

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 λy la care nu i se dă

dobândită o nouă părție tehnice exprimă singurul beta-redus

Corp: $\lambda x. (\underline{y} \; y)$ C e deosebit deosebit părție

Prin lombod: y Prin lombod e primul λ

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

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

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

lomb lomb y, nu ajută să sub corporul unei let sau să prin lombod x \Rightarrow

\Rightarrow x e nu liber în părție x de la λx

\Downarrow
nu încearcă să 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 împreună cu $(x \; x)$

$\Rightarrow \lambda z. ((x \; x) \; (x \; x)) \rightarrow$ rezultat 1st folos.

\downarrow
suntem păred? - NU. \Rightarrow suntem păred?

2. (define y 10)

$$\left(\begin{array}{c} (\text{lombodo}(x)) \\ \left[\begin{array}{c} (\text{lombodo}(y)) \\ \left[\begin{array}{c} (\text{if } (>x\; 2) \\ (+\; 1\; y) \\ (+\; x\; y) \end{array} \right] \end{array} \right] \end{array} \right) s$$

Jo nu îl suntem? nu, nu este î

Jo nu îl suntem nu? nu, părțile sunt expresii
înfoare părțile includeră funcționale

$(\text{lombodo}(y))(\text{if } (>x\; 2)(+\; 1\; y)(+\; x\; y))$ nu

x legăt de S ($x \in S$), e să și cum suntem.

$(\text{lombodo}(y))(\text{if } (>S\; 2)(+\; 1\; y)(+\; S\; y)) \Rightarrow$
 $\Rightarrow (\text{lombodo}(y))(+\; 1\; y))$ înfoare

JK... \Rightarrow funcție nu este aplicabilă mai departe
 \Rightarrow nu suntem din urmă să nu se aplice

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 $x \mapsto (\text{filter} (\text{lambda}(y)) (\text{zero?} (\text{remainder} (x, y)))) \ L$)

Vom să producă 2 liste de perechi unde $x \in L_1 \Rightarrow$ perechi cu x și din $L_1 \Rightarrow$

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

4. Aceea că se poate scrie un lambda care să facă și un L și să doară L nu este chiar

știrea că suntem nevoiți să scriem $f \circ g$ sau $f \circ g \circ h$ și că f este primul element din $L \circ f \Rightarrow$ nu e posibil

și doar nevoie să scriem $[]$

\Downarrow
 rezolvăm ca să scriem filter și să scriem map.

(let (if odd.)

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

$(\text{filter} (\text{map } g \ L)))$ \Downarrow f este map și g este filter

) Suntem 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 f este expresia 1)

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

: $\text{cons} = \text{expresia } 3$

$f (\text{tail } x) (\text{tail } y) = \text{expresia } 4$ (f explicit pe ...)

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

$\Rightarrow f$ este de tipul $a \rightarrow (b \rightarrow c)$ \Downarrow $x :: a, y :: b \Rightarrow f :: a \rightarrow b \rightarrow c$

(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 fpr pe = fpr pe difirent

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

egal explicit pe ? fct, f & ; cele ? fcti sunt egale doar in
daca sunt de tipul $a \rightarrow b$

$a \rightarrow b$: Haskell cau

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

\Rightarrow fpr $E_{\text{num}} = \text{Int} [1..10]$

(λ) fpr la m, sa ne stupi \Rightarrow map cau

cau el. sunt date cau \Rightarrow

\Rightarrow cau s-a vina de numerelor

map cau lista cau bazeaza = fold cau cau True

\Downarrow E_{num} in folo de $\text{Num} a$.

cau cau = cau fpr nu cau map.

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

Similaire fonction dans celle avec tuple. ou bien si on do 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

on a écrit au dessus => const flottant pex toutes les règles

des const flottant pex toutes les règles => fibo ou set des a, b & c flottant

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

=> fonction const de insertion n'importe quelle string.

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

com si on fait des symboles dans chars / si on puit 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. Ainsi de toutes sortes de logique ou produit de set

moi j'arrive pas à comprendre ça.

Il y a 3 pps: Com astuces, ou des erreurs

7. des erreurs (Tou, bine) = Tou ne donne bine

Tou ne a obtenu bine

{=>

=) LPOI: Logisch pred de und I:

hint: ∇ doesn't change (T_{in}, δ are)

1

deinen (neu) du uns gern

down (line, new)

ne puerum sibi sicut ostendit

$\text{stereo}(\text{Cine}, \text{Cine})$ ($\text{color}(\text{Cine}) \Rightarrow \text{Orion} \text{ Cine} \cdot \text{Orion} \text{ Cine}$)

$H \propto f^2$ $\Delta t \propto f^{-1}$, best case system for needed time \Rightarrow time decrease in needed time

\Rightarrow either $\text{Cinv} \cdot \text{Cout}$ $\leq \text{Cerr}$ or $\text{Cinv} \cdot \text{Cout} > \text{Cerr}$ \Rightarrow $\text{Cerr} \leq \text{Cinv} \cdot \text{Cout}$

\Rightarrow oricore Parsons . oricore Med . oricore (Parsons, P_{Med}) \Rightarrow sociale (Parsons, Med)

? not ostensible (Iou, bine)

dein priv. net

slug from R.A.

transform \mathcal{S} from chart \mathcal{S}'

Classe

- (1) au simpletif: Tastenre (Personne, Mod) V deonne (Personne, Mod)
- (2) au passif: T deonne (Inv, binv) Si fo c nesleufje into elante
- (3) compl. neg: astemp (Inv, binv)

(i) \rightarrow (ii): Tortfeasor (Person A, Med) \cup donee (Person B, Med)

astenos (T_{av} , b^{pre})

— — — — — Personne à l'ouest

Se n't grub substit. w

csterne(i, m, b, n) mestiso cu o cursă substituție ⇒ cei 2 literali vor fi ⇒

\Rightarrow Connection & decision (Personen, Med) under option flexibility \Rightarrow

\Rightarrow domain (Tau, bin) (u)

$$\text{under } \text{ (1)} + \text{ (2)}$$

dsome (ju, ðim)

7 donne (is, bine)

— — — — —

claudio vides

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 λ :

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

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

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

prim. act = $\lambda x. x$ = funcție identitate

= $\lambda x. x$

prim. funcție (de la primul λx)

⇒ $\lambda x. x$

corp ($x \ x$)

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

prim. act = 2

prim. funcț. ($z \ z$)

corp $\lambda x. x \rightarrow$ identity

2. (definie 10)

(

(lambda (x))

$y \Rightarrow \lambda x$ astfel ca să fie

(if ($> x$ 2))

prim. ste pe $s \Rightarrow x = s \Rightarrow$

(lambda (y))

$\Rightarrow (\text{if } (> s \ 2)) \Rightarrow$ J. oare să primă
doar o funcție de

(+y 1)

lambda (y)

(+x y)

(+) doar și λy astfel ca să

)

) și nu primește număr \Rightarrow

)

s

= funcție astfel λy 1)

5)

(+x y)

= funcție astfel λy 1)

)

s

= funcție astfel λy 1)

...

s

= funcție astfel λy 1)

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

(define const $L_1 L_2$)

(map

(lambda (x)

(cons x (filter

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

(L2)

))

map p^t o functie van const & een filter const

L_1

)

a) Geef de code!

