RICM2 - 2004/05
Langage et Programmation 2

Manipulation de listes

Exercice 1: Longueur d'une liste
Décrire une fonction qui à une liste de caractères associe la longueur de cette liste.

Spécification
long : une liste de caractères → un entier

Relations de récurrence
long([])=0
long(e.l)=1+long(l)

Caml
let rec liste nba liste = match liste with
[] -> 0
|(e : :l) -> 1+long(l)

Exercice 2: Nombre d'occurrences
Décrire une fonction qui à une séquence de caractère associe le nombre de " a " de cette séquence.

Spécification
nba : une liste de caractères → un entier

Relations de récurrence
nba([])=0
nba(e.l)=(si e='a'alors1 sinon0)+long(l)

Caml
let rec nba l = match l with
[] -> 0
|(x : :l1) -> (if x='a' then 1 else 0) + nba l1 ;

Exercice 3: Concaténation
Décrire une fonction qui à deux listes de caractères associe la concaténation

de ces deux listes.

Spécification

concat : deux listes de caractères → une liste de caractères

Relations de récurrence

concat([],l2)=l2

concat(e1.l1',l2)=e1.concat(l1',l2)

Caml

```
let rec concat l1 l2 = match l1 with
    [] -> l2
    |(x : :xs ) -> (x : :concat xs l2);
```

Preuve de l'associativité de la concaténation

P(l1)=∀ l2, L3, l1@(l2@l3)=(l1@l2)@l3

Montrer ∀ l1, P(l1)

l1=[] :

[]@(l2@l3)=(l2@l3)=l2@l3 (1)

([]@l2)@l3=(l2)@l3=l2@l3 (2)

De (1) et (2) on déduit que []@(l2@l3)=([]@l2)@l3 (3)

x : :l1 :

On suppose P(l1)

(x : :l1)@(l2@l3)=x : :((l1)@(l2@l3))=x : :(l1@(l2@l3)) (4)

((x : :l1)@l2)@l3=x : :((l1)@l2)@l3)=x : :((l1@l2)@l3) (5)

De (4), (5) et P(l1) on déduit que (x : :l1)@(l2@l3)=((x : :l1)@l2)@l3 (6)

Par récurrence on déduit de (3) et (5) ∀ l2, L3, P(l1)

Exercice 4: Inversion

Décrire une fonction qui à une liste de caractères associe l'inverse de cette liste.

Spécification

inv : une liste de caractères → une liste de caractères

Relations de récurrence

inv([]=[]

inv(e1.l1)=inv(l1)oe1

Version 1

Caml

```
let rec inv = function
    [] -> []
    | (e : :l) -> inv l @ [e];
```

Version 2

Caml

```
let rec invaux acc liste = match liste with
    [] -> acc
    | (e : :l) -> invaux (e : :acc) l;
let rev liste =
    invaux liste [];
```

Passage de la version 1 à la version 2

P(l)=∀ a, invaux(l,a)=inv(l)@a

Montrer ∀ l, P(l)

l=[] :

invaux(a,[])=a et inv([])@a=a

donc invaux([],a)=inv([].a) (1)

e@l :

On suppose P(l)

invaux(e@l,a)=invaux(e@a,l)

et (inv(l)@e)@a=inv(l)@(e@a)

or ∀ a, invaux(l,a)=inv(l)@a

donc invaux(e@l,a)=inv(l)@(e@a) (2)

Par récurrence on déduite de (1) et (2) P(l)=∀ a, invaux(l,a)=inv(l)@a

Autres exercices possibles

- Egalité de deux listes
- Une liste ordonnée
- Extraction du n ième élément
- Appartenance à une liste
- Palindrome
- Interclassement de deux listes triées
- Anagramme