

# Pregotiv Express - Prinzipiel freiset

## 1. Reduzere expresii lombodo

$$(\lambda y. \lambda x. (\underline{y} \; y) \; (x \; x))$$

lombodo y lombodo x aplicat peste y aplicat peste x aplicat peste x.

căti ...? căti beta-redus?  $\rightarrow$  de unde că avem  $x \Rightarrow$  redus primul caștig  $\lambda y$  la care nu i se dă

dobândită o nouă formă expresie singurul beta-redus

Corp:  $\lambda x. (\underline{y} \; y)$  C e deosebit deosebit primul pct de corp

Prin lombod: y Prin lombod e primul  $\lambda$

Prin actual: x aplicat pe x  $(x \; x)$  Prin set e celălalt obiect  
++ ce se da -

introducere corp; deosebit deosebit & fără introducere corp pr y cu prin act.

$\hookrightarrow$  nu e o idee bună pt căt. cănd punem  $(x \; x)$  unde x e nu liber în

lomb lomb y, nu ajută să sub corpul unei let sau sa prin lombod x  $\Rightarrow$

$\Rightarrow$  x e nu liber în lomb lomb x de la  $\lambda x$

$\Downarrow$   
nu poate fi conv. unde îl redus. pe x:  $\lambda x. (\underline{y} \; y) \rightarrow \lambda z. (\underline{y} \; y)$

$\Rightarrow \lambda z. (\underline{y} \; y)$

$$[(x \; x)/y]$$

pe y împărtășește  $(x \; x)$

$\Rightarrow \lambda z. ((x \; x) \; (x \; x)) \rightarrow$  rezultat 1<sup>st</sup> folos.

$\downarrow$   
cum vrem să redesc? - NU.  $\Rightarrow$  suntem în

## 2. (define y 10)

$$\begin{cases} (\text{lombodo}(x)) \\ \quad \left[ \begin{cases} (\text{lombodo}(y)) \\ \quad \left[ \begin{cases} (+ \; (\geq x \; 2)) \\ \quad (+ \; y) \\ \quad (+ \; x \; y) \end{cases} \right] \end{cases} \right] \end{cases}$$

Jo nu îl suntem? nu, nu este î

Jo nu îl suntem nu? nu, probabil că expresia  
introduce părți închise funcționale

$(\text{lombodo}(y)) \text{if } (> x \; 2) \text{ then } (+ \; y) \text{ else } (+ \; x \; y)) \rightarrow$

x legat de S ( $x \in S$ ), e ceea ce suntem.

$(\text{lombodo}(y)) \text{if } (> S \; 2) \text{ then } (+ \; y) \text{ else } (+ \; x \; y)) \Rightarrow$

$\Rightarrow (\text{lombodo}(y)) \text{ if } (+ \; y) \text{ else }$

JK...  $\Rightarrow$  funcție nu este aplicată mai departe  
 $\Rightarrow$  nu suntem din nou să ne poată

3. 2 liste  $L_1$  &  $L_2$  & funcție  $f$  pentru care  $x \in L_1$  și  $L = \{y \in L_2 \mid y \in f(x)\}$  fără nec  
 $(x, L) \xrightarrow{\quad} \text{cifct}_f = \underbrace{\quad}_{\text{exp.}}$

map și filter

map (lambda(x))  $\leadsto \lambda x \text{ (filter (lambda(y) (zero? (remainder (x y)))) L))}$

Unor să producă liste de perechi unde  $x \in L_1 \Rightarrow$  perechi cu el din  $L_1 \Rightarrow$

$\Rightarrow$  map pe lista  $L_1$  cu lambda (x) unde producă perechi în  $L$ ,  $L$  este  
 lista tuturor elem din  $L_2 \Rightarrow$  filter

4. Aceea că o funcție este un lambda care returnează o listă și dacă doar  $L$  nu este că

dacă să luăm un apel nu să luăm ca fiind că  $L$  și doar să primim el din  $L \neq f \Rightarrow$  lucru

în doar rezultatul  $f$

$\Downarrow$   
 rezolvăm ca un fel de filter sau un fel de map.

(let (if odd.)

$(g ((\text{even } +) 1)))$  Este o adunare apoi o verificare a numărului impar.

$(\text{filter} (\text{map } g L)))$   $\Downarrow$  făcând map cu g și apoi filter după

)  $\Downarrow$  sau în Haskell:  $a = \text{map } g . \text{filter } f$

5. tipul unei funcții

$f : x, y \rightarrow (\text{head } x, \text{head } y) : f (\text{tail } x) (\text{tail } y)$

$\Rightarrow$  funcție  $f(x, y)$ ;  $x, y$  sunt liste

$f = \lambda x, y \rightarrow (\text{head } x, \text{head } y)$

este lambda de  $x, y$  care întărește  $\lambda$  și este expresia 1)

Expresia 2 este  $(\text{head } x, \text{head } y)$

:  $\equiv$  Expresia 3

$f (\text{tail } x) (\text{tail } y) = \text{Expresia 4}$  (făcând pe ...)

Prin expresia 1) tipul: dacă  $x$  este de tipul  $a$  și  $y$  este de tipul  $b$  și  $\text{Expresia 3}$  este de tipul  $c$

$\Rightarrow f$  este de tipul  $a \rightarrow b \rightarrow c$  și  $x :: a, y :: b \Rightarrow f :: a \rightarrow b \rightarrow c$

(1): Head  $x \Rightarrow x$  list  $\Rightarrow x :: [d]$

$y :: [e]$

newbie

(2)  $:: (d, e)$

$f :: d \rightarrow e$  elem, non per liste

(4) let liste del tipul  $d \Delta e \Rightarrow$  dim expresii  $y \Rightarrow f :: [d] \rightarrow [e] \rightarrow c$

(4)  $:: c$

(3)  $\Rightarrow$  cons  $\Rightarrow$  s dots nu si  $f$  sa fie o lista de cu ce sa potrebuie for cons  
Daca nu (lumea din lista)  $f$  sa fie tipul expresiei 2 de unde rezulta  $\Rightarrow$

$\Rightarrow (3) : c = [(d, e)]$

fie pos ca pos,

$\Rightarrow f :: [d] \rightarrow [e] \rightarrow [(d, e)]$

importa in expresie

primite  $x, y, f$  sa fie paranteze de  $x, y$  & const in  $f$  de  $x$  si  $y$   
 $\Rightarrow$  semnificat in Haskell sau dupa care este posibil

6. Closa  $E_2$  pt foliu Haskell sau ianuare sau num

$\Rightarrow$  fol Haskell sau ianuare sau num & int cau orice:

$f :: a \rightarrow b$  cau cau coresp for numeric  $\Rightarrow f :: (\text{Num} a) \Rightarrow a \rightarrow b$   
este de tipul  $a$  nu  $b$

intuitiv in  $E_2$ :

instante  $E_2$  de tipul de instantiat  $\Rightarrow f$ , nu cau numere  $\Rightarrow$  nu

nu cau in contextul  
fie  $f$  nu poate fi foliat in  $f$  pentru cau  
nu cau in contextul  
de numere  $\Rightarrow$

instance ( $\text{Num} a, E_2 b \Rightarrow E_2 (a \rightarrow b)$ ) where

; f este del fol pe = f este difert

$(==) f g = \text{foldl} (\lambda) \text{True} \$ \text{map} (\lambda x \rightarrow f x == g x) [1..10]$

egal explicit pe ? f? f? f? ; cele ? f? sunt egale doar in  
daca sunt de tipul  $a$

$a \rightarrow b$ : Haskell cau

sunt diferenca unu m  $\in K$ , m  $\in \{1, 10\}$ ;

$\Rightarrow$  f este  $E_2 m = \text{True} [1..10]$

( $\lambda$  f  $\Rightarrow$  f este m, si f este m  $\Rightarrow$  f este m)

cau el. sunt date cau  $\Rightarrow$

$\Rightarrow$  f este s-a vina dimenziunea

map cau lista ca baza  $\Rightarrow$  fold ca oca True

$\Downarrow$   $E_2 m$  in loc de  $\text{Num} a$ .

cau cau = oca f este m  $\Rightarrow$  f este m.

7. de fct à fct, Haskell car premier est list de caract. & int flottant

Similaire fonction dans celle avec tuple. ou si on a le set =>

=> pows "abc" => (a, b, c, ab, ba, ac, ... ) -> pows entière.

pows (char =

powers fibo int. ou flott ( > listas)

com const flottant in Haskell = 2 pps ex: fibo ou set des a, b & c, appti occupe

a b c

a a a, a a b a a c b a b b b c a c b c c

a a a a, a a c a b, a a a c, a b a, a b b, a b c ↑

pe deuxième niveau de récursion pex a, pex b & pex c le reste d'après ce que j'avais peur

de suivre l'ordre => const flottant pe toute lni en segi

des const flottant pe toute lni en segi => fibo ou set des flottants

=> similaire dans char; alors fibo int. ou listes de string, ou all normal =>

=> fonction const de insertion n'importe où dans string.

=> pows chars = map (:[]) chars ++

com si on fait des symboles dans chars / si on peut faire la fonction si on décompose

=> list comprehension; faire si on a des chars, appti faire string des

pows chars = map (:[]) chars ++ [c:s | s <- pows chars, c <- chars]

ou alors pows chars = flott where

flott = map (:[]) chars ++ [c:s | s <- flott, c <- s]

8. Avoir de toutes sortes de logique au produit de set

moi j'arrive pas à comprendre ça.

Il y a 3 pps: Com orfene, ou deonne

↑ deonne(Iou, bine) = Iou ou deonne bine

Iou ou a orfene bine

{=>

$\Rightarrow LPOI$ : Regeln und ob und  $I$ :

hmt:  $\neg \text{deeme}(I_{\text{in}}, b_{\text{in}})$

$\downarrow$

$\neg \text{deeme}(\text{pred})$  da vns  $\delta$  enne

$\text{deeme}(C_{\text{in}}, C_{\text{in}})$

mt gewen ob si' dl systeme

systeme(C<sub>in</sub>, C<sub>in</sub>) (außer C<sub>in</sub>)  $\Rightarrow$  Orte C<sub>in</sub> . Orte C<sub>in</sub>;

H<sup>mt</sup> für  $\delta$  &  $\delta$  an  $\delta$  enne, das ein system Es. med. enne  $\Rightarrow$  cine deeme in med. enne

$\Rightarrow$  orte C<sub>in</sub> . orte C<sub>in</sub> . system(C<sub>in</sub>, C<sub>in</sub>)  $\Rightarrow$  deeme(C<sub>in</sub>)  $\Rightarrow$

$\Rightarrow$  orte Personen . orte Med . system(Personen, Med)  $\Rightarrow$  deeme(Personen, Med)

? mt system(I<sub>in</sub>, b<sub>in</sub>)

deem prim. set

slug from R.A.

transform & formel abstrakt

Cloute

(1) an Simplifizie:  $\neg \text{steme}(\text{Personen}, \text{Med}) \vee \text{deeme}(\text{Personen}, \text{Med})$

(2) an premis:  $\neg \text{deeme}(I_{\text{in}}, b_{\text{in}})$  Si' fo' negativ in to cloute

(3) cond. neg:  $\text{steme}(I_{\text{in}}, b_{\text{in}})$

(1)  $\rightarrow$  (1):  $\neg \text{steme}(\text{Personen}, \text{Med}) \vee \text{deeme}(\text{Personen}, \text{Med})$

steme(I<sub>in</sub>, b<sub>in</sub>)

- - - - - - - - - - Personen I<sub>in</sub>, Med  $\leftarrow$  b<sub>in</sub>

Si' not sub substit. w

steme(I<sub>in</sub>, b<sub>in</sub>) mitte w acht subtit  $\Rightarrow$  cui? literali wörde  $\Rightarrow$

$\Rightarrow$  constante & deeme(Personen, Med) wobei option subtit  $\Rightarrow$

$\Rightarrow$  deeme(I<sub>in</sub>, b<sub>in</sub>) (u)

negat. en  
(u) + (2) deeme(I<sub>in</sub>, b<sub>in</sub>)

$\neg \text{deeme}(I_{\text{in}}, b_{\text{in}})$

- - - - - - - - -

cloute við

Q. Construiește Prolog care predă unul din cele mai scurte căi de listă cu cel mai multe elemente.

Să opereze este practic ca să sublîngărește din el liniile.

up(L, Up) :-

meniu co primul și să fie celeste;

up([E | RestL], [E | RestUp]) :-

pe restul lui co să pună el întrup deci doar fără) fără să > el. asta  $\Rightarrow$  predică  $\Rightarrow$  judecăt.

cine să înceapă listele și cu menirea

up([E | RestL], Up) :-

aux(RestL, RestUp)

Up = [E | RestUp].

& un meniu și în ultimul elem.

