UE - Programmation Fonctionnelle

TD 6. Les flux.

les flux sont 1 structure de données utilisée fréquement.

Application :

+ reconnaissance de langage.
+ traitement de signal
+ iterateurs
+ simulation de systèmes dynamique.
+ elements de flux calculés à "la volée" (faible mémoire conso)
+ découplation entre la production et la consomation value

A la base de langage de programmation dits :

+ réactifs (Lu)
+ synchrones (lustre, SCADE, signal, ~Simulink)

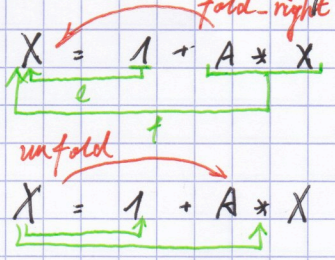
Spécification :

le liste et les flux sont "doubles" l'un de l'autre et correspondent à des solutions de

$$X = 1 + A * X$$

Nil | Cons of A * X

les flux sont potentiellement infinis.



fold_right f b e

in Java : next : unit → A option

in OCaml : next : X → A * X + 1

```
let rec fold f liste = match liste with
  | [] -> f None
  | hd :: tl -> f (Some (hd, fold f tl))
```

f : A * X → X
e : X

f : (fun (a, x) → g (Some (a, x)))
e : (g None)

(* flux infinis de 0 *)

let flux_nul = Flux.unfold (fun () -> Some (0, ())) () ;;

Exercice 1 :

+ let constant c = Flux.unfold (fun () -> Some (c, ())) () ;

+ let map f flux = Flux.apply (constant f) flux ;;

+ let map2 f flux1 flux2 = Flux.apply (map f flux1) flux2 ;;

Un flux n'est pas seulement une structure de données. il faut pouvoir calculer ses value lorsqu'on en a besoin seulment. calcule paresseux

Exercice 2 : Fibonacci 斐波那切

f_0	f_1	f_2	...		
+	f_1	f_2	f_3	...	
f_0	f_1	f_2	f_3	f_4	...

let tail f = match Flux.uncons f with

| None -> failwith "..."

| Some (_, tl) -> tl

let rec fibonacci = Tick.lazy (

(Flux.cons 0 (Flux.cons 1 (map2 (+) fibonacci (tail fibonacci)

))

))