LAB - 3

FUNCȚII CA VALORI de ordinul întâi

Pot fi:

→ valori ale unor variabile => sunt numite
→ argumente pt. alte funcții
→ întoarse de o funcție
→ membri ai unor structuri

CURRY / UNCURRY

Funcțiile pot fi:

① UNCURRIED => primește toți parametri

(define (plus x y) (+ x y))

↓ primește 2 param, toți odată

② CURRIED => primește param. pe rând

(define (plus x) (λ (y) (+ x y)))

↓ primește un param (x) și
întoarce o funcție care primește un param. (y) => param sunt pasați pe rând

VEZI DEMO !

TRANSFORMARE [curry -> uncurry]

((f 2) 3) ⇒ ((c -> u f) 2 3)

- forma curried
- param. pe rând

- nouă funcție, uncurried
- param toți odată

TRANSFORMARE [uncurry -> curry]

- (f 2 3) ⇒ (((u -> c f) 2) 3)

- forma uncurried
- param toți odată

- nouă funcție, curried
- param pe rând

FUNCȚIONALE

→ Funcțională = o funcție care primește ca parametru / întoarce o funcție

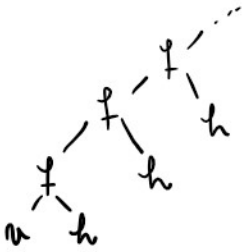
- map → rez: o listă

map f [1,2,3] => [f 1, f 2, f 3]
map f [1,2,3] [4,5,6] => [f 1 4, f 2 5, f 3 6]

- filter → rez: o listă
- foldl
- foldr

FOLDL (fold left)

```
foldl f v [] = v
foldl f v (h:t) = foldl f (f v h) t
```



→ folds the list up from left

Sintaxă

[foldl f v L_1 L_2 ...]

- f este o funcție cu
  - param 1 = (car L_1)
  - param 2 = (car L_2) ....
  - param n = acc (inițial = v)

=> λ x_1 x_2 -- acc -> ...

Ce vrem?

- o funcție 'my-map' care primește 2 param
  - f' = o funcție care preiau. 1 element
  - L = o listă

my-map (λ x -> x+1) [1,2,3,4] => [f' 1, f' 2, f' 3, f' 4] = [2,3,4,5]

(f' = funcție care primește un param și îl adună cu 1)

Execuție:

L	[1,2,3,4] -> [2,3,4] -> [3,4] -> [4] -> []
acc []	[2] -> [3,2] -> [4,3,2] -> [5,4,3,2] -> [5,4,3,2]
|| v	(f' 1) = 1+1 (f' 2) = 2+1
op	(f' h) : acc

↳ rez. întors de foldl => listă inversată la sfârșit

L	[1,2,3,4] → [2,3,4] → [3,4] → [4] → []
acc [] = v	[2] → [3,2] → [4,3,2] → [5,4,3,2]
	(f' 1) = 1+1 (f' 2) = 2+1
op	(f' h) : acc

↳ rez. întors de foldl
⇒ listă inversată la sfârșit

```
my-map f' L = reverse (foldl (λ x acc -> (f' x) : acc) [] L)
                              f                          v
```

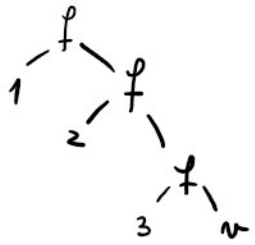
OBS: FOLDL, pt că este pe coadă, poate construi rezultatul inversat

FOLDR (fold right)

→ folds the list up from right

```
foldr f v []    = v
foldr f v (h:t) = f h (foldr f v t)
```

⇒ STIVĂ



↳ aplică f pe h și rezultatul apelului recursiv

• f este o funcție similară cu cea primită de foldl

L: [1,2,3,4] → [2,3,4] → [3,4] → [4] → []

[2,3,4,5] ← [3,4,5] ← [4,5] ← [5] ← [] → v

(f' 2) = 3 (f' 3) = 3+1 = 4 (f' 4) = 4+1 = 5

op. (f' h) : acc

```
my-map f' L = foldr (λx acc -> (f' x) : acc) [] L
```

EXERCIȚII

① rem-dup-left → data trecută → COADĂ ⇒ foldl

[1,2,3,2,4] ⇒ [1,2,3,4]

foldl f v L

f: (λ x acc → if x in acc
then acc
else x:acc)

v: []

! Rez. întors de foldl este o listă inversată ⇒

rem-dup-left = reverse (foldl (....) [] L)

② rem-dup-right ⇒ STIVĂ ⇒ foldr

• similar cu foldl, doar că rez. nu mai este inversat

③ overlay → foldl

Ce trb. făcut?

① punem 1 peste i (img. inițială) ⇒ o nouă imagine 1+i ("peste")
② punem 2 peste (1+i) ⇒ o nouă imagine (2+(1+i))
....

i [1,2,3] ⇒ ultima img. va fi deasupra

OBS: overlay { param 1 = img 1 ⇒ rez. este o nouă imagine, în care
param 2 = img 2 img 1 este peste img 2

• Ce vrem? ⇒ Să apelăm foldl cu param. corecți

[FORMĂ FOLDL:] foldl f v L ⇒ Cine sunt f și v?
↓ inițial img

f: (λ x acc → ....)
v: img. inițială

Execuție

L: [1,2,3] -> [2,3] -> [3] -> []

acc i : ov. ∧ i -> ov 2 ⓘ -> ....

f: (λ x acc -> overlay x acc) ⇝ overlay

h — img. curentă

inițial = v

=> foldl overlay inițial images

OBS: Când avem λ x y -> f x y, putem scrie direct f

(3) overlay => foldr

Ce frde. făcut?

① punem 3 peste i => o nouă imagine 3+i

② punem 2 peste (3+i) => — " — 2+(3+i)

....

[1,2,3] i => prima imagine va fi desenată

Ce vrem? => să apelăm foldr cu param corecți

[FORMĂ FOLDR]: foldr f v L

v — inițial

=> Cine sunt f și v?

f = λ x acc -> ...

v = inițial img

[1,2,3] -> [2,3] -> [3] -> []

ov h acc ← ov 3 i ← i

f: (λ x acc -> overlay x acc) ⇝ overlay

=> foldr overlay inițial images

④ mirror ((1,2,3), (4,5,6), (7,8,9)) => ((3,2,1), (6,5,4), (9,8,7))

map reverse matrix

⑤ slim-horizontal

((1 2 3) (4 5 6) (7 8 9)) → (1 2 3, 4 5 6, 7 8 9)

((overlay-> empty-image) lr), dar overlay-> este o funcție ↓ lista curentă

Ce vrem?

① să aplicăm funcția 'overlay->' pe fiecare linie => map

② funcția 'overlay->' este uncurry (primește 2 param odată); noi vrem să primească un singur param (linia curentă / lista curentă) => facem curry

=> ((uncurry-> curry overlay->) empty-image)

⑥ transpose

((1 2 3) (4 5 6) (7 8 9)) => ((1 4 7) (2 5 8) (3 6 9))

Vezi demo2.rkt

OBS => când vrem ca parametrii unei funcții să fie elem. dintr-o listă => apply

ex: apply + '(1 2 3) => + 1 2 3

⑦, ⑧ => freestyle