(define b

NU. een filter in functie van

reste elem pos o -> result g !!

(lambda (f g L)

(if (null? L)

'()

(append

(if (f (g (car L)))

(list (g (car L)))

'()

)

(b) $f \circ g (\text{car } L)$

)
)
)
)

een filter de functie $f \circ g (x)$ onde x is de
resto =
van elke elem in de lijst, dan appliceer g op de rest

o filter



(map g (filter (lambda (x)

(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 $x = \text{keenly} \Rightarrow y = \text{listo}$

\Rightarrow \rightarrow add or tie? so depends on the found Eq?

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

 Es ist ein endlicher
 aber unbeschränkt

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

Tipuri de interfețe, rezervante, predicofe.

Haskell
+ type inference

friday ←

Fernan Basset, Prologue

g. Give in words, the measure of form, mean deviation, form

I'm new here

Institute (Cine), now as the Film Cine)

Rezultatele se potrivesc
cu ce spune R.

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

A few days ago
I was thinking
about

Musotō (Cine), Tresor del Cine, Forum

wrote (you, form)

finants (Pou) \Rightarrow 7 nus de (ion, form) \oplus

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: $\left\{ \begin{array}{l} \text{îmveță (Cine), mearde (Cine, Cine), ceeașă cine} \\ \text{Orice o fi cine doar învăță cel din urmă înseamnă Orice cine.} \\ \text{Orice persoană doar învăță atunci persoana nu mearde de cine} \end{array} \right.$
Dacă nu e cine să fie Măd, ci Măd = forme \Rightarrow

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

\Rightarrow Orice o fi persoană (aceea), doar îmveță persoană înseamnă forme.

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

(1) implicativ: îmveță (persoană) \rightarrow mearde (persoană, forme)

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

(3) condiție nevoie: îmveță (ion)

Prin urmare în cele două

(1) și (3): îmveță (persoană) \vee mearde de (persoană, forme)

îmveță (ion)

persoană \leftarrow ion

\neg mearde de (ion, forme) (4)

(2) și (4): mearde de (ion, forme)

\neg mearde de (ion, forme)

----- doar vîză \Rightarrow presupunere falsă
în îmveță.

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 ^{de} 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 \cup
 doar net în urmă. (la prima valoare comput.)

b) după înmulțire, sunt funcții ale trei state.
 (la prima valoare
 adunării)

c) ~~(define computation ((+ s) s))~~ \rightarrow State-combină
 (define computation (λ() (+ s s))) \downarrow State-combină
 (define (f x) (cons x ((computation)))) \downarrow includei funcțiile

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 Aplicações de férias em 2021 - TAEs x3

$x \cdot y, y, yx$

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

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

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

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

$y:a \text{ ou } y:(a \rightarrow b \rightarrow c) \rightarrow b$ \downarrow com este y é unico e pode ser usado para
 \Downarrow $y = x \cdot y(yx) : y:a \rightarrow b \rightarrow c$,
 $y:y:(a \rightarrow b \rightarrow c) \rightarrow b$
 \Downarrow $y = x \cdot y(yx) : y:a \rightarrow b \rightarrow c$

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

ia pl (1, 2) se i passar do loop, depõi k=2 passa
ciclo mais intolável, depõi ... ou seja
não é possível

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

ia tenta oito desdobro \Rightarrow passa muitas possibilidades de f.

poss: $\boxed{1, 2, 3, 4} \rightarrow \text{ia1}$

6 class MyClass & where

$f:a \rightarrow a \rightarrow \emptyset$ possivel instanciar f? Ord, 25 and obstante a dupla inst. f.

Seu instance MyClass f] where

f $\underline{\text{used}}$

7. Nós temos 2 classes de números \Rightarrow

7. $\text{Cine}, \text{Zoo}(Cine)$ \Rightarrow $\text{Zoo}(Cine), \text{ZooAnimais}(Cine)$
 $\text{ZooAnimais}(Cine), \text{Animal}(Cine)$

\Downarrow
 $\text{Zoo}(Animal), \text{ZooAnimais}(Animal)$

Vamos definir o tipo animal em ZooAnimals, que é
muito
(vai ser usado often, static)

8. $P(R, S) :- \text{member}(X, R) \rightarrow \text{do something}$

Findall (y , member (y, f), $y \Leftarrow x$) \top), !, $\sqsubseteq(x, T(s))$.

$\varrho(x, A(\lceil x(A) \rceil), \varrho(x, \lceil A(B) \rceil, \lceil A(C) \rceil)) \vdash \varrho(x, B, C)$

↳ \Rightarrow $\log_{10} 5 \approx 0.699$ und 5^{th} Stelle?

to prevent other saline duplicates from

Die Regeln

\leftarrow onto fisi pe cei care sunt membrii

der dt.-de x 47; eine Inter-

lifts
new.

$$P((C_1, 2, 3, 4], S)$$

$$S = \{ \} ; \quad S = \{ 1, 2, 3, 4 \}$$

$$S = \{2, 3, 1, 0\}$$

↓ Desperimento,

$$S = [2, 3, 13, 4]$$

out to be next complicat

findall (ce, fct, liste)

Sì, io prendo possibilità di creare o bloccare ip., i flussi sono =)

$$\Rightarrow \varrho(x, A, [x]A) \in \{ \infty, [A\{B\}, CA\{C\}] : -q(x, B, C), \infty \}$$

\Rightarrow stiels nette- oder unnette \Rightarrow

q \downarrow (x, T, s)
playlist title voice

$$L: [1, 2, 3, 4] ; \quad 2: C_{L(1, 3, 4)} ; \quad 3: [C_{L(3, 2, 4)}, C_{L(3, 4, 1)}]$$

final $x \downarrow A_1[x(A)]$.
 al dices
 $a(x, [A_1 B], [A_1 C])$, open face & scroll down &
 then
 insert into ppf

g. σ (Private, cost)

```
front(X, F) :- p(point, X), !, findall(Y, (c(point, Y), Y != X), F)
```

↓ stop. ↪ first presel proto, cont'd later fric.

def de X corigint și părințele

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

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

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

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

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

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

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

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

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

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

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
prote tip

2019-B

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

am 3 β-redopti'. Tidspunktet med sin deopto =>

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

für α d. konv. $\Rightarrow (y \ (\lambda z. \underline{\lambda x. x} \ y)) \Rightarrow (y \ \lambda x. x)$ nach spätbind. Gesl.