-adugăm L Up (dă fi făcător

noi pe cînd se adună)

$\Rightarrow$  aux începe la cel mai mare

și de la urmă

Care lucru: fără el e mai mare

up(L, Up) :-

L = [E | RestL],

aux(RestL, E, RestUp),

Up = [E | RestUp].

aux([ ], \_, \_).  $\rightarrow$  tot de bază

aux(L, Max, Up) :-

L = [E | RestL],

E > Max,

aux(RestL, E, RestUp),

Up = [E | RestUp].

cel de sub este să pună ceva în loc să

aux(L, Max, Up) :-

L = [E | RestL],

E <= Max,

aux(RestL, Max, RestUp),

Up = RestUp.

Q. Problema: - de să se scrie în Prolog  
Solutia: se scrie.

# Examen 2022 - variabili

1. Reduceti expresia  $\lambda$ :

$$(\lambda x. (\lambda x. (x \ x) \ \lambda x. x) \ (x \ x)) = (\lambda x. \lambda x. x \ (x \ x)) \ \ominus$$

Scurt 2 β redagi:  $\Rightarrow$  Îl elimin pe cel din interior:

$$(\lambda x. (x \ x) \ \lambda x. x) = (x \ x)_{\lambda x. x / x} = (\lambda x. x \ \lambda x. x) =$$

poate astă =  $\lambda x. x$  = funcție identitate

poate funcția  $x$  (de la primul  $\lambda x$ )

cupr.  $(x \ x)$

$\ominus$  doar  
o sa

trebuie redenumesc:  $(\lambda z. \lambda x. x \ (z \ z)) = (z \ z) \ ? \ ? \ \ominus$

poate astă = 2

poate funcț.  $(z \ z)$

cupr.  $\lambda x. x \rightarrow$  identity

2. (definie 10)

(

(lambda (x))

$y \Rightarrow \lambda x$  astăptă ceea ce îl

(if ( $> x$  2))

primăstă pe  $s \Rightarrow x = s \Rightarrow$

(lambda (y))

$\Rightarrow (\text{if } (> s \ 2)) \Rightarrow$  J-are deoarece primul  
cuadrat este 4 și următorul

(+y 1)

lambda (y)

(+y 1) doar și  $\lambda y$  astăptă ceea ce

(+x y)

) și nu primește nicio  $=$

)

$s$

= într-o cale altă.  $\frac{(+y 1)}{}$

5)

(fx y)

= într-o cale altă.  $\frac{(+y 1)}{\downarrow}$

)

$s$

... să fie preluat  $\downarrow$  nu reac. niciunul.

3.  $L_1, L_2$  zijn const o lists die passen ( $x \in L$ ) en  $x \in L \wedge L \in L_2$ .  $L$  oordt  $x$ .

(define const  $L_1 L_2$ )

(map

(lambda ( $x$ )

(cons  $x$  (filter

(lambda ( $y$ ) (zero? (remainder  $y x$ ))))

$L_2$ )

))

$L_1$

)

a) Geef de code!

(define b

NU. een filter in functie van

reste elem pos -> result g !!

(lambda ( $f g L$ )

(if (null?  $L$ )

'()

(append

(if ( $f (f (g (cons L)))$

(list ( $g (cons L)$ )))

'())

)

(b)  $f g (cons L)$

)  
)  
)  
)  
)

een filter die  $f \circ g (x)$  vindt waarbij  $x$  eerste  
nog niet =  
een elke elem in lijst, dan appliceer  $g$  op de

o filter



(map  $g (\text{filter} (\lambda (x) (f (g x)))) L$ )

( $f (g x)) L$ )

|

(define (const  $f g L$ )

(filter  $f (\map g L))$ )

de gebundelde principes

## S. tipore

$f x y = \text{if } (x == \text{head } y) \text{ then } (x) \text{ else } f x (\text{tail } y)$   
 $\rightarrow$  seen before

could e  $x = \text{heady} \Rightarrow y = \text{listo}$

$\Rightarrow$   $\rightarrow$   $\text{add or tie?}$  so depends on the found Eq?

$\left\{ \begin{array}{l} x :: a \\ y :: [b] \text{ der head } y = x \Rightarrow b = a \Rightarrow y :: [a] \\ \text{int } [x] \Rightarrow [a] \end{array} \right.$ 

 $\hookrightarrow$   $a$  und  $b$   
 ist absoluter Riff

$$f: E_{\mathcal{Q}_0} \Rightarrow a \rightarrow [a] \rightarrow [a]$$

Tipuri, interfețe, rezervante, predicofe.

Haskell  
+ type inference

fridge ←

Fernan Boket, Ph.D.

8. Cine invoca, no vede se forse, meno detto, forse)

I'm new here

Trustee (Custodian) → trustee (Fiduciary Creditor)

Rejestracja gospodarstw  
oszczędności.

Acerca de los  
frenos de  
señal

En mots (Personnes) → noms de (Personnes yed)

मुख्यो (Cine), त्रिवेदी (Cine, Forum)

morede (iou, foow)

Friends (You)  $\rightarrow$  ? need decision, forms  $\textcircled{d}$

## Tinasto (Bn)

## 8. Logica și predicatori de ordinul I

- Acei 3 propoziții: (1) Cine îmveță, nu mearde forme  
(2) mearde de (Ion, forme)  
(3) Ion nu îmveță

$\Rightarrow$  harta este mearde de (Ion, forme)

Inducere:  
(1) învăță (Cine), mearde (Cine, Cine), ceeașă cine  
Orice o fi cine doar învăță cel dinca care țină de Orice Cine.  
Orice Personă doar învăță atunci Personă nu mearde de Orice  
Iată  
Dacă nu e cine să fie Măd, ci Măd = forme  $\Rightarrow$

$\Rightarrow$  învăță (Cine), mearde (Cine, forme), ceeașă cine (decă: Orice o  
fi și cine, doar învăță, nu mearde forme); doar cine = persoană

$\Rightarrow$  Orice o fi persoană (aceea), doar învăță persoană ținând  
de forme.

Demonstrație prin reducere la absurd (Preupuțirea prin reducere la  
absurd ca în învăță):

(1) implicativ: învăță (persoană)  $\rightarrow$  mearde (persoană, forme)

(2) premisa: mearde de (ion, forme)

(3) condiție nevoie: învăță (ion)

loc ușor după următoarele

(1) și (3): învăță (persoană)  $\vee$  țină de (persoană, forme)

învăță (ion)

-----

persoană  $\leftarrow$  ion

țină de (ion, forme) (4)

(2) și (4): mearde de (ion, forme)

țină de (ion, forme)

----- doar vîză  $\Rightarrow$  presupunere folos  
în învăță.

## 20lg A

$$1. E = (\lambda x. (x (\lambda y. z x)) \lambda x.x) = (\lambda x. (x z) \lambda x.x) \underset{\equiv}{\circ}$$

seuă și B redescă. Încearcă să încercăm să împărțim:

$$(\lambda y. z x) = z_{[x/y]} = z$$

corp: z

parametru: x

param format: y

$$\underset{\equiv}{\circ} \alpha = (\lambda x. (x z) \lambda y.y) \underset{\equiv}{\circ}$$

corp: (x z)

param actual: λy.y

param format: x

$$\underset{\equiv}{\circ} (x z)_{[\lambda y.y/x]} = (\lambda y.y z) \underset{\equiv}{\circ} z$$

2. a) de către se procedează.

b) prima valoare după prima <sup>de</sup> apelare este cea din mult?

c) în cadrul funcțiilor înlocuile de proceduri

(define computation (delay (+ s s)))

(\* s s)

(define (f x) (cons x (force computation)))

(map f '(1 2 3 u)) = '('1.10), '2.10), '3.10), '4.10))

a) Voi face el din '1 2 3 u) și facem cu rețea lui  
 e să stăte, pe care o pun de la 0      (force(delay(+ s s)))  
 10.      //

dacă nu e niciun (la prima valoare din comput.)

b) după înmulțire, sunt funcții ale listei state, (la prima valoare  
 căduse și)

c) ~~(define computation (((+ s) s)))~~ → state-computație  $\Rightarrow$  și cum se combină  
 (define computation ( $\lambda()$  (+ s s)))      | includei funcțiile  
 (define (f x) (cons x ((computation))))

3. (define (even L1 L2)

(map

(lambda (x)

(cons x

(length

(filter

(lambda (y)

(equal? (list x) (list y)))

)

L2

)

)

L1

)

h.  $f x y = xy (y x)$

neut ist so f also def per f

$f x y : \text{neuturale } \lambda: f = \lambda x y \rightarrow xy (y x)$

$x :: a, y :: b; (y x) :: c \Rightarrow \boxed{x = c}$

$x \otimes y \Rightarrow \frac{x :: d \rightarrow e \quad y :: g \rightarrow h}{(x y) :: e}$

$\frac{\alpha :: b \quad \beta :: b}{x y :: a \rightarrow b}$   
 $\boxed{\alpha \otimes b = e}$

$y \otimes x: \frac{y :: i \rightarrow j \quad x :: k \rightarrow l}{(y x) :: j}$

$x y \otimes (f x): \frac{(x y) :: m \rightarrow n \quad (y x) :: o \rightarrow p}{(xy(f x)) :: m}$

$xy (y x) :: n \quad f :: a \rightarrow b \rightarrow n$

3 Aplicació de f(x) i en exponençial

$x \cdot y, y, yx^b$

resultat  $x \cdot y : x:a \rightarrow b \quad y : y:c$   $\Rightarrow \frac{x:a \rightarrow b}{y:c} \quad \underline{xy:b}$

$y:d \quad yx: y:e \rightarrow g \quad x:h \quad \Rightarrow yx:g$

$y:a \Rightarrow (a \rightarrow b \rightarrow g) \rightarrow b \quad x:a \quad \Rightarrow (y:(a \rightarrow b \rightarrow g) \rightarrow b)(x:a)$

$\Rightarrow x:a \rightarrow b \rightarrow c$

$y:a \text{ orden } \pi, (a \rightarrow b \rightarrow g) \rightarrow b$   $\downarrow$   $y \in \text{menys } g$   $\text{ que en } a$   
 $\Downarrow$   $x:y = x:y(yx) : x:a \rightarrow b \rightarrow c$ ,  
 $y:a$   
 $\Downarrow y:yx \in b$   
 $\Downarrow$   $x:y = x:y(yx) : x:a \rightarrow b \rightarrow c$

5 a)  $(\text{cor } (\text{append } [1, 2] \cdot [3, 4]))$

ia el [1, 2] si i posa do front, dep i 1-2 pos i  
ciu moi front m total, dep i  
...  $\Rightarrow$   $\text{append } [1, 2] \cdot [3, 4]$   $\Rightarrow$   $\text{append } [1, 2, 3, 4]$

b)  $\text{head } [1, 2] \cdot [3, 4]$

ia front d'la descom  $\Rightarrow$  pos i front pos pto s'ha x final. f

pos:  $\boxed{1, 2, 3, 4} \rightarrow \text{head}$

6 class MyClass & where

$f:a \rightarrow a \rightarrow \Theta$  posibl i inst o f: Ord,  $\Theta$  ord obstant a dep i int R.

Sou instance MyClass F] where

$f = \text{head}$

7 Nu tots els animals són mamífers  $\Rightarrow$

$\text{7.1. line, } \text{2bors } (\text{line}) \quad \Rightarrow \quad \text{2bors } (\text{line}), \text{7.2mamífers } (\text{line})$   
 $\text{señalones } (\text{line}), \text{duelor } (\text{line}) \quad \Downarrow \quad \text{2bors } (\text{animal}), \text{7.2mamífers } (\text{animal})$

Un animal és un animal en l'estat actual, segí  
mamífers  
(s'ha nomenat often, stat)

8.  $p(R, S) :- \text{member}(R, S) \rightarrow R \in S$

$\text{findall}(Y, \text{member}(Y, R), [Y_1 = x] \wedge \dots \wedge Y_n]) \leftarrow !, q(X, T, S).$

$q(X, A, [x | A]) \leftarrow q(x, [A(B)], [A(C)]) :- q(x, B, C)$

↳ ce se întâmplă dacă sunt două?

↳ prima oare, elimină după celelalte

de la deasupra

← contează pe cei care sunt membre

dacă  $x \neq Y_i$  și nu este

listă  
nu.

$p([1, 2, 3, 4], S)$

$S = [] ; S = [1, 2, 3, 4]$

$S = [2, 3, 1, 4]$

$S = [2, 3, 4, 1]$

↳ doar prima,

nu este consecutiv.

contează multe complicații

$\text{findall}(ce, fa, lista)$

$S$ , io primul portocală și crește în ordinea crescătoare,

$\Rightarrow q(X, A, [x | A]) \leftarrow q(X, [A(B), A(C)], [A(D)]) :- q(x, B, C), opri$

⇒ stă să reiau să verific  $\Rightarrow$

$q(X, T, S)$

playlist  $\rightarrow$  loc

$L : [1, 2, 3, 4] ; 2 : [1, 2, 3, 4] ; 3 : [2, 3, 1, 4], [2, 3, 4, 1]$

primul  $X$ ,  $A, [x | A]$ .

al doilea

$q(X, [A(B), A(C)], [A(D)])$ , opri dacă al doilea și următorul

element este  $opri$

9.  $r(Pointe, word)$

$\text{first}(X, F) :- \text{r}(Pointe, X), !, \text{findall}(Y, (\text{c}(Pointe, Y), Y \neq X), F)$

stop.

