UPMC/info/master/SVP-5I554
Examen réparti I
Corrigé

Novembre 2016

Tous documents autorisés
Les preuves demandées sont à rédiger en langue naturelle, mais avec rigueur. Adoptez le style utilisé dans les notes de cours.

On définit la fonction `nbocc` qui compte le nombre d'occurrences d'un élément dans une liste d'entiers.

```
Fixpoint nbocc (x:nat) (xs:list nat) : nat :=
  match xs with
      nil => 0
    | (cons y ys) =>
      if (y = x) then (S (nbocc x ys))
      else (nbocc x ys)
  end.
```

Remarque : pour simplifier, on confond, sur le papier, l'égalité *propositionnelle* (de type `nat -> nat -> Prop`) et l'égalité *booléenne* (de type `nat -> nat -> bool`). On pourra ainsi raisonner libre par cas sur l'égalité : *ou bien* $x = y$ *; ou bien* $x \neq y$.

Question 1 Démontrez

```
forall (z:nat) (xs ys:list nat), (nbocc z (app xs ys)) = ((nbocc z xs) + (nbocc z ys)).
```

où `app` est la fonction de concaténation des listes.

Réponse

On montre `(nbocc z (app xs ys)) = ((nbocc z xs) + (nbocc z ys))` par induction sur `xs` :

— si `xs`≡`nil`, il faut montrer `(nbocc z (app nil ys)) = ((nbocc z nil) + (nbocc z ys))`.
On a
`(nbocc z (app nil ys))` = `(nbocc z ys)` (par déf. de `app`)
On a
`(nbocc z nil) + (nbocc z ys)` = `0 + (nbocc z ys)` (par déf. de `nbocc`)
= `(nbocc z ys)` (par déf. de l'addition)

CQFD

— si `xs`≡`(cons x' xs')`, on suppose
HR : `(nbocc z (app xs' ys)) = ((nbocc z xs') + (nbocc z ys))`,
il faut montrer
`(nbocc z (app (cons x' xs') ys)) = ((nbocc z (cons x' xs')) + (nbocc z ys))`.
On a
`(nbocc z (app (cons x' xs') ys))` = `(nbocc (cons x' (app xs' ys)))` (par déf. de `app`)

Il faut donc montrer que
`(nbocc z (cons x' (app xs' ys))) = ((nbocc z (cons x' xs')) + (nbocc z ys))`.
Par cas sur l'égalité (`z=x'` ou `z`$\neq$`x'`) :

— si `z=x'`, on a

`(nbocc z (cons x' (app xs' ys)))`	=	`S(nbocc z (app xs' ys))`
		(par déf. de `nbocc`)
	=	`S((nbocc z xs') + (nbocc z ys))`
		(par HR.)
`(nbocc z (cons x' xs')) + (nbocc z ys)`	=	`S(nbocc z xs') + (nbocc z ys)`
		(par déf. `nbocc`)
	=	`(S((nbocc z xs') + (nbocc z ys))`
		(par déf. de l'addition)

CQFD

— si `z`$\neq$`x'`, on a

`(nbocc z (cons x' (app xs' ys)))`	=	`(nbocc z (app xs' ys))`
		(par déf. de `nbocc`)
	=	`(nbocc z xs') + (nbocc z ys)`
		(par HR.)
`(nbocc z (cons x' xs')) + (nbocc z ys)`	=	`(nbocc z xs') + (nbocc z ys)`
		(par déf. `nbocc`)

CQFD

Question 2 on rappelle la définition de la fonction de retournement d'une liste, la fonction `rev` :

```
Fixpoint rev {A:Set} (xs:list A) :=
  match xs with
    nil => nil
  | (cons x xs) => (app (rev xs) (cons x nil))
  end.
```

Démontrez

```
forall (x:nat) (xs:list nat), (nbocc x xs) = (nbocc x (rev xs)).
```

Nota : vous pouvez utiliser tous les résultats arithmétiques standards, comme $0 + n = n$, $n + 0 = n$, $n+m = m+n$, $0 \leq n$, $0 \not< 0$, etc... Pensez simplement à signaler leur utilisation. Ceci vaut pour l'ensemble des questions.

Réponse

On montre `(nbocc x xs) = (nbocc x (rev xs))` par induction sur `xs` :

— si `xs`$\equiv$`nil`, il faut montrer `(nbocc x nil) = (nbocc x (rev nil))`.
Par définition, `(rev nil) = nil` donc `(nbocc x (rev nil)) = (nbocc x nil)`.