Alle ohne Sport für die kein Zettel wop => Step 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 ~~dry~~
semi-dry)

b) (define computation (delay (equal? 55)))

```
(define(f x) (mof(>x s) (force computation)))
```

cf schéma de moi seul de septembre à droite

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

(filter (combs (L))

(if (\leq (apply + l) (apply * l)))

#4

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

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

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

\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 următoarele liste

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

când-nu (cond), să luăm-nu (cond) \rightarrow să luăm cond

=> există o funcție

când nu nu e nici un

cănd

când-nu (cond) \Rightarrow nu când-nu (cond)

\Downarrow

$\exists x. \text{când-nu}(x) \Rightarrow \text{nu când-nu}(x) \Rightarrow$ se poate da implementație ($P \Rightarrow Q \equiv \neg P \vee Q$)

$\rightarrow \forall X. \exists \text{listii_nr}(x) \wedge \text{listi_nr}(x)$

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

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

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 și $p([B|D])$ deci vom să o să urmăreștem pe A .

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

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

suntem sănăti, vrem să scriem simplu, fără să scriem funcții auxiliare,
nu putem să scriem:

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

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

$K = \text{Hash } a$

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

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

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

18-A

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

deci = \rightarrow decesz; \cup combinares; \sqsubseteq es un α -equivalencia; \vdash = es una deducción; \vdash = es una deducción
 Punto final Punto final Punto general Punto general
 α es una deducción
 β es una deducción
 $\alpha \vdash \beta$

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

$$2. (\text{let } ((a 1) \\ (b 2)))$$

(let* ((a 1)
 (b 2)))

(let ((b 3)
 (c (+ b 2))))

(let* ((b 3)
 (c (+ b 2))))

(+ a b c))

||

$c = 4$ $a = 1$
 $b = 3$
 $+ 1 3 4 = 8$

\rightarrow aquí se evalúan
 directamente los
 valores numéricos
 que son los resultados
 de las expresiones

$$c (+ 2 2) = 4 \\ b = 3, a = 1 \\ + 1 3 4 = 8$$

3. Zie:: [a] \rightarrow [b] \rightarrow [(a b)]

[$(\text{define } (\text{cons } f \ \lambda_1 \ \lambda_2)$
 $\quad (\text{map } (\text{cons } (\text{map } f \ \lambda_1) (\text{map } f \ \lambda_2))))]$

f u b g u m o i f u d g R \Rightarrow $\Rightarrow (\text{define } (\text{cons } \lambda_1 \ \lambda_2)$ $\quad (\text{map } \text{cons } \lambda_1 \ \lambda_2))$

$$4. f x y z = x y . z = z \underbrace{\text{cons}}_{\text{cons}} \underline{w} \underline{x} y$$

X::a deu xy e fct & e función

y::b $\Rightarrow x::b \rightarrow e \rightsquigarrow n = a$ $f::(b \rightarrow e \rightsquigarrow n) \rightarrow b \rightarrow (d \rightarrow e) \rightarrow d \rightarrow b.$

z::c = d $\rightarrow e$ $\Rightarrow z::d \rightarrow e$

x y :: f $\rightarrow n$ $\Rightarrow d \rightarrow e / \rightarrow e \rightsquigarrow n$

d o g $\circ (x y) \Rightarrow e = g$ $\Rightarrow (x y) (e \rightsquigarrow \dots)$

5. instance Show(Emme, Neue, ShowD) \Rightarrow Show(a \sqsupset b) (where
 show f = " " map f [1..10] " "
 -- f is applied to number, i.e. bis is a show procedure.
 si liste
 of nz.
 of f.
 show f = composition (show o f) [1..10]

Si f si g given so we can show.
 Si o f si o g
 f & g [1..10]
 so we can
 show f & g
 2 3 4 5 6 7 8 9 10 11

6. [map concat (apply (+)/ZipWith [1..5] [1..])] ?
 [take S [m / m \leftarrow [m..], Need m m = 0] [m \leftarrow [1..]]]
 ↓ ↓ listes
 lists & little lists
 ↓ index ends to op
 multiply m
 m create
 m create

7. "I am este om", "Onci om are a bicicleta" \Rightarrow "om are bicicleta
 son ion este bagat". om(x), areBicicleta(x), bagat(x).
 calegi x

tois se ne gandim mai simplu: $\exists x. \text{om}(x)$

$\exists x. \text{areBicicleta}(x)$.

Onci om are bicicleta.) |

$\exists x. \text{bagat}(x)$

$\forall x. \text{om}(x) \Rightarrow \text{areBicicleta}(x) \Rightarrow \forall x. \text{om}(x) \vee \text{areBicicleta}(x) \Rightarrow$
 $\Rightarrow \exists x. \text{om}(x) \vee \text{areBicicleta}(x) \vdash$

{ om(iu) } (1) ion este om = om(iu)

{ $\neg \text{om}(x) \vee \text{areBicicleta}(x)$ } (2)

{ $\neg \text{areBicicleta}(iu)$ } (3)

{ $\neg \text{bagat}(iu)$ }

(1): (2) \rightarrow $\text{areBicicleta}(iu)$ (5)

(5) w (3) $\rightarrow \Gamma \vdash \text{S} \uparrow \text{of } \vdash$

ion este
 areBicicleta(iu) v bagat(iu)
 ||
 $\neg (\text{areBicicleta}(iu) \vee \text{bagat}(iu))$
 ||
 $\neg \text{areBicicleta}(iu) \wedge \neg \text{bagat}(iu)$.
 De non sequitur

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 (+ a2))) R=1+2=3 (verb)

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

Reactive Centrizable Mediator

deposits on fast fronts

Se fortesc uol semper in a.

Lambda ($\lambda b c$)

ost es in dieser fct,

is one of several;

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

euor.

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: $x \mapsto s_0 f(x) - f(0)$, lo fel so $y \in \cos S$ für $\cos y = 0$ \Rightarrow

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

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

$z \vdash c$

$\cup_{x::h \rightarrow e}$

$$f: (h \rightarrow e) \rightarrow (c \rightarrow g \rightarrow u) \xrightarrow{c} \underset{t}{\underset{\downarrow}{\cup}} \rightarrow g \rightarrow e$$

七

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

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“.
 $\text{toron}(x), \text{areSapă}(x), \text{destept}(x)$.