↳ și predici proto, contează faptul că  $X$  este un element

de  $F$  deoarece este un element

(Mop) MultiMap in Haskell:

a) data MultiMap K a = MM [(K, [a])] deriving (Eq, Show)

b) ins :: Eq K => K -> a -> MultiMap K a -> MultiMap K a

ins K a (MM lft) = MM {case back of

[] -> (K, [a]): front

(\_, as) : back -> (K, a : as): front ++ back

where

(front, back) = break ((== K), fst) lft

c) map' :: (a -> b) -> MultiMap K a -> MultiMap K b

map' f (MM lft) = MM {map' ((K, as) -> (K, map' f as)) lft  
do

precision-1 ... precision-n (new)

underf-1 ... underf-n

function:  $\frac{Var::a \quad Expr::b \text{ (T Lambda)}}{Var \rightarrow Expr::a \rightarrow b}$  Variables ... expression

application:  $\frac{Expr1::a \rightarrow b \quad Expr2::a}{(Expr1 \ Expr2)::b} \text{ (TAff)} \quad Expr1 \dots Expr2$

$+:$   $\frac{Expr1::Int \ Expr2::Int}{Expr1 + Expr2 :: Int} \text{ (T+)} \quad Expr:: \dots Even valid tip$

$f \ g = f(3) + 1:$   $\frac{g::a \quad (g 3) + ::b}{f::a \rightarrow b} \text{ (T Lambda)}$  let ... even cur. fct,

$\frac{(g 3)::Int \quad 1::Int}{(g 3) + 1::Int} \text{ (T+)} \Rightarrow b = Int$

$\frac{g::c \rightarrow d \quad 3::c}{(g 3)::d} \text{ (TAff)} \Rightarrow c = C \rightarrow d, c = Int, d = Int$

$\Rightarrow f::(Int \rightarrow Int) \rightarrow Int$

$f \ x \ y = x \ y \ (y \ x)$

$x::a \quad f::a \rightarrow b \rightarrow d$

$y::b \quad y \in \text{function} \Rightarrow y::e \rightarrow g \equiv b$

$\Rightarrow x::a \quad \begin{cases} f::a \rightarrow (e \rightarrow g) \\ y::e \rightarrow g \end{cases}$

but y is applicable for x =>

$e \equiv i$

but since x & y => h = g

but since x is applicable for y =>  $x::h \rightarrow i \ni a$

$x::h \rightarrow i \rightarrow$

$f::h \rightarrow i \rightarrow \dots \Rightarrow$  no function cyclic

STOP  $\Rightarrow$  must  
proto tips

2019-B

$$1. E = (y (\lambda x. \lambda x. x (\lambda y. yy)))$$

are 3  $\beta$ -receptors. These can be modulated by drugs  $\Rightarrow$

$\Rightarrow$  fct identit opf ee y  $\Rightarrow$  (y  $\equiv$   $\lambda x. \lambda x. x$  y))

für alle  $\alpha$  gilt  $\Rightarrow (\lambda y. (\lambda z. \underline{\lambda x. x}) y) \Rightarrow (\lambda y. \lambda x. x)$  nach obige. Cqd.

neu em sportivem leid zu corp => scop de ()

2. (define computation ( $\lambda(x)(\text{equal? } x))$ ) → includes functions

(define (f x) (and (> x 5) (computation)))

(filter  $\neq^{-1}(13 \leq 79)$ )

a) discussione dei ruoli,

equal se opereaza astfel  $\frac{1}{10}$  din fiecare decimaleaza > 5 din lista

or ~~donegalteel~~ '(7 g)

(soft ~~new~~  
sent in list)

↳ s. foto 25, dec merge

↓ due sono presenti

3- se fi apelat numai & date, primii dots, nu stiu se pot scrie in memoria, deoarece se apelaaza de

functions in memory, it can't see the code

b) (define computation (delay (equal? x y)))

(define(f x) (mod(>x s) (force computation)))

cf schéma de moi seul de septembre à droite

$\exists$  (`define`(`new` `LL`))

(filter (combs (<))

( $\#$  ( $\leq (\text{apply} + \text{L}) (\text{apply} * \text{L})$ ))

三

# f

) ) )

4.  $f = \text{map} (\text{++})$

$\text{map } f : L \rightarrow L' \Rightarrow \text{map} :: (a \rightarrow b) \rightarrow [a] \rightarrow [b] \quad (1)$

$(++) : L_1 ++ L_2 \rightarrow L_3 \Rightarrow (++) :: ([c] \rightarrow [c] \rightarrow [c])$   
 $L_1 \quad L_2 \quad L_3 = L_1 \cup L_2 \quad (2)$

cosă urmărește (1) sau (2) trebuie să fie și grupat următoare

$\text{map} :: (a \rightarrow b) \rightarrow [a] \rightarrow [b]$

$\text{map} \Leftrightarrow (++) :: ([c] \rightarrow ([c] \rightarrow [c])) \rightarrow [c] = [c]$

$\downarrow$

$b = [c] \rightarrow [c]$

$\Rightarrow f :: [c] \rightarrow [c \rightarrow c]$

$\text{map} (\text{++}) \rightarrow \text{list} \times \text{append} \text{ de liste din liste} \Rightarrow \text{primeste}$

liste de liste și întoarcă tot o listă de liste.

5. a) (length (map add 1 [1..10]))  $\rightarrow$  de 10 ori

$\downarrow$  adună f este lista de liste și următoare

b) length \$ map (+1) [1..10]  $\Rightarrow$  face  $(10-1)+1=10$

$\hookrightarrow$  nu încădă în definiția lui map că este o listă

nu avem definiție!

6. Implementare în Haskell sp (+) și (\*) și valoare

noi suntem fermăsi, putem, nu vom să ne complicăm, vom să prezentăm simplu

$\Rightarrow$  să supărați, să săgăduiți, să săgăduiți;

pentru fel de valoare; Bool;

instance Num Bool where

$(+) = (\lambda)$

$(*) = (\lambda)$

7. „Nu suntem cănd vor să luăm”  $\rightarrow$  nu vom mulțumi cănd vor să luăm  $\Rightarrow$

ca să nu suntem cănd, cănd nu suntem  $\Rightarrow$  să luăm cănd

$\Downarrow$

când nu suntem cănd  $\Rightarrow$  nu suntem cănd

=> există un număr

când nu suntem cănd

când

$\Rightarrow$

$\exists x. \text{când}_x \text{suntem}(x) \Rightarrow \text{când}_x \text{suntem}(x) \Rightarrow$  se poate da implicație ( $P \Rightarrow Q \equiv \neg P \vee Q$ )

$\rightarrow \forall X. \exists \text{listii\_nr}(x) \wedge \text{list\_nr}(x)$

$\{ \exists \text{listii\_nr}(x) \} \rightarrow \text{clouz}_1$

$\{ \text{list\_nr}(x) \} \rightarrow \text{clouz}_2.$

8.  $p(-, [ ], [ ]).$

$\Rightarrow p(A, [A|B], B) :- !.$   $\rightarrow$  stop.  $p$  are copia sa al  $p$  din primul  
de la listei  $[A|B]$

$p(A, [S|C], [B|D]) :- p(A, C, D).$

$B =$  restul listei.

$B$  e sublistă,  $A$  e elem de divizor  $\rightarrow (A, C, D)$

$[B|C] \rightarrow$  lista care înseamnă că  $B$  și  $C$  sunt un

gen listă, nu lista de liste, gen  $B|C$  conține pe  $C$

$\hookrightarrow$  fel să  $p$  în  $[B|D]$  devin cu 2 opere numere pe  $A$ .

Este predicatul select, iar doar primul argument este legat, fiind că pe  
primul elem, dăfă,  $x =$  elem din lista  $y$ ,  $\exists$  este  $y \setminus x$ .

9.  $\mathcal{E}(\text{Pointe}, \text{Spot}) -$

suntem lăsați, văd că stă simple, fără să ne ramăne funcții auxiliare,  
nu putem să aducem:

$\text{uni}(X, Y) :- \text{findoll}(Y,$   
 $\text{Pointe}, X), \mathcal{E}(\text{Bunie}, \text{Pointe}), \mathcal{E}(\text{Bunie},$   
 $\text{Unelii}, (\text{Unelii} = \text{Pointe}), \mathcal{E}(\text{Unelii}, Y)), V_1, \text{sort}$   
 $(V_1, V),$

10. HashSet ca bucket-set

class Hashable a where hash :: a  $\rightarrow$  Int

instance Hashable Int where hash x = mod x 21

data HashSet a = HS [(Int, [a])] deriving (Show, Eq)

ins :: Int  $\rightarrow$  HashSet Int  $\rightarrow$  HashSet Int

ins a (HS ps) = HS \$ case back of

$[] \rightarrow (K, [a])$ : front

$(-, as) : back \rightarrow (K, a : as)$ : front :: back

where

$K = \text{Hash } a$

$(\text{front}, \text{back}) = \text{break}((== K), \text{fst}) \text{ fst}$

$\text{map} :: (\text{Hashable } a, \text{Hashable } b) \Rightarrow (a \rightarrow b) \rightarrow \text{HashSet } a \rightarrow \text{HashSet } b$

$\text{map } f (\text{HS } \text{fst}) = \text{HS } \$ \text{ map } (\lambda (K, as) \rightarrow (K, \text{map } f as)) \text{ fst}$



5. instance  $\text{Show}(\text{Env}, \text{Next}, \text{ShowB}) \Rightarrow \text{Show}(a \dashv b)$

$\text{Show } f = "14" \text{ mod } f [1..10]$  (where  $f \in \mathbb{N}$ )

--  $f$  is split into  $\text{env}, \text{r}, \text{f} \dashv \text{b}$  &  $\text{Show}$  processes.

of  $\text{r} \dashv \text{b}$ .

$\text{Show } f = \text{concat}(\text{Show } f) [1..10]$  of  $f$ .

Same here given even so we can show.

if  $\text{of } g_i = \text{plus}$

$f \in I[1..10]$

is complete

so we see, if  $\text{f} = \text{concat} @ g_1 \dots g_n$

6. [map concat (apply (+) (zipWith [::] [::]))] ?

$\left[ \text{take } S \text{ } [m \mid m \leftarrow [m..]] \text{, med } m \text{ } m = 0 \right] \vdash m \leftarrow [1..]$   
 ↓ ↓ liste des      ↓ index von 0 bis n → multiplizieren  
 liste der Liste von S

7. "Il est un", "Il y a une bicyclette"  $\Rightarrow$  "Il y a des bicyclettes".  
 Savoir est bagot. savoir(x), des bicyclettes(x), bagot(x).  
 catégoriser x

Koisi se on põredina neli siinpileti:  $\exists x. \text{ser}(x)$

$\exists x.$  oneBicicle(x).

Once we are bicyclists, j) |

$$\exists x, h_{\text{soft}}(x)$$

$$\forall x. \text{oneBicycle}(x) \Rightarrow \text{oneBicycle}(x) \Rightarrow \exists x. \text{oneBicycle}(x) \vee \text{twoBicycles}(x) \Rightarrow$$

$\{ \text{an}(\text{in}) \} \quad (1) \quad \text{in este an } \equiv \text{an(in)}$

$$\{ \text{Tom}(x) \vee \text{reBicycle}(x) \} \Downarrow (2)$$

{720 bits of info} (3)

↳ 7 best of (iron) b(u)

(1)  $\Rightarrow$  one Biocidets (in) (5)

(5)  $w(3) \rightarrow \Gamma \downarrow \underline{STOP}$  11

- ↳  $\neg \text{oneBicycle} \vee \text{noBicycle}$
- ↳  $\neg \text{oneBicycle} \wedge \text{noBicycle}$
- ↳  $\neg \text{oneBicycle} \vee \text{noBicycle}$
- ↳  $\neg \text{oneBicycle} \wedge \text{noBicycle}$

8.  $\text{diff}(A, B, R) :- \text{findAll}(X, (\text{member}(X, A), \neg \text{member}(X, B)), R).$

so

$\text{intersect}(A, B, R) :- \text{findAll}(X, (\text{member}(X, A), \text{member}(X, B)), R).$

10. list of circular predicates where list signifies the one of cursor.

circular P = concat . repeat \$ P

get = head

next = tail.

so circular P = append . repeat \$ P

get = head . reverse

tail = tail . reverse

218-B

$$\begin{aligned}
 & 1. \lambda x. \lambda y ((\lambda x. \lambda y. y) (x y)) (y x) \rightarrow \lambda x. \lambda y. ((\lambda z. z) (x y)) (y x) \\
 & \quad \text{Diagram: A curved arrow from } (\lambda x. \lambda y. y) \text{ to } (\lambda z. z) \text{ with } x \text{ and } y \text{ swapped.} \\
 & \rightarrow \beta \lambda x. \lambda y (\lambda z. z (y x)) \xrightarrow{\beta} \lambda x. \lambda y (y x)
 \end{aligned}$$

2.  $(\text{left}((a_1)) \ a_1 \text{ used})$

$$(b \ 2) \quad b \equiv 2 \pmod{}$$

$$(c(+\alpha_2))) \quad r=1+2=3 \text{ (imbd)}$$

$$(+ a b c)) \quad 1+2+3=6.$$

## Reactive Centrizable Mediator

deposits on fast freeze

Se fortesc uol semper in a.

