corrige

October 2, 2024

1 Sujet 1 (02/10/2024)

1.1 Exercice 1

[1]:
```
let f x y l = (x+.y)::l
```

[1]:
```
val f : float -> float -> float list -> float list = <fun>
```

[2]:
```
let g x = x x
```

```
File "[2]", line 1, characters 12-13:
1 | let g x = x x
                ^
Error: This expression has type 'a -> 'b
       but an expression was expected of type 'a
       The type variable 'a occurs inside 'a -> 'b
```

[3]:
```
let h x y =
    let hp x = (x+.y)>6. in
    let hpp y = x::y in
    hpp [hp 6.]
```

[3]:
```
val h : bool -> float -> bool list = <fun>
```

[4]:
```
let p =
    let pp x y = x@y in
    pp [1.;2.]
```

[4]:
```
val p : float list -> float list = <fun>
```

Exercice 2

1. - taille$(F) = 1$ - taille$(N(a,b))$ = taille(a) + taille(b) + 1

2. - hauteur$(F) = 1$ - hauteur$(N(a,b)) = \max($hauteur$(a),$ hauteur$(b)) + 1$

3. - ASB à 1 feuille = 1 - ASB à 2 feuille = 2 - ASB à 3 feuille = 5

4. Récurrence forte

C_0 : arbre à 1 feuille = feuille = 1

Soit la propriété vrai pour tout arbre à $n+1$ feuilles, on souhaite montrer qu'il y a C_{n+1} arbre à $n+2$ feuilles. Pour cela On considere la racine d'une arbre strictement binaire. Ce noeud a exactement deux fils. Ainsi si on veut un arbre à $n+2$ feuilles, pour chaque arbre à k feuilles que l'on peut mettre dans le fils gauche, on peut mettre en fils gauche n'importe quel arbre de $n+2-k$ feuilles. Avec la principe de récurrence forte sur les C_k avec les k de 0 à n on retoruve bien l'équation attendu.

[8]:
```
type asb =
| F
| N of asb * asb
```

[8]:
```
type asb = F | N of asb * asb
```

[9]:
```
let rec parfait n = match n with
|0 -> F
|n -> let pn1 = parfait (n-1) in N(pn1,pn1)
```

[9]:
```
val parfait : int -> asb = <fun>
```

[10]:
```
let est_parfait asb =
    let rec aux asbp = match asbp with
    |F -> 0, true
    |N(a,b) -> let h1,c1 = aux a in
               let h2,c2 = aux b in
               (h1+1, c1 && c2 && h1=h2)
    in let _,res = aux asb in res
```

[10]:
```
val est_parfait : asb -> bool = <fun>
```

2 Sujet 2 (02/10/2024)

2.1 Exercice 1

1. S(S(S(S(Z))))

2. - eval(Z) = 0 - eval(S(e)) = 1+ eval(e)

3. - plus(Z,x) -> x - plus(S(x),y) -> S(plus(x,y)

4. Soit $x \in P'$ - $x = Z \to x \in P$ - $x = S(S(x)) \to x \in P \to S(x) \in P \to S(S(x)) \in P$

5. Les entiers pairs

6. Soit $y \in P'$: - plus(Z,y) = y - plus(S(S(x)),y) = S(plus(S(x),y) = S(S(plus(x,y))

[12]:
```
type peano =
|Z
|S of peano
```

[12]: `type peano = Z | S of peano`

[13]:
```
let rec est_pair p = match p with
|Z -> true
|S(S(x)) -> est_pair x
|_-> false
```

[13]: `val est_pair : peano -> bool = <fun>`

[14]:
```
let rec convert n = match n with
|0 -> Z
|n -> S(convert (n-1))
```

[14]: `val convert : int -> peano = <fun>`

[15]:
```
let rec somme l = match l with
|[] -> Z
|Z::t -> somme t
|S(x)::t -> S(somme (x::t))
```