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

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

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

$\neg P \Rightarrow Q \Leftrightarrow \neg P \vee Q$

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

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

$$\Downarrow$$

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

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

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

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

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

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

(5) cu (3) $\Rightarrow \Gamma \boxed{\text{STOP}}$.

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 car 1 \rightarrow resto p'm λ
 \rightarrow factorial.
Gr n (f (- n 1)))
)) \rightarrow n div λ)

(f 5)) $1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120?$
↳ resto e ng del λ

④ (define (calcul x y z) ($\#t$ x y z)) \rightarrow "meidare funktions"

(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

(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)

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

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

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

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

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

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

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

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

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

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

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

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

neg conclusion
dan:

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

(1) und (2): $x \leftarrow ex$, \Rightarrow $\text{eliminieren } C_1 \text{ bei } x \Rightarrow \{\text{einfachere Form}\}$

(1) see (3): $x \in \text{euclidean cluster} = \{l\}$
 $y \in \text{euclidean}$

(3) see (c): $y \in \text{Per}_\text{max}$, so eliminate clause $\Rightarrow \Gamma \vdash \neg y$ STOP

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

Case & Report no. R-11666 / case & effects splic. bni p p44.2

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

$f = \text{resultat}.$

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

A few small vicks do fine.

$p(A, [E|T] \models I, R) := e(A, T, R).$

T & R send liste core Tweep ce E sj mør-k mit pA.

dois é um e logo se nenhô, il logo se primeiro.

L_2 și R fac parte din ciclelori element și nu conțin pe L_1

$\rho(A, [E|T])$ one to one $\Rightarrow \rho([E|R])$

↓ deci schengō lo Tpl A, sas neni bine 2is, schengō lo

$(E_{IT})_{p \in A_1}$ los final. $\Rightarrow L_2 \leftrightarrow L_1$.

2) $(\text{Def } ((f \circ \lambda(x_4)) \ x)) \circ (f \circ (1/2 \circ)))$

(let $c \in f(\lambda(x_4))$

)) $x)$ $\not\models \text{este}(\lambda | x y)$

(f $S(1/2^+)$)

) "Attempts

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

4) $x=2/0$, $y=x$. \rightarrow es ein S der stetig ist \Leftrightarrow muß stetig sein,

x is \perp to y , y is x , $x = y \Rightarrow$ no \perp relation

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 (mug 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 e nevoie, să redenumăceze în funcție de exemplu \Rightarrow 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 $(y x) \Rightarrow x:: d \rightarrow e$.

y este $a \rightarrow f$, ce se aplică rest $x \Rightarrow y:: f \rightarrow h$

x este $(y x) \Rightarrow x$ primește ca rest $y \Rightarrow x:: h \Rightarrow d = h$.

y este $x \Rightarrow y$ primește ca 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 valoare cleu y)

$y:: g \rightarrow h$ $e = c$ (f este valoare cleu x.)

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

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

y este o bază pentru x divizor $\Rightarrow \underline{d \rightarrow c}$.

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

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

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.

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

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

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

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 ră descalcă, deci nu trăiește

cât despică E, while E nu poate să își λ \Rightarrow punc f și λ ... pentru devenire E

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

\rightarrow l e un fel de compus.

$f :: a$

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

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 spălat
 $f :: c \rightarrow d$
 $(f x) :: c$
 $d = b$
 $f :: e \rightarrow g$
 $x :: e = a$
 $g = c$
 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

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

trigo los rindores

[all_] → Prolog =>))

G. Eliminate other duplicates:

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

-- cot de bosque

clear [] = []

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

$$(\text{exto } (x : xs)) = x : [y \mid y \leftarrow \text{cons } xs, y \neq x]$$

-- sou $\text{cens}(x : xs) = x : \text{cens}(\text{filter}(\lambda = x)xs)$

-- so $\text{Res} \ell = \text{head } \ell : \text{cons}(\text{filter}(\text{rest} . (\text{rest } \ell))) \& \text{to}^{\ell} \ell$

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

↓ 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 impone en el 2-los

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

g total = g ...
↓ Naturals al 3-les el. skivs = 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 pas tout de suite (force f))F attend un argument, E est celui qui primeste l', mais exécute l' argument
fournit de F apelé avec l' argument \Rightarrow primeste liste a la \Rightarrow devine
spécifiques à cette. Mais cette spécificité \Rightarrow dans E il appelle à son tour 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: $\neg \exists \text{maiMare}(\text{elefant}, \text{socicel})$

$$\Rightarrow (1) \text{ cu } (2): x \leftarrow \text{elefant}, y \leftarrow \text{leu}$$

\Rightarrow rezolvare maiMare si

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

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

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

$$(3) \text{ cu } (5) \quad z \leftarrow \text{socicel} \& \text{rezolvare maiMare si } \{ \exists \text{maiMare}(\text{elefant}, \text{socicel}), z \} (8)$$

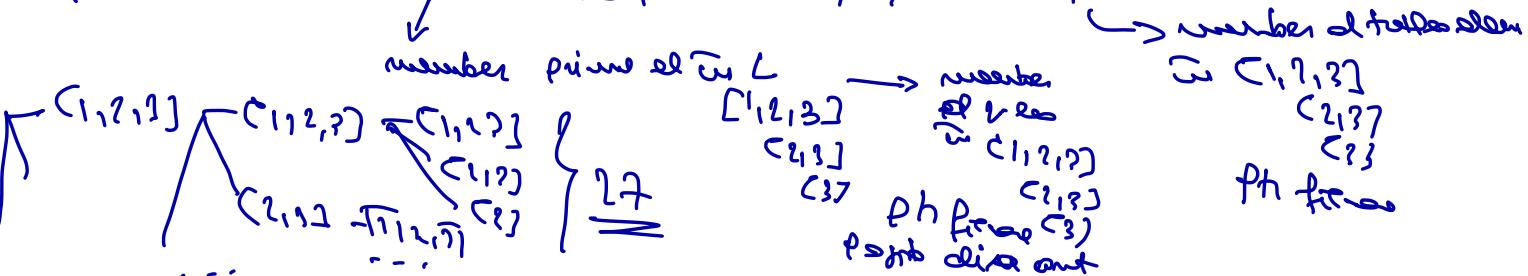
$$(8) \text{ cu } (4): \Gamma \perp \text{STOP} \Rightarrow \text{maiMare}(\text{elefant}, \text{socicel})$$

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

$$\neq(+X)$$

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

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



WU-A

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

2. (define (example2)
 (filter (lambda (l) (member (length l) l)) l2))

3. (let* [(x 1) x=1
 (y 2) y=2
 (f (delay (lambda (y) (+ x y))))
]
 x = k + y = 1 + y old y
 (let [(x 5)] => x = 5 old x
 ((force f) x) let λ
)
 \downarrow $1 + y, y \in \text{delay power, adding } x$
 \downarrow as x den let int =>
)

4. const :: [a] → [b] → [(a,b)]

$+5 \geq$

const [] [] = []

const A B = const (map (λ x → map (λ y → (x,y)) y)) x) LIST COMPREHENSIONS.

const A B = [(x,y) | x ∈ A, y ∈ B]

5. flex Peteri 2:

const = 1: map (*2) (flex

= [2^x | x ∈ Co..])