Lambda ( $\lambda b c$ )  
 $(\lambda b c) \rightarrow (\lambda c)$

osteoarthritis fct.

is one of round;

$$a=1, b=2; c=a+2, \text{ exec a}$$

euere.

3. write  $\lambda [a,b] \rightarrow ([a], [b])$ . can't functionale!

```
(define (reva L) ; prime inputs  
; prime!  
  ; (append (car L) (cons (cdr L))))  
  (cons (map car L) (map cdr L)))
```

$$4. f(x) = x \cdot y^2$$

Penetre intercept:

x::a ; y::b; z::c;

Mövit so und a const sunt:  $x$  to  $s$  für  $f_1$ ,  $y$  to  $f_2$   $\Rightarrow$   $z = \cos s$  für  $\cos p$   $\Rightarrow$

$$\Rightarrow x :: d \rightarrow e^{\alpha}; y :: f \rightarrow g \Rightarrow y :: c \rightarrow g \rightarrow h = b$$

do  $y ::= b$      $x \cdot y \neq x \circ y \Rightarrow x \circ y \neq x(yt(...))$

$z \vdash c$

$\cup_{x \in h^{-1}(e)}$

$$f: (h \rightarrow e) \rightarrow (c \rightarrow g \rightarrow u) \xrightarrow{\quad c \quad} g \rightarrow e$$

$g \rightarrow h \rightarrow k \rightarrow e$

5. Instance ( $\text{New } a, \text{ Show } b$ )  $\Rightarrow \text{Show}(a \sim b)$  where

$\text{show} = \text{const map } (\text{show. } f) [1..10]$

6.  $[\text{new } m \in [1..n], \text{ need } m \text{ nu} == 0 \mid m = [1..]]]$

7. „George este toron“, „Orice toron are o săpă“, „George este destept sau are o săpă“.  
 $\text{toron}(x), \text{areSapă}(x), \text{destept}(x)$ .

$\exists x. \text{toron}(x); \forall x. \text{areSapă}(x); \exists x. \text{destept}(x)$ .

$\text{toron}(\text{george}); \forall x. \text{toron}(x) \Rightarrow \text{areSapă}(x); \text{areSapă}(\text{george}); \text{destept}(\text{george})$

$$P \Rightarrow Q \equiv \neg P \vee Q$$

$\neg \text{george} \models \neg \text{areSapă}(\text{george})$

$\models \neg \text{destept}(\text{george})$

$\models \forall x. \neg \text{toron}(x) \vee \text{areSapă}(x)$

$\Downarrow$

$\models \exists x. \neg \text{toron}(x) \vee \text{areSapă}(x)$

{ $\text{toron}(\text{george})\}$  (1)}

{ $\models \neg \text{toron}(x), \text{areSapă}(x)\}$  (2)}

{ $\models \text{areSapă}(\text{george})\}$  (3)}

{ $\models \neg \text{destept}(\text{george})\}$  (4)}

(1) cu (2)  $\Rightarrow x \leftarrow \text{george} \Rightarrow \text{areSapă}(\text{george}) \Rightarrow \models \text{areSapă}(\text{george})\}$  (5)

(5) cu (3)  $\Rightarrow \models \neg \text{destept}(\text{george})$  STOP.

2016-A

1.  $(\lambda y. (\lambda x. \lambda y. x y) z) \rightarrow (\lambda y. (\lambda x. \lambda z. x y) z)$

I: de bo sh:  $(\lambda y. (\underline{\lambda x. \lambda z. x y}) z) \xrightarrow{\beta} (\lambda y. \underline{\lambda z. y z}) \rightarrow \xrightarrow{\beta} \lambda z. z$

ii. de bo st:  $(\underline{\lambda y. (\lambda x. \lambda z. x y)} z) \xrightarrow{\beta} (\lambda x. \underline{\lambda z. x z}) \xrightarrow{\beta} \lambda z. z$

2. (define (myAndMap L))

; fold ( $\lambda(x y) (\text{and } x y) \text{ if } L \text{ else lists}$ )

; (filter  $\#t$  (map and (map cons (map car L))))

; (filter (lambda(x) (if (x) #t #f)) (map and (map (cons (map car L)) L))))

; (filter and #f) ??

3. (let ((n 2)))

↳ (letrec ((f (lambda(n)  
; auto evaled (if (zero? n)  $\rightarrow$  resto e odd n.  
; in cons 1  $\rightarrow$  resto pin  $\lambda$   
 $\rightarrow$  factorial.  
; (n (f (- n 1)))  
; ))  $\rightarrow$  n div  $\lambda$

(#t s))  $1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120?$   
;  $\rightarrow$  resto e ng del  $\lambda$

(?) (define (calcul x y z) (#t x y z))  $\rightarrow$  "meidare fumtions"

(define (test variant) (calcul variant 2 (calcul complex 3)))

(define (test variant) (cond (variant1) (calcul variant1)))

— (else calcul-complex 3)))  $\rightarrow$  sp. cond at. cond  
E neu, gen cond ? fdr

(define (calcul x y z) (#t x y (force z)))

(define (test variant) (calcul variant 2 (delay(calcul-complex 3))))

$$5. f \times y = (y \times) x$$

$$x :: a, y :: b$$

$$(y \times) \Rightarrow y \text{ e fct} \Rightarrow y :: c \rightarrow d \equiv b$$

$y$  explicit per  $x$ , dan si  $y \times$  e fct

$$\begin{array}{ccc} (y \times) :: c \rightarrow d & (y \times) :: a \rightarrow d \\ \Downarrow & y \text{ is } \text{co}\text{-}\text{ong } x, x :: a & \Downarrow \\ y :: a \rightarrow a \rightarrow d & & (y \times) x :: d \end{array}$$

$$x :: a, y :: a \rightarrow a \rightarrow d; \text{nez} :: d \Rightarrow$$

$$\Rightarrow f :: a \rightarrow (\underbrace{a \rightarrow a \rightarrow d}_{\text{y}}) \rightarrow d$$

$$6. \text{instance}(E_2 \underline{a}, E_2 \underline{b}) \Rightarrow E_2 (a, b, c) \text{ where } \text{core elms}$$

$$(a_1, a_2, -) = (b_1, b_2, -) = (a_1 == b_1) \wedge (a_2 == b_2)$$

$$7. \text{set}\Delta :: [\alpha] \rightarrow [\alpha] \rightarrow [\alpha]$$

$$\text{set}\Delta a b = [x \mid x \in a, \text{not } x \in b]$$

8. "Oricemas" (si or negatif)

$$\forall x. \text{nos}(x); \nexists x. \text{oneNos}(x)$$

$$\forall x. \text{nos}(x) \Rightarrow \text{oneNos}(x) \Rightarrow \neg \forall x. \neg \text{nos}(x) \vee \text{oneNos}(x) \Rightarrow \exists x. \neg \text{nos}(x) \vee \text{oneNos}(x).$$

deci: negatif numbers per carno  $\Rightarrow$  real carno and nos.

$$\text{nos}(x, y); x \neq y; x \text{ nos terp } y; y \neq nos \text{ per } x \Rightarrow \text{oneNos}(y, x).$$

$$\text{deci } \forall x \forall y. \text{nos}(x, y); \forall z \forall t. \text{oneNos}(z, t).$$

$$\text{pt. 2nd nos terp: } \forall x \forall y. \text{nos}(x, y) \Rightarrow \exists z. \text{oneNos}(z, x).$$

$$9. \forall x. \text{TheBit}(x, \text{dimineato}) \Rightarrow \text{thy.ajungelo}(x, y), \text{test}(x, \text{dimineato}), \quad \begin{matrix} \text{1-jungelo} \\ (\text{eu, framen}) \end{matrix}$$

$$\text{elimin. aplikasi file} \Rightarrow \neg \forall x. \neg \text{TheBit}(x, \text{dimineato}) \vee \forall y. \text{ajungelo}(x, y)$$

$$\Rightarrow \exists x. \neg \text{TheBit}(x, \text{dimineato}) \vee \forall y. \text{ajungelo}(x, y)$$

$$\text{deci 2: } \neg \text{TheBit}(x, \text{dimineato}), \text{ajungelo}(x, y) \quad (1)$$

$$\neg \text{TheBit}(x, \text{dimineato}) \quad (2).$$

$$\neg \text{ajungelo}(x, \text{framen}) \quad (?)$$

neg conclusion  
dari:

$$\neg \text{ajungelo}(x, \text{framen})$$

(1)  $\wedge$  (2):  $x \in \text{ex}_i$ ,  $\&$  eliminare clusteri  $\Rightarrow \{\text{giving}(x_i, y)\}$  (4)

(1)  $\wedge$  (3):  $x \in \text{ex}_i$ ,  $\&$  eliminare clusteri = {L}

$y \in \text{ex}_{\text{new}}$

(3)  $\wedge$  (4):  $y \in \text{ex}_{\text{new}}$ ,  $\&$  eliminare clusteri  $\Rightarrow \Gamma \vdash \text{STOP}$

10.  $\rho(A, [ ], A)$ .  $\rho(A, [E|T], [E|R]) :- \rho(A, T, R)$ .

Face  $\rho$  repeat over  $R$  in pred / core & effects split. Dacă  $\rho$  este L1 & L2

$\rho(L_1, L_2, R)$ ,

$R$  = rezultat.

A nu mult vîță de  $A$ .

$\rho(A, [ ], A)$ .

$\rho(A, [E|T] [E|R]) :- \rho(A, T, R)$ .

$T$  &  $R$  sunt liste corecție cu  $E$  și nu sunt pe  $A$ .

dacă  $A$  nu e repeat de mult, îl luăm la primul pas.

$L_2 \& R$  fac un singur element și nu conțin pe  $L_1$

$\rho(A, [E|T])$  nu e net  $\rho(E|R)$

$\Downarrow$  deci schimbăm în  $T$  pe  $A$ , să ne ibinezi, schimbăm în

$[E|R]$  pe  $A$ , la final.  $\Rightarrow L_2 + L_1$ .

11. 1)  $f \# t \leq (1/2 \circ) \rightarrow f$  nu poate fi  $S \rightarrow$  nu dă eroare;  $\rho$  nu este inițială

2)  $(\text{let } ((f (\lambda (x y) x)) (f \leq (1/2 \circ)))$

(let  $((f (\lambda (x y)$

)  
))  
 $x)$        $f$  este  $(\lambda (x y)$

$(f \leq (1/2 \circ))$   
)  
 $x)$       ↓  
                show,  
                 $\lambda$ -step  
                un al doilea pas.

3) let  $f x y = x \# y \leq S(\text{div} 2 \circ)$

rek;  $x = 5, y = (\text{div} 2 \circ)$

4)  $x = 2/0, y = x \rightarrow$  nu este corect  $\& \Rightarrow$  nu este eroare,

$x \neq 1/0, y \neq x, x = y \Rightarrow$  nu este eroare  
= new and, both  
=) or both

16 - b

$$1. (\lambda x. \lambda y. \lambda z. y \ x) \ s \rightarrow (\lambda x. (\lambda y. \lambda z. y \ x) \ s)$$

Sunt 2 β redensi.

mai întâi de la st la st:  $\lambda y. \lambda z. y \ x \rightarrow_{\beta} \lambda z. x$

deci:  $(\lambda x. (\lambda y. \lambda z. y \ x) \ s) \rightarrow_{\beta} (\lambda x. \lambda z. y \ s) \rightarrow_{\beta} \lambda z. s$

dătoare să luăm:  $(\lambda x. (\lambda y. \lambda z. y \ x) \ s) \rightarrow_{\beta} (\lambda y. \lambda z. y \ s) \rightarrow_{\beta} \lambda z. s$

2. (define (map or map L))

(foldr (lambda (x y)(or x y)) #L))

; (filter (map or (map list L L)))

; (apply (map or (map list L L))))

ASTA E ZIP-UL ADEVĂR

3. (letrec ((f (lambda (n)))

(let rec (n (- n 1))))

calc  $n * \dots * 1 * 0 * (-1)$  = 0.