— si `xs`$\equiv$`(cons x' xs')`, on suppose
HR : `(nbocc x xs') = (nbocc x (rev xs'))`
il faut montrer `(nbocc x (cons x' xs')) = (nbocc x (rev (cons x' xs')))`.
On a

`(nbocc x (rev (cons x' xs')))`	=	`(nbocc (app (rev xs') (cons x' nil)))`
		(par déf. de `rev`)
	=	`(nbocc x (rev xs')) + (nbocc x (cons x' nil))`
		(par Question 1)
	=	`(nbocc x xs') + (nbocc x (cons x' nil))`
		(par HR.)

Il faut donc montrer que `(nbocc x (cons x' xs')) = (nbocc x xs') + (nbocc x (cons x' nil))`
Par cas sur `x=x'` :

— si `x=x'`, on a

`(nbocc x (cons x' xs'))`	$=$	`S(nbocc x xs')` (par déf. de `nbocc`)
`(nbocc x xs') + (nbocc x (cons x' nil))`	$=$	`(nbocc x xs') + 1` (par déf. de `nbocc`)
	$=$	`S(nbocc x xs')` (par arith.)

CQFD

— si `x`$\neq$`x'`, on a

`(nbocc x (cons x' xs'))`	$=$	`(nbocc x xs')` (par déf. de `nbocc`)
`(nbocc x xs') + (nbocc x (cons x' nil))`	$=$	`(nbocc x xs') + 0` (par déf. de `nbocc`)
	$=$	`(nbocc x xs')` (par arith.)

CQFD

Question 3 on définit le prédicat d'appartenance à une liste de la manière suivante :

```
Inductive Mem {A:Set} : A -> list A -> Prop :=
  Mem_alt : forall (x y:A) (xs:list A), (x=y) \/ (Mem x xs) -> (Mem x (cons y xs)).
```

Ce prédicat signifie que, pour tout `xs ys:list A`,
`(Mem x (conx y xs))` si et seulement si `x=y` ou `(Mem x xs)`.
Notez que, par définition, `(Mem x nil)` est toujours faux.

Démontrez

```
forall (x:nat) (xs:list nat), (0 < (nbocc x xs)) -> (Mem x xs).
```

Réponse

on montre `(0 < (nbocc x xs)) -> (Mem x xs)` par induction sur `xs` :

- si `xs`$\equiv$`nil`, il faut montrer `(0 < (nbocc x nil)) -> (Mem x nil)`.
  On a que `(nbocc x nil)=0`. Il faut donc montrer que `0<0 -> (Mem x nil)`. Ce qui est trivialement vérifié puisque `0<0` est faux.
- si `xs`$\equiv$`(cons x' xs')`, on suppose
  HR : `(0 < (nbocc x xs')) -> (Mem x xs')`
  il faut montrer `(0 < (nbocc x (cons x' xs'))) -> (Mem x (cons x' xs'))`.
  On suppose H : `0 < (nbocc x (cons x' xs'))`
  et on montre `(Mem x (cons x' xs))`.
  Par déf. de `Mem`, il suffit de montrer `(x=x') \/ (Mem x xs')`.
  Par cas sur `x=x'` :
  - si `x=x'`, c'est immédiat.
  - si `x`$\neq$`x'`, on montre que `(Mem x xs')`.
    Par HR. il suffit de montrer que `0<(nbocc x xs')`.
    Par déf. de `nbocc`, `(nbocc x (cons x' xs'))=(nbocc x xs')`. Donc, par H : `0<(nbocc x xs')`.

Question 4 démontrez

```
forall (x:nat) (xs:list nat), (Mem x xs) -> (0 < (nbocc x xs)).
```

Nota : souvenez vous que `(x < y) -> (y` $\leq$ `z) -> (x < z)`, pour tout `x y z:nat`.

Réponse

On montre `(Mem x xs) -> (0 < (nbocc x xs))` par induction sur `xs'`

- si `xs`≡`nil`, il faut montrer `(Mem x nil) -> (0 < (nbocc x nil))`.
  Ce qui est trivial puisque `(Mem x nil)` est faux, par déf. de `Mem`.
- si `xs`≡`(cons x' xs')`, on suppose
  HR : `(Mem x xs') -> (0 < (nbocc x xs'))`
  il faut montrer `(Mem x (cons x' xs')) -> (0 < (nbocc x (cons x' xs')))`
  On suppose H : `(Mem x (cons x' xs'))`
  et on montre `(0 < (nbocc x (cons x' xs')))`.
  Par cas sur l'égalité `x`=`x'`
  - si `x`=`x'`, par déf. de `nbocc`, il faut montrer que `0 < S(nbocc x xs')`. Ce qui est donné par arith.
  - si `x`≠`x'`, par déf. de `nbocc`, il faut montrer `0 < (nbocc x xs')`.
    En appliquant HR, il faut montrer `(Mem x xs')`.
    Par H et déf. de `Mem`, on a que `(x=x') \/ (Mem x xs')`. Comme on est dans le cas où `x`≠`x'`, c'est que `(Mem x xs')`.