= [2 * i | i ← iterate (*2) 1]

= iterate (*2) 1

= (gen2 i | i ∈ Co..])

where

gen2 :: Integer → Integer

gen2 0 = 1

gen2 m = 2 * gen2(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,

adică un triplet cu

diferențe de tip și diferențe

\approx x este variabilă,
 $y = z \Rightarrow y$ 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 a)`

8. „Cine tocă și nu înțeleapt de căci 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, Societate, sau el îngrijit, NMK este membru.”

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

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

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

Societate, sau el îngrijit $\Rightarrow \text{membru}(\text{Societate})$; condusă: $\text{membru}(\text{Societate})$;

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)

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

nested send f \rightarrow f =)

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&2 ; due di 1, ff 2 è alt ff.

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

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

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

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

$$f :: a \rightarrow b$$

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

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

$$y :: d$$

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

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

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

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

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

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

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

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

$$y \in d$$

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(Persona)`; `cunoastePredecesor(Persona, 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 $x < x + j$ și $p = x + j - 1$ și $t = \text{moment } j$.

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

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

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

$filtter([], T, LF)$ $filtter([H T], T, LF) :-$	$:- \text{filter}(T, T, LF)$, $\text{findAll}(X, \text{member}(X, L), LF)$ $:- \text{findAll}(X, \text{member}(X, L), X > T), LF$
---	---

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. $\underline{\lambda z. \lambda y. \lambda x. (y \neq z) y} \lambda y. y \rightarrow \lambda z. \lambda w. \lambda x. (w \neq z) y \lambda y. y \rightarrow \beta$
deci' esto n-ao nico comu = 1)

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

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' en int capa de let-alue;

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

no p'ect spars e' let source

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

da difes obvianza \Rightarrow Nu e nico dif.

3. (define (f L)

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

(filterL (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 \Rightarrow Choose (1), (2) Pre cas mezo

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

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

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

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 modul,

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

⇒ introducere 2

↓

nu e nicio diferență pe cele două lucru,

totuși în final săptămâna pe o lăză,

pe baza - qdă la 1 și fără semnificativ,

3. (define (f L)

(car (reverse (rest L))))

; doar ca să scăpați el cu să se relateze nr. pozitiv

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

doar nu
poate fi

5. Def cl's `Fronted`, tip select & exist posn v def o let go each (viral & o also can each wth elem, pt & lists)

Ques `Fronted` t where `frontend`:: $t \sim v$; `backend`:: $t \sim v$
instance `Fronted []` where

`frontend = head`

`backend = head, reverse,`

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

`spine-mult(Iin)`, `trace(Mout)`; `spine-moi.putin(Iin, Pout)`.

$\vdash x.spine\text{-mult}(x) \wedge \exists y.trace(y) \Rightarrow spine\text{-moi-putin}(x, y)$.

$\nexists x.\nexists spine\text{-mult}(x) \vee \nexists y.\nexists trace(y) \vee spine\text{-moi-putin}(x, y)$.

$\exists x.\nexists spine\text{-mult}(x) \vee \exists y.\exists trace(y) \vee spine\text{-moi-putin}(x, y)$.

`spine-moi-putin(Iin, Pout)` $\Rightarrow \nexists spine\text{-moi-putin}(I_{in}, P_{out})$

P_p R A \hookrightarrow \Rightarrow

close 2: {`spine-mult(Pin)`} (1)

{`trace(more)`} (2)

{ $\nexists spine\text{-mult}(x), \exists trace(y), spine\text{-moi-putin}(x, y)$ } (3)

{ $\nexists spine\text{-moi-putin}(i_{in}, more)$ } (4)

(1) \cup (3) $\Rightarrow x \leftarrow i_{in}, disj_{spine} spine-mult $\Rightarrow \{ \exists trace(y), spine\text{-moi-putin}(i_{in}, y) \} \models (5)$$

(2) \cup (5) $\Rightarrow y \leftarrow more$, disj_{trace} trace $\Rightarrow \{ spine\text{-moi-putin}(i_{in}, more) \}$

(6) \cup (4) $\Rightarrow \Gamma \vdash S \top P \models spine\text{-moi-putin}(i_{in}, more)$. (6)

7. $x(L, A, B, N)$ defn & o link n defn $> A, < B$.

$x(+L, +A, +B, -N)$

$x([], A, B, 0)$

$x(L, A, B, N) :- \text{findall}(X, \text{member}(X, L), X > A, X < B, S), \text{length}(S, N).$

1017-B

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

2. (define a 2)

$\xrightarrow{\text{let } ((a 1) \Rightarrow a = 1)}$

$(b \ a) \xrightarrow{= b = a = 2}$

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

$\downarrow \text{with let for } a$

$\downarrow \text{new let to replace } a$

$\downarrow \text{copr let, } a \text{ is prime, no define.}$

$\downarrow \text{copr let to replace } a,$
 $b = \text{old } a;$

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

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

$(b \ a) \xrightarrow{b = a' = 1} \text{if } a = 2$

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

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.

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

$(x, y, z) \Rightarrow x :: j, y :: j, z :: j$ $\xrightarrow{\quad} \text{def } a \text{ as list}$

b = [j]

$\xrightarrow{\quad} j \equiv j \rightarrow j \rightarrow j \rightarrow (j \rightarrow \text{Bool})$

$\xrightarrow{\quad} j \equiv j$

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

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

5. close w/o fl doto Point a = MakePoint a &

close Ended t where frontEnd::t v->v; backEnd::t v->v.

instance Ended Point where

$$\text{frontEnd}(\text{MakePoint } x) = x$$

$$\text{backEnd}(\text{MakePoint } x) = x$$

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

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

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

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

$\exists x(L, A, B, N) : - \text{findAll}(x, \text{elements}(x, L), x \subset A, x \overset{>}{=} B, S),$
 $\text{length}(S, N).$

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

$f : a \rightarrow b$

$x : c, y : d, z : e, g : h$ $f : i \rightarrow j \rightarrow (k \rightarrow (l \rightarrow j)) \rightarrow [j]$,
 $\text{map} : (i \rightarrow j)^\circ [k] \rightarrow g^\circ$ \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.) +
 +