(f f 2? n -1)

1

(\* (+ n 1) (f n))

in factored

f 5 => n = 5 => n = 4 => n ≠ -1 => (\* 5 \* 4)

\* 5 \* 4 \* 3 \* 2 \* 1 ≠ 1

f 4 => n = 4 => n = 3 => 3 ≠ -1 => (\* 4 \* 3)

\* 4 \* 3 \* 2 \* 1 ≠ 1

f 3 => n = 3 => n = 2 => 2 ≠ -1 => (\* 3 \* 2)

\* 3 \* 2 \* 1 ≠ 1

f 2 => n = 2 => n = 1 => 1 ≠ -1 => (\* 2 \* 1)

\* 2 \* 1 ≠ 1

f 1 => n = 1 => n = 0 => 0 ≠ -1 => (\* 1 \* 0)

\* 1 \* 0 ≠ 1

f 0 => n = 0 => n = -1 => 0 ≠ -1 => 1 ≠ 1

1 ≠ 1

4. (define (clerk x y z) (if x y z))

(define (test variant) (clerk variant2 (clerk-complex 3)))

Pregătirea se poate face astfel ca primitivii  $\Rightarrow$

dacă at cănd este nevoie, să nu fie să se întâmpănește primitivii

la noi varianti și fără folos.

(define (clerk x y z) (if x y (force z)))

(define (test variant) (clerk variant2 (delay (clerk-complex 3))))

5.  $f \times g = x(yx)$

gen:  $f:: a \rightarrow b \rightarrow c$ .  
       $\downarrow \quad \downarrow \quad \downarrow$   
       $x \quad y \quad \text{rest}$

$x$  este fapt ce se aplică  $(y \times x) \Rightarrow x:: d \rightarrow e$ .

$y$  este  $x \Rightarrow y$  primul este rest  $x \Rightarrow y:: f \rightarrow h$

$x$  este  $(y \times x) \Rightarrow x$  primul este  $y \Rightarrow x:: h \Rightarrow d = y$ .

$y$  este  $x \Rightarrow y$  primul este rest  $x \Rightarrow y:: e \rightarrow h \Rightarrow g = e$

$x:: h \rightarrow e ; y:: e \rightarrow h$

dec:  $f:: a \rightarrow b \rightarrow c$

$x:: d \rightarrow e$        $d = h$  ( $x$  este valoarea de la  $y$ )

$y:: g \rightarrow h$        $e = c$  ( $f$  este valoarea de la  $x$ )

$f:: \underbrace{(d \rightarrow c)}_{\text{do}} \rightarrow ((d \rightarrow c) \rightarrow d) \Rightarrow a = g = d \rightarrow e$  ( $y$  este valoarea de la  $x$ ).

$b = g \rightarrow h \equiv g \rightarrow d \Rightarrow b = (d \rightarrow c) \rightarrow d$

$\forall$  bisognă să dividă  $\Rightarrow \underline{d \rightarrow c}$ .

6. instance ( $\exists_2 a, \exists_2 b, \text{ord}(a, \text{ord}(b)) \Rightarrow \text{ord}(a, b, c)$  where

$(a_1, \sim, \sim) (\leq) (b_1, \sim, \sim) = (a_1 \leq b_1)$

7.  $\text{SetN} : [a] \rightarrow [\bar{o}] \rightarrow [a]$

$$\text{SetN } a \ b = [x \mid x \in a, x \in b] = [x \mid x \in a, \downarrow \text{elem} \in b]$$

8. "Orice copil are o monă."

nu avem o folozi de 2 ori "

$\text{copil}(x)$ , are  $\text{Mon}(x)$ .

$$\nexists x. \text{copil}(x) \wedge \nexists x. \text{areMon}(x)$$

$$\nexists x. \text{copil}(x) \Rightarrow \text{areMon}(x) \Rightarrow \nexists x. \nexists \text{copil}(x) \vee \text{areMon}(x) \Rightarrow$$

$$\Rightarrow \exists x. \nexists \text{copil}(x) \vee \text{areMon}(x)$$

În mare:  $\nexists x. \text{copil}(x) \Rightarrow \text{areMon}(x)$

dorim să scriem legile de " $\Rightarrow$ "  $\Rightarrow$  căre?  $\Rightarrow$  univ y.

$$\Rightarrow \nexists x. \text{copil}(x); \quad \nexists x, \exists y. \text{areMon}(y, x) \Rightarrow$$

$$\Rightarrow \nexists x. \text{copil}(x) \Rightarrow \exists y. \text{areMon}(y, x)$$

9.  $\nexists x. \text{areCr, port}) \Rightarrow \nexists y. \text{are}(x, y)$ .

$\text{are}(eu, conte)$

+

$\text{are}(eu, port)$

||

$$\nexists x. \text{are}(x, conte) \Rightarrow \nexists y. \text{are}(x, y) \rightarrow \nexists x. \text{are}(x, conte) \wedge \nexists y. \text{are}(x, y) \rightarrow$$

$$\rightarrow \exists x. \nexists \text{are}(x, conte) \vee \exists y. \text{are}(x, y)$$

cand negație:  $\nexists \text{are}(x, port)$

Closure:  $\{\nexists \text{are}(x, conte), \text{are}(x, y)\}$  (1)

$\{\text{are}(eu, conte}\}$  (2)

$\{\nexists \text{are}(eu, port)\}$ . (?)

$$(1) \cup (2) \rightarrow x \in eu \Rightarrow \{\text{are}(eu, y)\} \setminus (1)$$

$$(1) \cup (?) \rightarrow y \in port \wedge \text{disjunc} \Rightarrow \Gamma \vdash \text{STOP} \Rightarrow$$

$$\Rightarrow \underline{\text{are}(eu, port)}$$

10.  $p([[], A, A]).$

$p([E \mid T], A, [E \mid R]) :- p(T, A, R).$

visible in A def A  $p(L_1, L_2, R)$ ?

Ce se intampla si cum descompunem cu egi si rezultat

$T \rightarrow R$  cu ocazii A  $\Rightarrow$  A este după R finalul lui T  $\Rightarrow \underline{L_1 + L_2}, ++, append$

12. 1)  $(f \neq s(1/2 \circ)) \rightarrow$  oare mai devă de unde suntem

2) let  $((f (\lambda (x y)$

$\frac{x)}{f} \rightarrow \lambda \text{ of } 2 \text{ arg} \Rightarrow \text{err.}$   
 $(f \leq (1/2 \circ)))$

3) let  $f x y = x \overset{?}{\in} s(\text{div } 2 \circ) \Rightarrow x=5, y=\text{div } 2 \circ$

4)  $x=2/5, y=x. \rightarrow$  nu este astfel  $\Rightarrow$  yes!, e ok.

15-a

17 17 17 17 14 13

$$\begin{aligned}
 1. & (\lambda y. ((\lambda x. \lambda y. x \cdot y) \cdot z) \lambda x. \lambda y. y) \xrightarrow{\alpha} \\
 & \xrightarrow{\alpha} (\lambda y. ((\underline{\lambda x. \lambda z. x} \underline{y}) \cdot z) \lambda x. \lambda y. y) \xrightarrow{\beta} \\
 & \xrightarrow{\beta} (\lambda y. (\underline{\lambda z. y} \cdot z) \lambda x. \lambda y. y) \xrightarrow{\beta} \\
 & \xrightarrow{\beta} (\lambda y. \underline{y} \underline{\lambda x. \lambda y. y}) \xrightarrow{\beta} \lambda x. \lambda y. y \text{ si loss of } \alpha.
 \end{aligned}$$

2. (define (setN L1 L2)

(filter (lambda (x)  
(member x L2)) L1))

3. !(define fnic ( $\lambda (a)$ )

$a = 'cena.$

2. (let  $L (f (F a))$

3. (g (delay (E a)))

4. f )) )

(f nuc (E 'cena))

declaratii si eval. F & E?

1 este peste E  $\Rightarrow$  E este final, la 4.

2 este in let  $\Rightarrow$  2.

chiar dacă f e în scope let - din deoarece el e defocat, deci nu trăiește

cât despre E, unde E nu poate fi  $\lambda \Rightarrow$  punct fct  $\lambda \dots$  pentru deosebit de E

4.  $f x = f (f x)$

$\rightarrow$  la un fel de compus.

$f :: a$

$f \circ fct \Rightarrow f :: R \rightarrow d$

f primește ca arg x  $\Rightarrow x :: e$

f se poate apela pe urmărt f x  $\Rightarrow f :: R \rightarrow R$

$x :: R; f :: e \rightarrow e$ .

sau:  $f :: a \rightarrow b$  explicit  
 $\frac{x :: a}{(f (f x)) :: b}$  mai sp  
 $f :: c \rightarrow d$   
 $(f x) :: c$   
 $d = b$   
 $f :: e \rightarrow g$   
 $x :: e = a$   
 $g = c$   
 din  $f : a = c = e, b = d = g$ .  
 $f :: a \rightarrow g, f : t \rightarrow e$ .

5. instance  $(\text{Ord } a) \Rightarrow \text{Ord} [a]$  where

$$(a_1 : \_) \leq (a_2 : \_) = a_1 \leq a_2$$

tuvo los vindores

[all\_] → Protocol => )

G. Clinics often duplicate:

$\text{Const} : [a] \rightarrow [a]$

-- cot de botó

area [ ] = [ ]

-- censor  $L = [m \mid \text{head } L = m \rightarrow \text{not } \text{seen}(\text{tail } L)] \rightarrow m =$

(*ext* (*x*: *xs*) = *x*. [*y* | *y*  $\leftarrow$  *cons* *xs*, *y*  $\neq$  *x*])

-- Sou  $\text{Res} \ell = \text{head } \ell : \text{cons}(\text{filter}(\text{rest} . (\_ == \text{head } \ell))) \& \text{toL} \ell$

7. Ce que s'écrit? (tire 10 \$ 20 avec (+) [1, 1, ..]) = (tire 10 \$ 1, ..., +)

↓ odoung 1 lo el bri's.  
 Si fa lo diente ell,  
 ↓ gen primell 10.  
 lo friend an  
 odifit 1st tr 1  
 munt & add 1  
 ↓  
 ↓  
 0 = 0, 1, ...  
 no def 0, 1, ... ps no ps odd 1.  
 ↓ primell 10  
 ↓ lo diente  
 ultimely elem.  
 ell bri's.  
 ↓ def pt is  
 primell 10 elem diente  
 si respond en al 2-los

$$H \rightarrow \underbrace{0, 1, 2, \dots}_{10} \rightarrow \underbrace{1, 2, 3, \dots}_{10} = \underbrace{1, \dots}_{10}$$

gated  $\Rightarrow$  a...  
↓ naturally al 3-les el. skin = 2.

$$8. \text{ Cine } \wedge \text{ Seats}, \text{ so ok?} \Rightarrow \text{Se Seats(cine)}, \text{ Se Seats(cine)} \\ \Leftrightarrow \text{color in cine. Color is cine? No?} \xrightarrow{\text{origin}} \\ \begin{array}{c} \forall x. \text{Se Seats}(x) \Rightarrow \forall x. \text{Se Seats}(x) \Rightarrow \text{Se Seats}(x), \\ \forall y. \text{Se Seats}(y) \end{array} \quad \frac{}{\exists x. \exists y. \text{Se Seats}(x) \wedge \text{Se Seats}(y)} \quad \text{neither} \\ \Downarrow \\ \exists x. \exists y. \text{Se Seats}(x) \vee \text{Se Seats}(y)$$

9. trace (Loc, Alt Loc)

st Adokoleu ch  
mijf.

trace (st, Adokoleu)

trace (Adokoleu, ch.)

$$\begin{array}{c} \forall x_1 \forall y. \text{trace}(x_1, y) \\ \forall z_1 \forall y. \text{trace}(y, z_1) \end{array} \Rightarrow \forall x \forall y \forall z. \text{trace}(x, z)$$

$$\text{deci: } \forall x \forall y. \text{trace}(x, y) \wedge \forall z. \text{trace}(y, z) \Rightarrow \text{trace}(x, z).$$

$$\exists x. \exists y. \exists z. \text{trace}(x, y) \vee \exists z. \text{trace}(y, z) \vee \text{trace}(x, z)$$

$$\Rightarrow \exists x. \exists y. \exists z. \text{trace}(x, y) \vee \exists z. \text{trace}(y, z) \vee \text{trace}(x, z).$$

$$\text{deci: } \{ \text{trace(st, Adokoleu)} \} (1)$$

$$\text{trace(st, ch)} \Rightarrow \text{nept}$$

$$\{ \text{trace(Adokoleu, ch)} \} (2)$$

$$\{ \text{trace}(x, y) \vee \exists z. \text{trace}(y, z) \vee \text{trace}(x, z) \} (3)$$

$$\{ \exists z. \text{trace}(st, ch) \} (4).$$

$$(1) \cup (3): x \leftarrow st, y \leftarrow Adokoleu \Rightarrow \{ \exists z. \text{trace}(Adokoleu, z) \vee \text{trace}(st, z) \} \quad (5)$$

$$(2) \cup (3): x \leftarrow Adokoleu, y \leftarrow ch \Rightarrow \{ \exists z. \text{trace}(ch, z) \vee \text{trace}(Adokoleu, z) \} \quad (6)$$

$$(2) \cup (5): z \leftarrow ch, \text{Se reduce, } \{ \text{trace}(st, ch) \} \quad (7)$$

$$(4) \cup (7) \cdot \Gamma \vdash \text{SVDL} \quad \square$$

10.  $\text{mapf}(+L, -LO)$        $\text{map } f, f(+x_1, -x_0)$

$\text{mapf}(+L, -LO) :- \text{findall}(X_0, (\text{member}(X, L) \wedge f(X, X_0)), LO)$

$\text{findall}(\quad, \text{goal}, \text{bag}).$

template ↓  
functor ↓ lists, occ.

template bags in bag.