[15]: `val somme : peano list -> peano = <fun>`

Exercice 2

[21]:
```
let rec assoc x l = match l with
|[] -> None
|(a,b)::t -> if a=x then Some(b)
             else assoc x t
```

[21]: `val assoc : 'a -> ('a * 'b) list -> 'b option = <fun>`

[22]:
```
let rec set x y l = match l with
|[] -> [(x,y)]
|(a,b)::t -> if x=a then (x,y)::t
             else (a,b)::(set x y t)
```

[22]: `val set : 'a -> 'b -> ('a * 'b) list -> ('a * 'b) list = <fun>`

[24]:
```
let maxi l =
    let rec aux l acc = match l with
    |[] -> acc
    |h::t -> if h>acc then aux t h else aux t acc
    in match l with
    |[] -> failwith "euh"
    |h::t -> aux t h
```

[24]:
```
val maxi : 'a list -> 'a = <fun>
```

[25]:
```
let count l n =
    let rec aux lp acc = match l with
    |[] -> acc
    |h::t -> if n=h then aux t (acc+1) else aux t acc
    in aux l 0
```

[25]:
```
val count : 'a list -> 'a -> int = <fun>
```

[27]:
```
let all_count l =
    let rec aux l acc = match l with
    |[] -> acc
    |h::t -> match assoc h acc with
        |None -> aux t (set h 1 acc)
        |Some(x) -> aux t (set h (x+1) acc)
    in aux l []
```

[27]:
```
val all_count : 'a list -> ('a * int) list = <fun>
```

[28]:
```
all_count [1;3;4;4;1;6]
```

[28]:
```
- : (int * int) list = [(1, 2); (3, 1); (4, 2); (6, 1)]
```

3 Sujet 3 (02/10/2024)

3.1 Exercice 1

1. - longueur([]) = 0 - longueur(h::t) = 1+longueur(t)

2. - concat([],l) = l - concat(h::t,l) = h::(concat(t,l))

3. TODO voir tortue

4. TODO voir tortue

[5]:
```
let append1 l1 l2 = l1@l2

let append2 l1 l2 =
    let rec aux l1p l2p = match l1p with
    |[] -> l2p
    |h::t -> aux t (h::l2p)
    in aux (List.rev l1) l2
```

[5]:
```
val append1 : 'a list -> 'a list -> 'a list = <fun>
```

[5]:
```
val append2 : 'a list -> 'a list -> 'a list = <fun>
```

[7]:
```
append2 [1;2] [3;4]
```

[7]:
```
- : int list = [1; 2; 3; 4]
```

[9]:
```
let rec flatten l = match l with
|[] -> []
|h::t -> h @ (flatten t)
```

[9]:
```
val flatten : 'a list list -> 'a list = <fun>
```

Exercice 2

[10]:
```
type 'a stream =
| Nil
| Cons of 'a * (unit-> 'a stream)
```

[10]:
```
type 'a stream = Nil | Cons of 'a * (unit -> 'a stream)
```

[11]:
```
let singleton x =
    let nil () = Nil in
    Cons(x,nil)
```

[11]:
```
val singleton : 'a -> 'a stream = <fun>
```

[12]:
```
let cons x s =
    let next () = s in
    Cons(x,next)
```

[12]:
```
val cons : 'a -> 'a stream -> 'a stream = <fun>
```

[13]:
```
let uncons s = match s with
|Nil -> None
|Cons(x,next) -> Some(x,(next ()))
```

[13]:
```
val uncons : 'a stream -> ('a * 'a stream) option = <fun>
```

[14]:
```
let rec stream_to_list s = match s with
| Nil -> []
| Cons(x,next) -> x::(stream_to_list (next ()))
```

[14]:
```
val stream_to_list : 'a stream -> 'a list = <fun>
```

[15]:
```
let rec list_to_stream l = match l with
|[] -> Nil
|h::t -> let next () = list_to_stream t in Cons(h,next)
```

[15]:
```
val list_to_stream : 'a list -> 'a stream = <fun>
```