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

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

$\rightarrow \exists x \exists y. \neg \text{begat}(x) \vee \text{brother}(y) \vee \neg \text{mother}(x) \vee \text{friends}(y).$

Closure: { λ bill. { λ song(bill), λ move(bill), λ fly(bill)} (1)}

{ λ best(bill) (2)}

{ λ son(bill) (3)}

{ λ move(bill) (4)}

{ λ 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. $x(L, M) = \text{next_list}(L, M)$

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

CLASE IN HASKELL

```

length_ [] = 0
length_ (_:xs) = 1 + length_ xs
length :: [a] -> Int
map :: (a -> b) -> [a] -> [b]

elem _ []      = False
elem x (y:ys) = x == y || elem x ys
elem :: Eq a => a -> [a] -> Bool

Clase
class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool

data Person = Person {name :: String, cnp :: Integer}
instance Eq Person where
  Person name1 cnp1 == Person name2 cnp2 = name1 == name2 && cnp1
  == cnp2
  p1 /= p2 = not (p1 == p2)
data BST a = Empty | Node a (BST a) (BST a)
instance Eq a => Eq (BST a) where
  Empty == Empty = True
  Node info1 l1 r1 == Node info2 l2 r2 = info1 == info2 && l1 ==
  l2 && r1 == r2
  == = False
  t1 /= t2 = not (t1 == t2)

class (Eq a) => Ord a where
  compare           :: a -> a -> Ordering
  (<), (<=), (>), (>=) :: a -> a -> Bool
  max, min          :: a -> a -> a

  compare x y = if x == y then EQ
                 else if x <= y then LT
                 else GT

  x < y = case compare x y of { LT -> True; _ -> False }
  x <= y = case compare x y of { GT -> False; _ -> True }
  x > y = case compare x y of { GT -> True; _ -> False }
  x >= y = case compare x y of { LT -> False; _ -> True }

  max x y = if x <= y then y else x
  min x y = if x <= y then x else y

class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool
  a /= b = not (a == b)

```

```

instance Eq Person where
    Person name1 cnp1 == Person name2 cnp2 =
        name1 == name2 && cnp1 == cnp2

```

```

class Eq a where
    (==), (/=)           :: a -> a -> Bool
    x /= y               = not (x == y)
    x == y               = not (x /= y)

```

Clase predefinite

Ord - pentru tipuri care pot fi ordonate - definește funcții precum <, >, <=, etc.

Show - pentru tipuri care pot fi reprezentate ca String-uri - principala funcție este show. Această funcție este folosită și de consola GHCi atunci când afișează rezultatele.

Deriving

```
data BST a = Empty | Node a (BST a) (BST a) deriving (Eq)
```

```
treeMap :: (a -> b) -> BST a -> BST b
```

```
treeMap f Empty = Empty
```

```
treeMap f (BST info l r) = BST (f info) (treeMap f l) (treeMap f r)
```

```
data Maybe a = Nothing | Just a
```

```
maybeMap :: (a -> b) -> Maybe a -> Maybe b
```

```
maybeMap f Nothing = Nothing
```

```
maybeMap f (Just x) = Just (f x)
```

```
class Functor container where
```

```
    fmap :: (a -> b) -> container a -> container b
```

```
instance Functor BST where
```

```
    fmap f Empty = Empty
```

```
    fmap f (Node a l r) = Node (f a) (fmap f l) (fmap f r)
```

```
{-
```

Clasa Container reprezintă o clasă folosită pentru enumerarea elementelor unei structuri de date (listă, arbore, graf, etc.)

```
-}
```

```
class Container t where
```

```
    contents :: t a -> [a]
```

```
{-
```

Clasa Invertible reprezintă o clasă folosită pentru inversarea ordinii de apariție a elementelor unei structuri de date (listă, arbore, etc.)

```
-}
```

```
class Invertible a where
```

```
    invert :: a -> a
```

```

{- BST (Binary Search Tree) reprezintă un tip de date ce modelează
   un arbore binar de căutare. Ați definit funcționalitățile
   acestui
   tip de date în cadrul laboratorului 8.

   Funcționalitățile sunt deja definite în cadrul acestui
   laborator.
-}

data BST a = BSTNod
  { vl :: a
  , lt :: BST a
  , rt :: BST a
  } | BSTNil

insertElem :: (Ord a, Eq a) => BST a -> a -> BST a
insertElem BSTNil elem = BSTNod elem BSTNil BSTNil
insertElem root@(BSTNod value left right) elem
  | value == elem = root
  | value < elem = BSTNod value left (insertElem right elem)
  | value > elem = BSTNod value (insertElem left elem) right

findElem :: (Ord a, Eq a) => BST a -> a -> Maybe a
findElem BSTNil _ = Nothing
findElem (BSTNod value left right) elem
  | value == elem = Just value
  | value < elem = findElem right elem
  | value > elem = findElem left elem

size :: BST a -> Int
size BSTNil = 0
size (BSTNod _ left right) = 1 + size left + size right

height :: BST a -> Int
height BSTNil = 0
height (BSTNod elem left right) = 1 + max (height left) (height
right)

inorder :: BST a -> [a]
inorder BSTNil = []
inorder (BSTNod elem left right) = inorder left ++ [elem] ++ inorder
right

{- Arbore folosit pentru testare
-}

root = foldl insertElem BSTNil [7, 4, 12, 2, 3, 1, 10, 15, 8]

{- 1. Instantiați Eq pentru tipul de date BST, prin care se

```

```
    verifică dacă doi arbori de acoperire sunt identici.  
- }  
  
instance Eq a => Eq (BST a) where  
    (==) BSTNil BSTNil = True  
    (==) (BSTNod e1 l1 r1) (BSTNod e2 l2 r2) = e1 == e2 && l1 == l2  
&& r1 == r2  
    (==) _ _ = False
```

```
{ -  
2. Instantiați Show pentru tipul de date BST.  
Fiecare nivel de adâncime în arbore va fi reprezentat de un număr  
corespunzător de tab-uri. De exemplu pentru nivelul 2 de adâncime  
se vor adăuga două tab-uri.
```

```
Fiecare element din arbore va avea linia sa, adică câte un  
element  
din arbore pe o linie.
```

```
- }
```

```
printLevel :: Show a => Char -> Int -> BST a -> [Char]  
printLevel _ BSTNil = ""  
printLevel tab level (BSTNod root left right) = replicate level tab  
++ show root ++ "\n"  
++ printLevel tab (level + 1) left  
++ printLevel tab (level + 1) right  
  
instance Show a => Show (BST a) where  
    show BSTNil = ""  
    show node@BSTNod{} = printLevel '\t' 0 node
```

```
{ -  
3. Instantiați Ord pentru tipul de date BST.  
Criteriul de comparare a doi arbori va fi după înălțimea lor  
(funcția height).
```

```
Trebuie implementată funcția (<=)  
- }
```

```
instance Ord a => Ord (BST a) where  
    (<=) t1 t2 = height t1 <= height t2
```

```
{ -  
4. Instantiați Invertible pentru tipul de date BST.  
Funcția invert, în acest caz, va inversa ordinea subarborilor.  
- }
```

```
instance Invertible (BST a) where  
    invert BSTNil = BSTNil
```

```
    invert (BSTNod a left right) = BSTNod a (invert right) (invert left)
```

```
{ -
```