11.  $p(L, [1, 2, 3]) \rightarrow ?$

$p(D, [A, B, C]) :- \text{member}(A, D), \text{member}(B, D), \text{member}(C, D).$

||  
  └→ A member  
  B member  
  C member      { dñ D. similitud.  
                  └→ offer map los, despues find en 4, p. se mapf

esto es un desplazamiento de la lista en el 1, ?, 3 apartir de "primitivo"; el resultado.

## IS-B

$$\begin{aligned}
 1. & ((\lambda x. \lambda y. \lambda z. y, \lambda x. x) (\lambda z. xt. z z)) \xrightarrow{\beta} \\
 & \rightarrow \beta ((\lambda y. \lambda z. y (\lambda z. xt. z z)) z) \xrightarrow{\beta} ((\lambda y. \lambda w. y (\lambda z. \underline{xt. z}) z)) z \xrightarrow{\beta} \\
 & \rightarrow \beta ((\lambda y. \lambda w. y \underline{xt. z}) z) \xrightarrow{\beta} (\lambda w. \underline{xt. z} z) \xrightarrow{\beta} xt. z
 \end{aligned}$$

2. (define (cons A B))

(let\* ((L1 (map list A B))) ; zip

(L2 (cons (map car L) (map cd L))) ; unzip

(L3 (apply append L3) ; une liste de deux listes de deux éléments)

; zip les éléments

(L4 (remove-duplicates L2)))

(L4))

3. 1. (define grec ( $\lambda(a)$ 

2. (let [ (f (delay (F a))) (x(g a)) ]

3. f ))

4. (grec (E largument))

delay nécessite force  $\Rightarrow$  ne se exécute F si on n'exécute pas (force f))F attend un argument, E est celui qu'il primeste, mais en fait il attend  
l'argument fournit par F apelé au cours l'argument  $\Rightarrow$  primeste liste a la  $\Rightarrow$  deux  
appels à force. Mais cette opération  $\Rightarrow$  deux F et appelle à force la ligne 4.

4. f x = x (f x)

5. instance  $(E_2 \xrightarrow{a_1} \text{Ord } b) \Rightarrow^{\text{My Ord}} (\alpha, b)$  where  
 $(-, a_1) \xrightarrow{f} (-, \alpha_2) = (\alpha, \sum a_2)$   
 $\Leftarrow \dim \text{Ord} \neq \# c = \dim \text{Ord}$

6.  $\text{deps} :: [\alpha] \rightarrow [\alpha]$

$\text{deps} [] = []$

$\text{deps} [x : xs] = [m \mid x = m, \text{elem } m \in xs]$

- Sou  $\text{deps}(h : t)$

-- elem  $h + = m$ :  $\text{deps}(\text{filter}(l = h) t)$

-- otherwise =  $\text{deps } t$

$\Rightarrow (\text{take } 10 \text{ $zipWith (+) s (fpr s)$}) = (\text{take } 10 \text{ $f (fpr . take) s$})$

is prime,  $\Rightarrow$  elem dim seems dim S & univ. el. dim S for S

except gen prime w sl 2-dec, sl 2-dec w sl takes

gen  $a_1 a_2 a_3 \dots$       esto se fijo w prime w sl  
 $| | |$        $a_1 a_2 a_3 a_4 \dots$   
 $a_2 a_3 a_4 \dots$        $\underbrace{\quad\quad\quad\quad}_{\text{fijo.}}$   
 $\left( \begin{array}{ccc} a_1 & a_2 & a_3 \\ + & + & + \\ a_2 & a_3 & a_4 \\ // & // & // \\ y_0 & a_4 & a_5 \end{array} \right)$

8. "Give one code, one porto"

one Code(Cine)  $\Rightarrow \forall x. \text{one Code}(x)$        $\forall x. \text{one Code} \Rightarrow \text{one Code}(x)$

one Porte(Cine)  $\Rightarrow \exists x. \text{one Porte}(x)$

Sou one(Cine, Ce);  $\forall x, \exists y. \text{one}(x, y)$

one(Cine, code)  $\Rightarrow$  one(Cine, porte)  $\Leftrightarrow$

$\underline{\underline{\Rightarrow \forall x. \text{one}(x, code) \Rightarrow \text{one}(x, porte)}}$

9. „elefantul e mai mare decât leu“  $\Rightarrow \forall x, y. \text{maiMare}(x, y) \text{ general}$   
 „leu e mai mare decât socicel“  
 „mai mare = & transitiiv“  
 ↓  
 $\forall x \forall y \forall z. \text{maiMare}(x, y) \wedge \text{maiMare}(y, z) \Rightarrow \text{maiMare}(x, z)$

maiMare(elefant, leu);  
 maiMare(leu, socicel);  
 concluzie maiMare(Elefant, socicel)

$$\forall x \forall y \forall z. \text{maiMare}(x, y) \wedge \text{maiMare}(y, z) \Rightarrow \text{maiMare}(x, z)$$

$$\text{Supozitie implicatiu} \Rightarrow \exists x \exists y \exists z. \text{maiMare}(x, y) \vee \text{maiMare}(y, z) \vee \text{maiMare}(x, z)$$

$$\Rightarrow \exists x \exists y \exists z. \text{maiMare}(x, y) \vee \text{maiMare}(y, z) \vee \text{maiMare}(x, z)$$

daca cō „elefantul e mai mare decât socicel“  $\exists x \exists y \exists z. \text{maiMare}(x, y) \vee \text{maiMare}(y, z) \vee \text{maiMare}(x, z)$

$$\text{Pp RA} \hookrightarrow \exists \text{maiMare}(\text{elefant}, \text{socicel}) \Rightarrow \text{Clonare}: \{ \exists \text{maiMare}(x, y) \vee \exists \text{maiMare}(y, z) \vee \exists \text{maiMare}(x, z) \} \{ 1 \}$$

Concluzie negata:  $\neg \exists \text{maiMare}(\text{elefant}, \text{socicel})$

$$\Rightarrow \{ 1 \} \cup \{ 2 \}: x \leftarrow \text{elefant}, y \leftarrow \text{leu}$$

$\Rightarrow$  rezolvare maiMare si

$$\{ \exists \text{maiMare}(\text{leu}, z) \vee \text{maiMare}(\text{elefant}, z) \} \{ 5 \}$$

$$\{ 1 \} \cup \{ 3 \}: x \leftarrow \text{leu}, y \leftarrow \text{socicel}, \& rezolvare maiMare si \{ \exists \text{maiMare}(\text{socicel}, z) \} \{ 6 \}$$

$$\{ 1 \} \cup \{ 4 \}: x \leftarrow \text{elefant}, z \leftarrow \text{socicel}, \& rezolvare maiMare si \{ \exists \text{maiMare}(\text{elefant}, y) \vee \exists \text{maiMare}(y, \text{socicel}) \} \{ 7 \}$$

$$\{ 3 \} \cup \{ 5 \}: z \leftarrow \text{socicel} \& rezolvare primul MaiMare  $\Rightarrow \{ \text{maiMare}(\text{elefant}, \text{socicel}) \} \{ 8 \}$$$

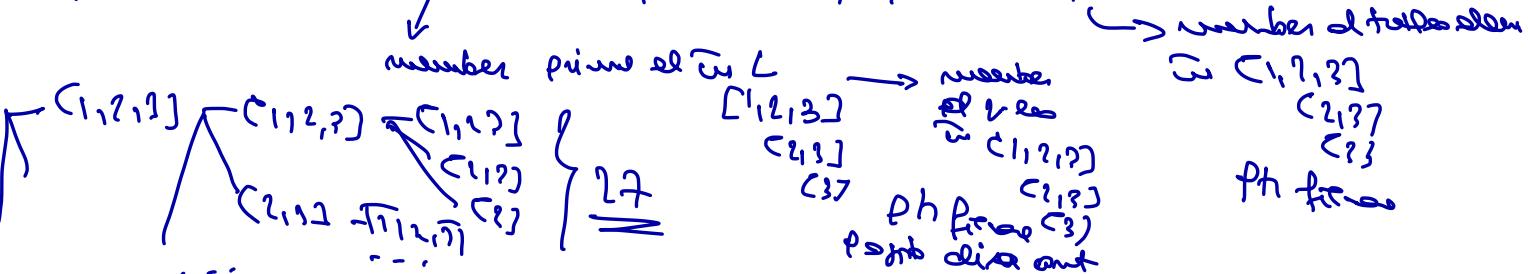
$$\{ 8 \} \cup \{ 4 \}: \Gamma \perp \text{STOP} \Rightarrow \text{maiMare}(\text{elefant}, \text{socicel})$$

$$10. \text{filterF}(+L, -LO) = \text{findAll}(X, (\text{member}(X, L), \neq(X))), LO$$

$$\neq(+X)$$

$$11. p([1, 2, 3], L) = ?$$

$p(A, [A, B, C]) :- \text{member}(A, D), \text{member}(B, D), \text{member}(C, D).$



wlu - A

$$\begin{aligned}
 & \xrightarrow{\alpha} ((\lambda y. (\lambda x. \lambda y. x \ y)) \lambda z. z) \ z \rightarrow ((\lambda y. (\lambda x. \lambda z. x \ y)) \lambda v. v) \ v \\
 & \xrightarrow{\beta} (\lambda y. \lambda z. y \ (\lambda x. x)) \ v \xrightarrow{\beta} (\lambda z. \lambda x. x) \ v \rightarrow \lambda x. x
 \end{aligned}$$

2. (define (expt L 2))

```
(filter(lambda(l) (number (length l) 4)) l2))
```

$$3. \text{ (Let } x=1)$$

$$(y=2) \quad y=2$$

$\text{cf}(\text{delay } (\lambda(y) \text{ (+, } x, y)))$

3

Let  $\{x_i\}_{i=1}^n$

((force f) x)

)  
)

↪  $1+y, y$  edot o pann, adi $\delta$ , x

da  $x$  denkt int  $\Rightarrow$

$$4. \text{ Ceros: } [a] \rightarrow [b] \rightarrow [(a, b)]$$

$$\begin{array}{r} 1+5=6 \\ \hline \end{array}$$

new [ ] [ ] = [ ]

$\text{rem } A \ B = \text{comot}(\text{map}(x \rightarrow \text{map}(y \rightarrow (x, y)) \ y) \ x)$

LIST COMPREHENSIONS.

$$c_{\text{ext}} A B = \{(x, y) \mid x \in A, y \in B\}$$

5 flex pertei 2:

$c_{\text{rea}} = 1: \text{map } (*2) \text{ (lue)}$

$$= \{2^x \mid x \in \text{Co.}\}$$

= [2 \* i | i <= iterate (≈ 2) 1]

= iterate ( $\neq 2$ ) 1

$$= \{g_{\omega_2}^{-1}(i) \in C_0\}$$

where

`pow :: Integer -> Integer`

QSWR 0 = 1

$$\text{pow2 } m=2 \rightarrow \text{pow2}(m-1)$$

6.  $f(x) = [(y_1, y_2) \mid (y_1, z, t) \in x, y_1 = z]$

$f$  este funcție  $\Rightarrow f: A \rightarrow B$

$x$  este  $A$  și  $B = A$

este multime  $[c]$

este multime de perechi  $[(d, d)] \Rightarrow y :: d$ .

$(y, z, t) \in x \Rightarrow$  este triplet, și fi tipuri  $\approx$

$y = z \Rightarrow z :: d$ . deci  $(y, z, t) \in x \Rightarrow$

rezultatul  $:: [(d, d)]$

este rezultatul  $:: b \approx$

$$b = \underline{\underline{[(d, d)]}}$$

$f: [(d, d, e)] \rightarrow [(d, d)]$

listă de modificări ale unui triplet s.t. să

țină de o singură tip, deci un triplet cu diferențe de tip și diferențe

$\approx$   $x$  este variabilă,  
 $y = z \approx y$  dezvoltă  
trece.

$y :: d, z :: d$

final triplet, și fi  
tipuri diferențe

$\approx t :: e$

$x :: (d, d, e)$

$x :: [(d, d, e)]$

$r: a \Rightarrow [(d, d, e)] = a$

$f: a \Rightarrow f: [(d, d, e)]$

7. instance (`Num a, Show b`)  $\Rightarrow$  `Show(a ~> b)` where

`Show f = show(f 0)`

8. „Cine tocă și nu înțeleapt de către cine urbește”. Atunci

Predicale:  $\forall x. \text{tocă}(x)$ ;  $\forall x, y. \text{nu}^i \text{Înteleapt}(x, y)$ ;  $\forall x. \text{urbește}(x)$ .

