

Preuves assistées par ordinateur – TD n° 6

Les listes en Coq

En Coq, l' type d s *listes polymorphes* est défini à l'aide d la définition inductiv suivant

```
Inductive list (A : Type) : Type :=
| nil : list A
| cons : A -> list A -> list A.
```

qui introduit dans l' nvironn m nt courant un constructeur d typ `list : Type → Type` (qui à chaque typ `A : Type` associ l typ d list s correspondant `list A : Type`) ainsi qu l s d ux construct urs polymorph s

```
nil   : for ll A : Type, list A
cons  : for ll A : Type, A -> list A -> list A
```

Contrairement à Caml, l polymorphism est *explicite* n Coq. Ceci s traduit à la fois :

- dans l s typ s d s construct urs `nil` t `cons`, où l'on voit apparaître xplicit m nt la quantification d typ `for ll A : Type, ...` (qui d m ur implicite n Caml)
- dans l'utilisation d c s construct urs, qui att nd nt xplicit m nt l paramètre d typ `A` comm pr mi r argument. Ainsi :
 - la list vid d'éléments d `A` s'écrit : `nil A` (`: list A`);
 - l'ajout d l'élément `x : A` à un list `l : list A` s'écrit : `cons A x l` (`: list A`).

Remarque : En Coq, la quantification univers ll `for ll x : T, U(x)` introduit un *type fonctionnel* qui généralis l typ flèche `T -> U`, t qu'on appll *produit dépendant*.

Question préliminaire Le paramètre `A : Type` étant fixé, quel principe d récurrence est-il naturel d'introduire pour raisonner sur l s list s ? On vérifiera la réponse avec `Check list_ind`.

Exercice 1 – Concaténation de listes

L'opération (polymorph) d concaténation d list s est défini n Coq par :

```
Fixpoint conc t (A : Type) (l1 l2 : list A) : list A :=
  m tch l1 with
  | nil      => l2
  | cons x tl => cons A x (conc t A tl l2)
end.
```

1. Quel st l typ d `conc t` ?
2. Montrer n Coq l s propositions :

```
for ll (A : Type) (l : list A), conc t A (nil A) l = l
for ll (A : Type) (l : list A), conc t A l (nil A) = l
```

Laquelle d c s d ux propositions correspond à une égalité définitionnell ?

3. Montrer qu l'opération d concaténation est associativ .

Exercice 2 – Longueur

1. Définir une fonction `length` à deux arguments $A : \text{Type}$ et $l : \text{list } A$ telle que `length A l` retourne la longueur de la liste l (exprimée comme un objet de type `nat`).
2. Quel est le type de la fonction `length`?
3. Montrer que $\text{length } A (\text{concat } A l_1 l_2) = \text{length } A l_1 + \text{length } A l_2$.

Exercice 3 – Retournement

1. Définir une fonction `reverse` : `forall (A : Type), list A -> list A` retournant la liste donnée dans l'ordre inverse. On pourra pour cela introduire une fonction auxiliaire
$$\text{rev_ppend} : \text{forall } (A : \text{Type}), \text{list } A \rightarrow \text{list } A \rightarrow \text{list } A$$
qui retourne la première des deux listes (i.e. le 2^e argument de `rev_ppend`) et la concatène avec la seconde (i.e. le 3^e argument).
2. Montrer que $\text{length } A (\text{reverse } A l) = \text{length } A l$.
3. Montrer que $\text{reverse } A (\text{concat } A l_1 l_2) = \text{concat } A (\text{reverse } A l_2) (\text{reverse } A l_1)$

Remarque : Pour résoudre ces questions, il est utile d'énoncer des lemmes intermédiaires !