5. Instantiați Functor pentru tipul de date BST.

Funcția fmap este similară funcției map, prin care se aplică o funcție f tuturor elementelor din structură.

```
- }
```

```
instance Functor BST where
```

```
    fmap f BSTNil = BSTNil
```

```
    fmap f (BSTNod a left right) = BSTNod (f a) (fmap f left) (fmap f right)
```

```
{ -
```

6. Instantiați Foldable pentru tipul de date BST.

Funcția foldr are aceeași funcționalitate atunci când ea este aplicată pe liste.

Nu trebuie să implementați și foldl, clasa Foldable nu acoperă și această funcție.

```
- }
```

```
instance Foldable BST where
```

```
    foldr f acc BSTNil = acc
```

```
    foldr f acc (BSTNod value left right) = foldr f (f value newAcc) left
```

```
        where newAcc = foldr f acc right
```

```
{ -
```

7. Instantiați Container pentru tipul de date BST.

Pentru implementarea funcției contents o să folosiți funcția foldr, implementată la exercițiul anterior.

```
- }
```

```
instance Container BST where
```

```
    contents tree = foldr (:) [] tree
```

```
{ -
```

8. Implementați sizeFold, care calculează numărul de elemente din cadrul unui arbore binar de căutare (există deja funcția size, care face același lucru, definită mai sus).

```
- }
```

```
sizeFold :: BST a -> Int
```

```
sizeFold tree = foldr (\_ acc -> acc + 1) 0 tree
```


Close , tipuri în spirit Haskell

class Eq a where

(==) :: a → a → Bool

instance Eq Integer where

$$x == y = x \text{ `integerEq' } y.$$

instance (Eq a) ⇒ Eq (Tree a) where

leaf a == leaf b = a == b

(Branch l1 r1) == (Branch l2 r2) = (l1 == l2) ∧ (r1 == r2)

— == — = False

class (Eq a) ⇒ Ord a where

(<), (<=), (>=), (>) :: a → a → Bool

max, min :: a → a

$$x < y = x <= y \wedge x \neq y.$$

instance Num a where ...

data Point = Pt { pointX, pointY :: Float }.

pointX :: Point → Float -- dato do x

-- do fel și bo pointY

absPoint :: Point → Float -- funție.

absPoint (Pt { pointX = x, pointY = y }) = sqrt (x^2 + y^2)

Close standard Haskell

{ data Ordering = EQ | LT | GT

{ compare :: Ord a ⇒ a → a → Ordering

show :: (Show a) ⇒ a → String

instance Show a ⇒ Show (Tree a) where

ShowFor x = showTree x. →

instance Show a => Show (Tree a) where

Show t = showTree

ShowTree :: (Show a) => Tree a -> Show

shows (Leaf x) = shows x

showsTree (Branch l r) = l :) . showsTree l . ('!' :) . showsTree r . ('!' :)

data Tree a = Leaf a | Branch (Tree a) (Tree a) deriving Eq

instance Ord a => Ord (Tree a) where

(Leaf -) <= (Branch -) = True

(Leaf x) <= (Leaf y) = x <= y

(Branch -) <= (Leaf -) = False

(Branch l r) <= (Branch l' r') = l == l' && r <= r' || r <= l'

Semesteri funkcii folog

member(X, L)

append(L_1, L_2, L_R) sau append([L], L)

prefix ($P_{\text{last}}, \text{whole}$)

select(X, L_1, L_2) $L_1 \setminus X = L_2$ sau. select(X, X_L, Y, Y_L): select($B_1, [a, b, c, b],$
 $[c, d, e, f], X$).

nextto(X, Y, L) (X & Y de la L)

delete($L, @E, L_2$)

nth0(M, L, E, R) (primul element din L cu E , $R = \text{restul}$)

last, same_length, reverse, flatten (din [L] fac lista ca toate

subseq, max_member, min_member, sum_list, max_list, min_list,
intersection, union,

pair($X, Y, [X, Y]$).

split(L, L_1, L_2):- neplit(P_0), L_1, L_2, P_0 .

findall(P, P, L) lista de fapturi satisabile P .

setof(X, P, L) sa fie o lista de fapturi care duc la L

bagof(X, P, L) sa fie o lista de fapturi care duc la L

random(L, H, X), between(C, H, X), succ(X, Y), obs(X), max(X, Y)

rand(X), trunc(X), round(X), ceiling(X), sqrt(X)

% Exista last(?Elem, ?List) my_last(X,[X]). my_last(X,[_ L]) :- my_last(X,L).	swap([X, Y T], [Y, X T]):-X>Y. swap([Z T], [Z TT]):-swap(T, TT).	busort(L,S):-swap(L,LS),!,busort(LS, S). busort(S,S).
last_but_one(X,[X,_]). last_but_one(X,[_,Y Ys]) :- last_but_one(X,[Y Ys]).	cmmdc(X,0,X) :- X > 0. cmmdc(X,Y,G) :- Y > 0, Z is X mod Y, cmmdc(Y,Z,G).	tree(1,t(a,t(b,t(d,nil,nil),t(e,nil,nil)),t(c,nil,t(f,t(g,nil,nil),nil))))). tree(2,t(a,nil,nil)). tree(3,nil).
element_at(X,[X _],1). element_at(X,[_ L],K) :- K > 1, K1 is K - 1, element_at(X,L,K1).	istree(nil). istree(t(_,_R)) :- istree(L), istree(R).	symmetric(nil). symmetric(t(_,_R)) :- mirror(L,R).
my_length([],0). my_length([_ L],N) :- my_length(L,N1), N is N1 + 1.	min([X], X):-!. min([P R], P):-min(R,X), X > P, !. min([P R],X):-min(R,X), X = = P.	mirror(nil,nil). mirror(t(_,_L1,R1),t(_,_L2,R2)) :- mirror(L1,R2), mirror(R1,L2).
my_reverse(L1,L2) :- my_rev(L1,L2,[]). my_rev([],L2,L2) :- !. my_rev([X Xs],L2,Acc) :- my_rev(Xs,L2,[X Acc]).	split(L,O,[],L). split([X Xs],N,[X Ys],Zs) :- N > 0, N1 is N - 1, split(Xs,N1,Ys,Zs).	remove_at(X,[X Xs],1,Xs). remove_at(X,[Y Xs],K,[Y Ys]) :- K > 1, K1 is K - 1, remove_at(X,Xs,K1,Ys).
is_palindrome(L) :- reverse(L,L).	insert_at(X,L,K,R) :- remove_at(X,R,K,L).	min-sort([], [])):-!. min-sort(L, [M LS]):-min(L, M), select(M, L, LM), min-sort(LM, LS).
my_flatten(X,[X]) :- \+ is_list(X). my_flatten([],[]). my_flatten([X Xs],Zs) :- my_flatten(X,Y), my_flatten(Xs,Ys), append(Y,Ys,Zs).	drop(L1,N,L2) :- drop(L1,N,L2,N). drop([],_,[],_). drop([_ Xs],N,Ys,1) :- drop(Xs,N,Ys,N). drop([X Xs],N,[X Ys],K) :- K > 1, K1 is K - 1, drop(Xs,N,Ys,K1).	slice([X _],1,1,[X]). slice([X Xs],1,K,[X Ys]) :- K > 1, K1 is K - 1, slice(Xs,1,K1,Ys). slice([_ Xs],I,K,Ys) :- I > 1, I1 is I - 1, K1 is K - 1, slice(Xs,I1,K1,Ys).
compress([],[]). compress([X],[X]). compress([X,X Xs],Zs) :- compress([X Xs],Zs). compress([X,Y Ys],[X Zs]) :- X \= Y, compress([Y Ys],Zs).	decode([],[]). decode([X Ys],[X Zs]) :- \+ is_list(X), decode(Ys,Zs). decode([[1,X] Ys],[X Zs]) :- decode(Ys,Zs). decode([[N,X] Ys],[X Zs]) :- N > 1, N1 is N - 1, decode([[N1,X] Ys],Zs).	count(X,[],[],1,X). count(X,[],[],N,[N,X]) :- N > 1. count(X,[Y Ys],[Y Ys],1,X) :- X \= Y. count(X,[Y Ys],[Y Ys],N,[N,X]) :- N > 1, X \= Y. count(X,[X Xs],Ys,K,T) :- K1 is K + 1, count(X,Xs,Ys,K1,T).
transfer(X,[],[],[X]). transfer(X,[Y Ys],[Y Ys],[X]) :- X \= Y. transfer(X,[X Xs],Ys,[X Zs]) :- transfer(X,Xs,Ys,Zs).	dupli([],[]). dupli([X Xs],[X,X Ys]) :- dupli(Xs,Ys).	encode_direct([],[]). encode_direct([X Xs],[Z Zs]) :- count(X,Xs,Ys,1,Z), encode_direct(Ys,Zs).
pack([],[]). pack([X Xs],[Z Zs]) :- transfer(X,Xs,Ys,Z), pack(Ys,Zs).	range(I,I,[I]). range(I,K,[I L]) :- I < K, I1 is I + 1, range(I1,K,L).	has_factor(N,L) :- N mod L =:= 0. has_factor(N,L) :- L * L < N, L2 is L + 2, has_factor(N,L2).
transform([],[]). transform([[X Xs] Ys],[[N,X] Zs]) :- length([X Xs],N), transform(Ys,Zs).	combination(0,_,[]). combination(K,L,[X Xs]) :- K > 0, el(X,L,R), K1 is K-1, combination(K1,R,Xs).	is_prime(2). is_prime(3). is_prime(P) :- integer(P), P > 3, P mod 2 =\= 0, \+ has_factor(P,3).
encode(L1,L2) :- pack(L1,L), transform(L,L2).	el(X,[X L],L). el(X,[_ L],R) :- el(X,L,R).	qsort([],[]). qsort([H T],S) :- split(H,T,L, R),qsort(L,LS), qsort(R,RS), append(LS,[H RS],S).
strip([],[]). strip([[1,X] Ys],[X Zs]) :- strip(Ys,Zs). strip([[N,X] Ys],[[N,X] Zs]) :- N > 1, strip(Ys,Zs).	rotate([],_,[]) :- !. rotate(L1,N,L2) :- length(L1,NL1), N1 is N mod NL1, split(L1,N1,S1,S2), append(S2,S1,L2).	rnd_select(_,0,[]). rnd_select(Xs,N,[X Zs]) :- N > 0, length(Xs,L), I is random(L) + 1, remove_at(X,Xs,I,Ys), N1 is N - 1, rnd_select(Ys,N1,Zs).
encode_modified(L1,L2) :- encode(L1,L), strip(L,L2).	lotto(N,M,L) :- range(1,M,R), rnd_select(R,N,L).	rnd_permu(L1,L2) :- length(L1,N), rnd_select(L1,N,L2).

Semester 1 Haskell

add :: !Int → Int → Int

inc = add 1

map :: (a → b) → [a] → [b]

map f [] = []

map f (x:xs) = f x : map f xs

(++) :: [a] → [a] → [a]

[] ++ ys = ys

(x:xs) ++ ys = x:(xs ++ ys)

(.) :: (b → c) → (a → b) → (a → c)

f . g = \x → f(g x)

ones = 1: ones

length n = n : length (n+1)

squares = map (^2) (numbers 0)

fib = \b : [a] → b | (a,b) ← zip fib (map fib fib)

zip (x:xs) (y:ys) = (x,y) : zip xs ys

zip xs ys = []

head (x:xs) = x

head [] = error ; head :: [a] → a

length :: [a] → Int

length [] = 0

length (x:xs) = 1 + length xs

tail :: [a] → [a]

tail (x:xs) = xs.

$x \text{ 'elem' } c = \text{False}$

$x \text{ 'elem' } (y: \alpha) = x == y \wedge (\exists \alpha \text{ 'elem' } y)$

$\text{elem} : (\text{Eq } \alpha) \Rightarrow \alpha \rightarrow [\alpha] \rightarrow \text{Bool}$

$(+), (-), (\times) : (\text{Num } \alpha) \Rightarrow \alpha \rightarrow \alpha \rightarrow \alpha$

$\text{negate}, \text{abs} : (\text{Num } \alpha) \Rightarrow \alpha \rightarrow \alpha$.

nearest, round, floor, ceiling, numerator, denominator, conjugate, hollow, sqrt,
tuke, drop, filter