$\forall x \forall y. \text{tocă}(x) \wedge \text{urbește}(y) \Rightarrow \text{nu}^i \text{Înteleapt}(x, y)$  AFAT, NU TRANS

9. „Din numărul toti somonii sunt membru, Socote, sau el îngrijit, NMK este membru.”

$\forall x. \text{membru}(x)$ ;  $\forall x. \text{Socote}(x)$ ; (predicale)

totii somonii sunt membri  $\Rightarrow \forall x. \text{membru}(x) \Rightarrow \text{membru}(x) \Rightarrow$

$\Rightarrow$  Separare implicativă:  $\forall x. \text{Pm}(x) \vee \text{membru}(x) \Rightarrow \exists x. \text{Pm}(x) \wedge \text{membru}(x)$

Socote, sau el îngrijit  $\Rightarrow \text{membru}(\text{Socote})$ ; condizanță:  $\text{membru}(\text{Socote})$ ;

In-B

1.  $((\lambda y. (\lambda x. \lambda y. x y) \lambda x. x) z) \xrightarrow{\beta} ((\lambda y. (\underline{\lambda x. \lambda z. x y}) \lambda x. x) z) \xrightarrow{\beta}$   
 $\rightarrow \beta ((\underline{\lambda y.} \lambda z. y \underline{\lambda x. x}) z) \xrightarrow{\beta} (\underline{\lambda z.} \underline{\lambda x. x} \underline{z}) = \lambda x. x$

2. (let + (x 3) x=3

$$(y u) \quad y=4 \\ (\text{let } y \Rightarrow \lambda(y) = y_1 + 3 \\ (\text{f(delay } \lambda(y) (+ x y)))]$$

(let [(x 1)]  $\rightarrow$  let x, x = 1

slight difference  $\Rightarrow$

$$((\text{force } f) x) \\ \hookrightarrow 1 = 1 + 3 = 4$$

nested send f  $\rightarrow$  f = 1

3. (define (even L1 L2)

(filter (lambda (L) (not (member (length L) L1))) L2))

5. instance (new a, show b)  $\Rightarrow$  show (a, b) where

$$\text{show } f = \text{show } (f 1)$$

6. even = iterate (+u) 1 nested

10. down(+L, -LS)

down([], []):- !

down([H|T], LS):- down(T, S1),

findall(X, (member(X, S1), X < H), S2)

LS = [S2 | S1].

11. transformer(L, x) = ?

$h \in x$  were not depends

processor([H|T], x):- member(H, x), processor(T, x).  $h \in L$

transformer(L, x):- length(L, N), length(x, N), processor(L, x).

$\hookrightarrow$  length L = length x

$\hookrightarrow$  an accept length instead.

## 13-A

$$\begin{aligned}
 1. & (\lambda x. \underbrace{\lambda y. \lambda z.}_{\approx} \lambda x. x) (\lambda x. (\lambda x. x) \lambda x. (\lambda x. x))) y \xrightarrow{\beta} (\lambda y. \lambda x. (\lambda x. (\lambda x. x) \lambda x. (\lambda x. x))) \\
 & \xrightarrow{\beta} \lambda x. (\lambda x. (\lambda x. x) \lambda x. (\lambda x. x)) (\lambda x. x) \quad \text{if } \underline{x} = x.
 \end{aligned}$$

2. (define (oddstarts L)

$$\begin{aligned}
 & (\text{length} (\text{filter} \#t (\text{map} (\text{lambda} (L) (\text{odd?} (\text{car} L)))) L))) \\
 & ;(\text{length} (\text{filter} \text{odd?} (\text{map} \text{car} L)))
 \end{aligned}$$

3. (apply

(lambda (x y)

$$\begin{aligned}
 & (\text{let} ([x 1] \quad x=1 \quad \downarrow \Rightarrow y, \\
 & \quad [y 2]) \quad y=2 \\
 & \quad (+ x y)) \quad \cancel{y}=3 \\
 & (\text{let} ([x 2] \quad \text{alt } x \Rightarrow x=2 \\
 & \quad [y 3]) \quad \text{alt } y \Rightarrow y=3 \\
 & \quad (\text{list} x y)) \quad \text{list} \Rightarrow [2 3]
 \end{aligned}$$

(3)

4. (define f (delay (lambda (a) (display a) (newline)))

(define ff (force f))  $\xrightarrow{\text{force } a}$ (and ff 1)  $\xrightarrow{\text{ff } 2 \#f}$  (and ff 1 alt ff 2 #f)

w ant#f da w ff : 1&amp;2 ; due di 1, ff 2 è alt ff.

5. (let selector x y z = x in selector (+ 2) (+ 3) (+ 4)).

↓ funzione prot. pura. Non è una mut, se fatti due valori prima

di f si deve prima selezionare se si fa prima f si,

$$6. f \times y = (x \ y) \ y$$

$$f :: a \rightarrow b$$

$x \ y :: a \Rightarrow f \text{ et } f \text{ d de 2 var}$

$$\Rightarrow f :: F \rightarrow d \rightarrow b \Rightarrow x :: c$$

$$y :: d$$

$(x \ y) \Rightarrow x \in fct, y \text{ response}$

$$x :: e \rightarrow g \equiv_c \Rightarrow f :: (e \rightarrow g) \rightarrow d \rightarrow b.$$

$x \text{ primitive pr } y \Rightarrow x :: d \rightarrow g, e \in d$

$$\Rightarrow f :: (d \rightarrow g) \rightarrow d \rightarrow b$$

$(x \ y) \ y \Rightarrow (x \ y) \circ fct \Rightarrow \text{don } (x \ y) \ primitive$

$$x :: d \rightarrow (i \rightarrow i), g \equiv i \rightarrow i \quad \text{et } y \Rightarrow i \in d$$

$$\Rightarrow f :: \underbrace{(d \rightarrow (i \rightarrow i))}_{x} \rightarrow \underbrace{d}_{y} \rightarrow i$$

$$f :: (d \rightarrow (i \rightarrow i)) \rightarrow d \rightarrow i$$

$$y \in y$$

$$f :: (d \rightarrow (i \rightarrow i)) \rightarrow d \rightarrow i$$

7. instance (`New(a, Ord b)`)  $\Rightarrow$  `Ord (a  $\rightarrow$  b)` where

$$f_1 \leftarrow f_2 = f_1 \circ \leftarrow = f_2 \circ$$

8. "Pentru fiecare există un moment în care își cunoaște predecesor".

`One(Person)`; `cunoastePredecesor(Person, Moment)`.

$\forall x. \text{one}(x); \forall x \exists y. \text{cunoastePredecesor}(x, y);$

$\forall x \exists y. \text{one}(x) \Rightarrow \text{cunoastePredecesor}(x, y),$

$\forall x. \text{one}(x) \Rightarrow \exists t. \exists p. \text{predecesor}(p, x) \wedge \text{cunoaste}(x, p, t).$

deci, predecesor  $\Leftarrow \exists ! j$  cu care x este în primul loc predecesor la mom. t.

9. `filter(+L,+T,-LF)`

`filter([ ], T, [ ])`.

`filter([H|T], T, LF) :-`

$\begin{cases} :- \text{filter}(T, T, LF), \text{findAll}(X, \text{member}(X, L), LF) \\ (X, LF1, X > T), LF2, LF = ([LF1 | LF2]) \end{cases}$

$\begin{cases} :- \text{findAll}(X, \text{member}(X, L), X > T), LF. \end{cases}$

10.  $\text{cens}(+L, -LO)$ .

$\text{cens}([ ], [ ])$ .

$\text{cens}([H|T], LO) :- \text{findall}(X, (\text{member}(X, T), X \neq H), \overline{T}),$   
 $\text{cens}(\overline{T}, LO).$

2017-18

1.  $((\lambda z. \lambda y. \lambda x. (y z) y) \lambda y. y) \xrightarrow{\alpha} \lambda z. \lambda w. \lambda x. (w z) y \lambda y. y \xrightarrow{\beta}$   
deci' esto n. ore nico comu = ))

$$\rightarrow (\lambda w. \lambda x. (\underline{w} \underline{y}) \underline{\lambda y. y}) \xrightarrow{\beta} \lambda x. (\lambda y. y) \xrightarrow{\beta} \lambda x. y.$$

2. (let ((a 1))

(b 2))

(+ a b)

)

1 & 2 si lleva lo a & b

sou a & b primos relativos

solo si cum (a b)

e' un int capaces let-val;

el scuent (a b) visible  $\Rightarrow$  ret 3.

no p'ect spars e let source

cela 2 ~~orden~~ p'c seco n' l'mo,

da difes obvianca  $\Rightarrow$  Nu e nico dif.

3. (define (f L)

(let\* ((obsL (map obs L)))

(filteredL (filter (lambda(y) (and (map (lambda(x) (> x y)) obsL))))

(apply map filteredL)))

SPM

; (or (filter (lambda(e) (null? (filter (compose (lambda(x) e) abs) L)))) L))

; (or (filter (lambda(e) (null? (filter (lambda(a) (< e (obs a))) L)))) L))

; (last (sort L <))

4.  $f \circ g \in \text{let } p_2 = \text{filter } (g, u) \text{ in } t + l_2$

$f \in \text{functor} \Rightarrow f: a \rightarrow b$ .

$g \in \text{monad lawi } f \Rightarrow g: c$

$u \in \text{unit lawi } f \Rightarrow u: d$

$t \in \text{unit lawi } f \Rightarrow t: e$

$p_2 \in \text{let monad lawi } f \Rightarrow p_2: f$

filter este fact,  $g, u \in \text{fd}, g \in \text{fd}, u \in \text{fd} \Rightarrow g: f \rightarrow e \in$   
 $\text{filter } \xrightarrow{\text{list}} h: n \rightarrow n, \exists d$

$t + g \text{ aplicat pe lista } \Rightarrow t \wedge p_2 \text{ sunt liste } \Rightarrow t_1: [e], p_2: [f]$

$g, u \Rightarrow g(u(...))$

$\Rightarrow \text{daca } u \text{ primeaza } \text{arg } g \Rightarrow u = g \Rightarrow g: n \rightarrow e$

$g \circ u : n \rightarrow e \Rightarrow h = \text{functie si}$

$\text{filter fd liste, fd care corespunde}$

$g \circ h \text{ aplicat pe liste, primeaza } \text{corespunde } \text{trebuie liste}, \text{ale}$

$\Rightarrow t_1: [n], t_2: [ne] \Rightarrow$

$\Rightarrow f: (n \xrightarrow{\text{functie}}) \rightarrow (n \rightarrow n) \rightarrow (n) \rightarrow (n) \rightarrow (e)$

$(g, u) \text{ instance care au } \text{functii } \text{interiori } g, u, \text{care } e \text{ este lista}$

daca, daca de la capat cu nu restul:

$f: (a \rightarrow b) \rightarrow (c \rightarrow a) \rightarrow (c) \rightarrow (c) \rightarrow (c) \rightarrow (c)$

$\underbrace{g}_{\text{ }} \quad \underbrace{h}_{\text{ }} \quad \underbrace{t_1}_{\text{ }} \quad \underbrace{p_2}_{\text{ }} \quad \underbrace{\text{restul}}_{\text{ }}$

5. definim o mai close pt. tipul colectiei care este tipul v def. a fd. (leste) puncte  
si direct, toti fd care sunt. V. sau din stoc.

class Ended t where frontEnd :: t v -> v; backEnd :: t v -> v

instance Ended Triple where este primul triplet tipului.

$\text{frontEnd } (T \ x \ _- \ -) = x$

$\text{backEnd } (T \ - \ - \ x) = x$ .

6. „Ucenic ued so ſame pre došol“;

ucenic(luke); došol(yoda)

mezo-inverte(luke,yoda).

$\forall x. \text{ucenic}(x); \forall y. \text{došol}(y) \Rightarrow \exists z. \text{mezo-inverte}(x, y)$

$\forall x \forall y. \text{ucenic}(x) \wedge \text{došol}(y) \Rightarrow \text{mezo-inverte}(x, y)$

Eliminacna implikcia  $\Rightarrow \neg \forall x \neg \forall y. \neg \text{ucenic}(x) \vee \neg \text{došol}(y) \vee \text{mezo-inverte}(x, y)$ .

$\Rightarrow \exists x \exists y. \exists \text{ucenic}(x) \vee \exists \text{došol}(y) \vee \neg \text{mezo-inverte}(x, y)$ .

$\{\exists x \exists y. \exists \text{ucenic}(x), \exists \text{došol}(y), \neg \text{mezo-inverte}(x, y)\}, (1)$

$\{\text{ucenic}(luke), \text{došol}(yoda)\} \models (2)$ .

Concluzia: mezo-inverte(luke,yoda);

Negacna vydelenie:  $\neg \text{mezo-inverte}(luke,yoda)$ ;

Pp RAZO  $\xrightarrow{\quad}$  Choose (1), (2) Pre cas mezo

$\{\neg \text{mezo-inverte}(luke, yoda)\} \models (3)$

dim(1)  $\wedge$  (1)  $\Rightarrow x \leftarrow \text{luke}, y \leftarrow \text{yoda}; \neg \text{ucenic ucenic} \wedge \text{došol} \Leftarrow$   
 $\neg \text{mezo-inverte}(luke, yoda)\} \models (4)$

dim(3)  $\wedge$  (4)  $\Rightarrow \perp \Rightarrow \text{STOP} \Rightarrow \text{mezo-inverte}(luke, yoda)$ .

