Adrien Portier

Software Foundations - Logical Foundations (1) Basics

Table des matières

Manipulation d'un type inductive de base (jours)

Nous pouvons créer le type jour en énumérant tous ses éléments :

In [1]:
```
Inductive jour : Type :=
| Lundi
| Mardi
| Mercredi
| Jeudi
| Vendredi
| Samedi
| Dimanche.
```

Out[1]: Cell evaluated.

Avec ce type "jour", on peut construire une fonction qui associe à un jour, le prochain jour :

In [2]:
```
Definition prochain(d : jour) : jour :=
match d with
| Lundi => Mardi
| Mardi => Mercredi
| Mercredi => Jeudi
| Jeudi => Vendredi
| Vendredi => Samedi
| Samedi => Dimanche
| Dimanche => Lundi
end.
```

Out[2]: Cell evaluated.

Quel est le jour qui vient après Lundi ?

In [3]:
```
Compute(prochain Lundi).
```

Out[3]:
```
     = Mardi
     : jour
```

Cell evaluated.

Quel est le jour après celui qui vient après Mercredi ?

In [4]:
```
Compute(prochain(prochain(Mercredi))).
```

Out[4]:
```
     = Vendredi
     : jour
```

Cell evaluated.

Quel est le jour après celui qui vient après celui qui vient après Samedi ?

In [5]:
```
Compute(prochain(prochain(prochain(Samedi)))).
```

Out[5]:
```
     = Mardi
     : jour
```

Cell evaluated.

Nous voulons vérifier que si on est Samedi, alors après-demain est Lundi :

In [6]:
```
Example test_prochain_jour:
  (prochain (prochain Samedi)) = Lundi.
```

Out[6]:
```
Proving: test_prochain_jour

1 subgoal

-------------- (1/1)
prochain (prochain Samedi) = Lundi
```

Cell evaluated.

Nous simplifions l'expression (prochain (prochain _)) et par réflexivité, nous avons que Lundi = Lundi :

In [7]:
```
Proof. simpl. reflexivity. Qed.
```

Out[7]: Cell evaluated.

Booléens

Créons le type booléen bien qu'il soit déjà implémenté dans certaines librairies,

In [11]:
```
Inductive bool : Type :=
| true
| false.
```

Out[11]: Cell evaluated.

et nous définissons la négation ¬ par :

In [12]:
```
Definition negb(b : bool) : bool :=
match b with
| true => false
| false => true
end.
```

Out[12]: Cell evaluated.

que nous pouvons vérifier comme précédemment,

In [13]:
```
Example test_negf:
  negb(false) = true.
Proof. simpl. reflexivity. Qed.
```

Out[13]: Cell evaluated.

En regardant la table de vérité pour "ou", ∨, nous arrivons facilement à définir

In [18]:
```
Definition orb(b1 : bool) (b2 : bool) : bool :=
match b1 with
| true => true
| false => b2
end.
```

Out[18]: Cell evaluated.

que nous pouvons vérifier par la table complète :

In [19]:
```
Example TrueOrFalse:
  orb(true)(false) = true.
Proof. simpl. reflexivity. Qed.

Example TrueOrTrue:
  orb(true)(true) = true.
Proof. simpl. reflexivity. Qed.

Example FalseOrFalse:
  orb(false)(false) = false.
Proof. simpl. reflexivity. Qed.

Example FalseOrTrue:
  orb(false)(true) = true.
Proof. simpl. reflexivity. Qed.
```

Out[19]: Cell evaluated.

Nous faisons de même pour "et", ∧, que nous définissons par

In [39]:
```
Definition andb(b1:bool)(b2:bool):bool:=
match b1 with
| true => b2
| false => false
end.
```

Out[39]: Cell evaluated.

nous vérifions la table de vérité :

In [40]:
```
Example and1:
  andb(true)(false) = false.
Proof. simpl. reflexivity. Qed.

Example and2:
  andb(true)(true) = true.
Proof. simpl. reflexivity. Qed.

Example and3:
  andb(false)(false) = false.
Proof. simpl. reflexivity. Qed.

Example and4:
  andb(false)(true) = false.
Proof. simpl. reflexivity. Qed.
```

Out[40]: Cell evaluated.

Au lieu de devoir utiliser ces définitions assez pénibles, peu pratiques, nous pouvons définir des notations pour utiliser orb ET andb !

In [60]:
```
Notation "x && y" := (andb x y).
Notation "x || y" := (orb x y ).
```

Out[60]: Cell evaluated.

Nous pouvons prendre quelques exemples :

In [62]:
```
Example FalseOrFalseOrFalseOrTrue:
  false || false || false || true = true.
Proof. simpl. reflexivity. Qed.

Example TrueAndTrueAndTrueAndFalse:
  true && true && true && false = false.
Proof. simpl. reflexivity. Qed.

Example TrueAndTrue_OrFalse:
  (true && true) || false = true.
Proof. simpl. reflexivity. Qed.
```

Out[62]: Cell evaluated.

Nous pouvons aussi définir une fonction nand, "¬∧", ,

In [77]:
```
Definition nandb (b1:bool) (b2:bool) : bool :=
match b1 with
| true => negb(b2)
| false => true
end.
```

Out[77]: Cell evaluated.

qu'on vérifie facilement par un exemple,

In [95]:
```
Example NandbTrueFalse: (nandb true false) = negb (andb(true)(false)).B
Proof. simpl. reflexivity. Qed.
Example NandbFalseFalse: (nandb false false) = negb (andb(false)(false)).
Proof. simpl. reflexivity. Qed.
Example NandbFalseTrue: (nandb false true) = negb (andb(false)(false)).
Proof. simpl. reflexivity. Qed.
Example NandbTrueTrue: (nandb true true) = negb (andb(true)(true)).
Proof. simpl. reflexivity. Qed.
```

Out[95]: Cell evaluated.

Bien évidemment, le choix des noms attirés aux exemples est libre, si on remplaçait "NandbTrueTrue" par "Tomate" cela fonctionnerait tout autant.

Nous pouvons fournir une définition alternative aux définitions sur les booléens en utilisant des instructions conditionnelles :

In [78]:
```
Definition negb' (b : bool) : bool :=
if b then false
else true.

Definition andb' (b1:bool)(b2:bool) : bool :=
if b1 then b2
else false.

Definition orb' (b1:bool)(b2:bool) : bool :=
if b1 then true
else false.
```

Out[78]: Cell evaluated.

> Plus compact, mais plus dur à comprendre dès la première lecture du code

In [ ]: