power : nat) : nat := $base^{power}$
- pred (n : nat) : nat := returns the number before n (pred 2 $\Rightarrow$ 1, pred 0 $\Rightarrow$ 0)
- even (n:nat) : bool := returns if a number is even
- odd (n:nat) : bool := ¬even n
- factorial (n:nat) : nat := $n!$
- leb (n m : nat) : bool := returns true iff $n \leq m$. Notation is $x <=?y$

1.3 Theorems

- plus_O_n : ∀ n : nat, 0 + n = n.
- add_0_r : ∀ n:nat, n + 0 = n.
- plus_n_Sm : ∀ n m : nat, S (n + m) = n + (S m). (also known as sum_add_one)
- add_comm : ∀ n m : nat, n + m = m + n.
- add_assoc : ∀ n m p : nat, n + (m + p) = (n + m) + p.
- plus_1_l : ∀ n:nat, 1 + n = S n.
- mult_0_l : ∀ n:nat, 0 * n = 0.

- mul_0_r : ∀ n:nat, n * 0 = 0.
- mult_n_1 : ∀ p : nat, p * 1 = p.
- n_mul_1_plus_k : ∀ n k : nat, n * (S k) = n + n * k .
- mul_comm : ∀ m n : nat, m * n = n * m.
- minus_n_n : ∀ n, minus n n = 0.
- S_injective : ∀ (n m : nat), S n = S m ⇒ n = m.
- eq_implies_succ_equal : ∀ (n m : nat), n = m ⇒ S n = S m.
- eqb_0_l : ∀ n, 0 =? n = true ⇒ n = 0.
- eqb_true : ∀ n m, n =? m = true ⇒ n = m.
- plus_n_n_injective : ∀ n m, n + n = m + m ⇒ n = m.
- zero_or_succ : ∀ n : nat, n = 0 ∨ n = S (pred n).

2 List

2.1 Definition

```
Inductive list (X:Type) : Type :=
  | nil
  | cons (x : X) (l : list X).
```

2.2 Important Functions

- In (x : A) (l : list A) : Prop := returns a prop relating whether x is in l.
- rev (l:list X) : list X := reverses l.
- length (l : list X) : nat := length of l.
- repeat (x : X) (count : nat) : list X := returns list of *count* instances of x.
- app (l1 l2 : list X) : list X := appends l2 to the end of l1. Notation is l1 ++ l2.
- combine (lx : list X) (ly : list Y) : list (X*Y) := two list are threaded to become a list of pairs
- split (l : list (X*Y)) : (list X) * (list Y) := list of pairs is split to become a a pair of lists.
- nth_error (l : list X) (n : nat): option X := returns the nth element of a list in the form of Some X or it returns None.

2.3 Important Notation

- x::y := cons x y
- [] := nil
- [x; ...; y] := cons x ... (cons y nil)
- x ++ y := app x y.

2.4 Theorems

- app_nil_r : ∀ l:list X, l ++ [] = l.
- app_assoc : ∀ A (l m n:list A), l ++ m ++ n = (l ++ m) ++ n.
- app_length : ∀ (l1 l2 : list X), length (l1 ++ l2) = length l1 + length l2.
- rev_app_distr: ∀ (l1 l2 : list X), rev (l1 ++ l2) = rev l2 ++ rev l1.

3 Bool

3.1 Definition

```
Inductive bool : Type :=
  | true
  | false.
```

3.2 Important Functions

- negb (b:bool) : bool := $\neg b$
- andb (b1:bool) (b2:bool) : bool := b1 ∧ b2
- orb (b1:bool) (b2:bool) : bool := b1 ∨ b2
- nandb (b1:bool) (b2:bool) : bool := $\neg(b1 \wedge b2)$.

3.3 Theorems

- negb_involutive : ∀ b : bool, negb (negb b) = b.
- andb_commutative : ∀ g c, $g \wedge c = c \wedge g$.

4 Bin

4.1 Definition

```
Inductive bin : Type :=
  | Z
  | B0 (n : bin)
  | B1 (n : bin).
```

4.2 Important Functions

- incr (m:bin) : bin := increment m
- bin_to_nat (m:bin) : nat := binary number to natural number formatting

5 Prod / Pair

5.1 Definition

```
Inductive prod (X Y : Type) : Type :=
| pair (x : X) (y : Y).
```

5.2 Important Functions

- fst (p : X * Y) : X := returns first element of pair
- snd (p : X * Y) : Y := returns second element of pair

5.3 Important Notation

- (x,y) := pair x y.
- X * Y := prod X Y

6 Option

6.1 Definition

```
Inductive option (X:Type) : Type :=
  | Some (x : X)
  | None.
```

7 Important Functional Theorems

- f_equal : $\forall$ (f: A $\Rightarrow$ B) (x y: A), x = y $\Rightarrow$ f x = f y.
- injective (f : A $\Rightarrow$ B) := $\forall$ x y : A, f x = f y $\Rightarrow$ x = y.

8 Important Logical Theorems

- conjI : A $\Rightarrow$ B $\Rightarrow$ (A $\wedge$ B).
- conjE1 : (A $\vee$ B) $\Rightarrow$ A .
- and_commutes : (A $\wedge$ B) $\Rightarrow$ (B $\wedge$ A) .
- or_swap : (A $\vee$ B) $\Rightarrow$ (B $\vee$ A).
- contrapos : (A $\Rightarrow$ B) $\Rightarrow$ $\neg$B $\Rightarrow$ $\neg$A.
- dnI : A $\Rightarrow$ $\neg\neg$A.
- orAB : (A $\vee$ B) $\Rightarrow$ $\neg$ ($\neg$A $\wedge$ $\neg$B).
- andAB : (A $\wedge$ B) $\Rightarrow$ $\neg$ ($\neg$A $\vee$ $\neg$B).
- arrowAB : (A $\Rightarrow$ B) $\Rightarrow$ $\neg$ (A $\wedge$ $\neg$ B).

- iff_sym : (A $\iff$ B) $\Rightarrow$ (B $\iff$ A).
- iffNot : (A $\iff$ B) $\Rightarrow$ ($\neg$A $\iff$ $\neg$B).

9 Filter

9.1 Definition

```
Fixpoint filter {X:Type} (test: X->bool) (l:list X) : list X :=
  match l with
  | [] => []
  | h :: t =>
    if test h then h :: (filter test t)
    else filter test t
  end.
```

10 Map

10.1 Definition

```
Fixpoint map {X Y : Type} (f : X->Y) (l : list X) : list Y :=
  match l with
  | []     => []
  | h :: t => (f h) :: (map f t)
  end.
```

10.2 Theorems

- gamma : $\forall$ (f: X$\Rightarrow$Y) (l : list X) (x : X), map f (l ++ [x]) = map f l ++ [f x].

11 Fold

11.1 Definition

```
Fixpoint fold {X Y: Type} (f : X->Y->Y) (l : list X) (b : Y) : Y :=
  match l with
  | nil => b
  | h :: t => f h (fold f t b)
  end.
```