7.  $x(+L, -M)$

$x([ ], 999)$ .

$\% x(L, M) :- \text{findAll}(Y, (\text{number}(Y, L), Y < M), M). - \text{NU}$

$\% x(L, M) :- \text{forall}(\text{number}(E, L), X \in E). - \text{PROFI}$

$x([H|T], M) :- x(T, M_1), (H < M_1 \rightarrow M = H; M = M_1).$

$$1. ((\lambda x. \lambda y. \lambda z (x y) \lambda x. x) z) \xrightarrow{\beta} ((\lambda x. \lambda y. \lambda w (x y) \lambda x. x) z) \xrightarrow{\beta}$$

new = true, i devine după deoseb = > d

$$\xrightarrow{\beta} (\lambda y. \lambda w (\lambda x. x y) z) \xrightarrow{\beta} (\lambda y. \lambda w. y z) \xrightarrow{\beta} \lambda w. z.$$

2. (let ((a 1))  
 (let ((b a))  
 (+ a b)))

{      (let \* ((a 1))  
               (b a))  
               (+ a b))

↓

deci: cu cel primul

let începe cu dătura let

↑ a din primul let este utilizat

în al doilea let și este scrie

⇒ aceeași bucură de red

introducere 2.

acei legători sunt a evităto să fie

deci (b a) nu este asemenea, b chiar b = q?

⇒ introducere 2

↓

nu e nicio diferență pe cele două lucru,

totuși în final se obține pe o lăție

pe b loc = q loc la 1 și fără semnificativ.

### 3. (define (f L))

(car (reverse (rest L))))

; deoarece scris 'dumne' la celelalte nr, nu e o eroare

; ( car ( filter (λ(e) (null? (filter (λ(c) (< e c)) L)) L)))

doar nu  
există.

mai mult decât de să filtreze el din L

doar să filtreze el din L și să nu fie posibil să mai mai.,  
deoarece, filtrează în primul rând, rezultatul

doar să filtreze el din L

și urmărește să nu fie posibil să mai mai.,  
deoarece, filtrează în primul rând, rezultatul

4.  $f \circ g \circ h = map(g \circ h) \circ f$

$$f :: a \rightarrow b \quad map :: (t \rightarrow +2) \rightarrow [t]$$

$$g \circ h :: i \rightarrow k$$

$$g :: c$$

$g \circ h$  funcție, compunere, infițătă în listă  $\Rightarrow$

$$f :: (j \rightarrow k) \rightarrow ([i] \rightarrow [k])$$

$$h :: d$$

$$\Rightarrow f :: [i];$$

$$g :: j \rightarrow k$$

$$do \downarrow$$

$$f :: e$$

aceeași tip

uri care să fie corecte pentru g · h

$$g (u(t))$$

g este o funcție care returnează

$$h \circ g = h :: i$$

5. Def class Fronted, tip select & exist pos to v def o let go each (viral & o also can each left elem, pth & lists)

Ques Fronted t where frontend :: t v → v; backend :: t v → v

instance Fronted [] where

frontend = head

backend = head, reverse,

6. Give spine mult, spine moi putting & spine trace.

spine-mult(I<sub>in</sub>), trace(M<sub>out</sub>) ; spine-moi.putin(I<sub>in</sub>, P<sub>out</sub>).

if x.spine-mult(x) ∧ ∃ y.trace(y) ⇒ spine-moi.putin(x, y).

¬ ∀ x. ¬ spine-mult(x) ∨ ? ∃ y. ?trace(y) ∨ spine-moi.putin(x, y).

?x. ?spine-mult(x) ∨ ?Hy. ?trace(y) ∨ spine-moi.putin(x, y).

spine-moi.putin(I<sub>in</sub>, P<sub>out</sub>) ⇒ ?spine-moi.putin(I<sub>in</sub>, P<sub>out</sub>)

P<sub>p</sub> R A ⊢ ————— ⇒

close 2: {spine-mult(P<sub>in</sub>)} (1)

{trace(M<sub>out</sub>)} (2)

{?spine-mult(x), ?trace(y), spine-moi.putin(x, y)} (3)

{?spine-moi.putin(i<sub>in</sub>, m<sub>out</sub>)} (4)

(1) ⊢ (3) ⇒ x ← i<sub>in</sub>, di ⊢ spine-mult ⇒ {?trace(y), spine-moi.putin(i<sub>in</sub>, y)} ⊢ (5)

(2) ⊢ (5) ⇒ y ← m<sub>out</sub>, dispense trace ⇒ {spine-moi.putin(i<sub>in</sub>, m<sub>out</sub>)} (6)

(6) ⊢ (4) ⇒ Γ ⊢ STOP. ⇒ spine-moi.putin(i<sub>in</sub>, m<sub>out</sub>).

7. X(L, A, B, N) defn & o link in defn > A, < B.

X(+L, +A, +B, -N)

X([ ], A, B, 0)

X(L, A, B, N) :- findall(X, member(X, L), X>A, X<B, S), length(S, N).

# 1017-B

$$1. ((\lambda x. \lambda y. \lambda z (y \ x) \ y) \ \lambda z \ z) \rightarrow ((\underline{\lambda x. \lambda w} \ \underline{\lambda z. (w \ x)} \ y) \ \lambda z \ z) \xrightarrow{\beta} \\ \xrightarrow{\beta} (\underline{\lambda w. \lambda z. (w \ y)} \ \lambda z \ z) \xrightarrow{\beta} \lambda z. (\underline{\lambda z. z} \ y) \xrightarrow{\beta} \lambda z. y.$$

2. (define a 2)

$\xrightarrow{\beta} a = 2$

(let ((a 1)  $\Rightarrow a' = 1$ )

$(b \ a)) \xrightarrow{\beta} b = a' = 1 \xrightarrow{\beta} b = 1$

$(+ a \ b)) \xrightarrow{\beta} 2! + b = 2 + 1 \xrightarrow{\beta} 3$

with def p for let  $\xrightarrow{\beta} a$

new def to p old a  
ci p p prim o, is  
es una p interno define.  
p uade p old a

(define a 2)  $\xrightarrow{\beta} a = 2$

(letrec ((a 1)  $\Rightarrow a' = 1$ )  $\rightarrow$  let a

$(b \ a)) \xrightarrow{\beta} b = a' = 1 \xrightarrow{\beta} b = 1$

$(+ a \ b)) \xrightarrow{\beta} 2! + b = 2 + 1 \xrightarrow{\beta} 3$

$\hookrightarrow$  corp let  $\rightarrow$  el uade p old a,  
be old o;

3. (define (f L))

; (coi filter (lambda (null? (filter (lambda (x) (> x a)) L)) L))).

jsou

;(coi sort L))

(coi sort L)).

4. f x y z = filtering [x, y, z]

f::a  $\rightarrow$  b

x::c, y::d, z::e, g::h  $\Rightarrow a \equiv c \rightarrow d \rightarrow e \rightarrow h$

g::i  $\rightarrow$  j; j = Bool; g :: i  $\rightarrow$  Bool; h  $\equiv$  j  $\rightarrow$  Bool.

$\Rightarrow a \equiv c \rightarrow d \rightarrow e \rightarrow (i \rightarrow \text{Bool})$   $\xrightarrow{\beta} a \equiv j \rightarrow j \rightarrow j \rightarrow (j \rightarrow \text{Bool})$

$(x, y, z) \Rightarrow x :: j, y :: j, z :: j$  do r e p e list

b = [j]

$\Downarrow j = j$

$a \equiv j \rightarrow j \rightarrow j \rightarrow (j \rightarrow \text{Bool})$

f:: j  $\rightarrow$  j  $\rightarrow$  j  $\rightarrow$  (j  $\rightarrow$  Bool)  $\rightarrow$  [j]

5. close w/o fl. dots. Point  $a = \text{makePoint } a$

close Ended + where frontEnd :: +  $v \rightarrow v$ ; backEnd :: +  $v \rightarrow v$ .

instance Ended Point where

frontEnd (MakePoint  $x_+$ ) =  $x$

backEnd (MakePoint  $-x$ ) =  $x$

6. astene(Nectar, bine)

one(Nectar)

desire(Nectar, bine).

$\forall x. \text{one}(x) \wedge \forall y. \text{astene}(x, y) \Rightarrow \text{desire}(x, y)$ .

$\exists x. \text{one}(x) \vee \exists y. \text{astene}(x, y) \vee \text{desire}(x, y)$ .

$\exists x. \text{one}(x) \vee \exists y. \text{astene}(x, y) \vee \text{desire}(x, y)$ .

$\{\text{one}(x), \text{astene}(x, y), \text{desire}(x, y)\} \{1\}$

$\{\text{astene}(\text{Nectar}, \text{bine})\} \{2\}$ .

$\{\text{one}(\text{Nectar})\} \{3\}$

$\{\text{desire}(\text{Nectar}, \text{bine})\} \{4\}$

dim(1)  $\cup$  dim(2) :  $x \in \text{Nectar}, y \in \text{bine} \Rightarrow$

$\Rightarrow \{\text{one}(\text{Nectar}), \text{desire}(\text{Nectar}, \text{bine})\} \{5\}$

dim(5)  $\cup$  dim(3)  $\Rightarrow \{\text{desire}(\text{Nectar}, \text{bine})\} \{6\}$

dim(6)  $\cup$  dim(4)  $\Rightarrow \Gamma \Rightarrow \text{STOP}$ .

$\forall x(L, A, B, N) : - \text{findAll}(x, \text{elements}(x, L), x < A, x \stackrel{>}{=} B, S),$   
 $\text{length}(S, N)$ .

2017-0

$$\begin{aligned} 1. ((\lambda x. \lambda y. \lambda z. (x y)) \lambda x. y) &\xrightarrow{\alpha} ((\underline{\lambda x. \lambda w. \lambda z. (x w)}) \underline{\lambda x. y}) \xrightarrow{\beta} \\ &\xrightarrow{\beta} (\lambda w. \lambda z. (\lambda x. y \underline{w})) \xrightarrow{\alpha} \lambda z. (\underline{\lambda x. y} \underline{z}) \rightarrow \lambda z. y. \end{aligned}$$

2. - for B

3. (define f L)

(co (filter () @) (null? (filter (lambda (a) (compose  
(even?) (obj a)) obj) L)) L)))

4.  $f : x \# y \# g = \text{map } g [x, y, z]$

$f : a \rightarrow b$

$x : c, y : d, z : e, g : h$        $f : i \rightarrow j \rightarrow (k \rightarrow (l \rightarrow j)) \rightarrow [j]$ ,  
 $\text{map} : (i \rightarrow j) [k] \rightarrow g^o$   
 $\Downarrow$   
 $g : i \rightarrow j \equiv h$   
 $x : f, z : i, g : u$

5. class str. data NestedL a = A o | C [NestedL]

class Ended + where frontEnd :: t v → v; backEnd :: t v → v

instance Ended NestedL where

frontEnd (A a) = a; frontEnd (C e) = frontEnd \$ head e  
backEnd (A a) = a; backEnd (C e) = backEnd \$ last e

6. „Um bezüglich einer Person gibt es“

begat(bf), son(bab), mother(bab)

friends(bab.) +  
                   ||

$\forall x. \text{begat}(x); \text{brother}(x); \forall x. \text{mother}(x); \text{brother}(x);$

$\forall x \forall y. \text{begat}(x) \wedge \text{son}(y) \wedge \text{mother}(x) \Rightarrow \text{friends}(y). \rightarrow$

$\rightarrow \exists x \exists y. \forall z. \text{begat}(x) \vee \text{son}(y) \vee \forall z. \text{mother}(x) \vee \text{friends}(y).$

Closure: { $\lambda$  bill. { $\lambda$  song(bill),  $\lambda$  move(bill),  $\lambda$  fly(bill)} (1)}

{ $\lambda$  best(bill) (2)}

{ $\lambda$  son(bill) (3)}

{ $\lambda$  move(bill) (4)}

{ $\lambda$  flies(bill) (5)}

(1)  $w(1) \Rightarrow x \leftarrow \text{bill} \Rightarrow \{\lambda \text{ song}(y); \text{move}(\text{bill});$   
 $\qquad \qquad \qquad \text{flies}(y)\} (6)$

(5)  $w(5) \Rightarrow y \leftarrow \text{bill} \Rightarrow \{\lambda \text{ song}(\text{bill}), \lambda \text{ move}(\text{bill})\}$

(7)  $w(4) \Rightarrow \{\lambda \text{ son}(\text{bill})\} (8) \quad (7)$

(8)  $w(3) \Rightarrow \Gamma \downarrow$

7.  $x(L, M) = \text{next\_list}(L, M)$

$x(L, M) :- \text{member}(M, L), \text{forall}(\text{member}(E, L), x > E).$