Pregătire examen

I CALCUL LAMBDA

1 Calcul lambda

a) 1) tb să înlocuim ap. libere

2) ale variabilei x

3) în expr. $\lambda y.(x\ y)$

4) cu y

$(\lambda x.\lambda y.(x\ y)\ (y))$ — x: se înlocuiește; corp; înlocuiesc cu y; y → apariție liberă; x → apariție liberă

b) β-reducere: $\lambda y.(y\ y)$ pt că y devine legată & mută, e legat la acest y

tb să aplic α conversie pt a evita asta

$$(\lambda x.\lambda y.(x\ y)\ y) \to_\alpha \lambda x.\lambda z.(x\ z)\ y) \to_\beta \lambda z.(x\ z)_{[y/x]} = \lambda z.(y\ z)$$

↓ în vit. ap. lib.

2. $(\lambda x.(\lambda x.x\ x)\ a) \to_\beta (\lambda x.x\ a) \to_\beta x_{(a/x)} \to a$

$\lambda x.x\ x \downarrow_\beta x$; de la dr la st.

de la st la dr: $(\lambda x.(\lambda x.x\ x)\ a) \to_\beta (\lambda x.x\ a) \to a.$

ap lib pt primu cl st tb înloc cu a

3. $((\lambda y.(\lambda x.\lambda y.x\ (y))\ \lambda x.x)\ \Omega)$ — liber; înloc

α conv pe parant. int., să aut ap. lib a lui y → nu se modif nimic nu avem ap...

$\to_\beta ((\lambda x.\lambda y.x\ (\lambda x.x))\ \Omega) \to (\lambda y.\lambda x.x\ \Omega) \to \lambda x.x$

4. $(\lambda y.((\lambda x.\lambda y.x\ (y)\ \Omega)\ \lambda x.\lambda y.y) \to_\beta ((\lambda x.\lambda y.x\ \lambda x.\lambda y.y)\ \Omega) \to (\lambda y.\lambda x.\lambda y.y\ \Omega) \to$ (nu am ap lib.)

$\to \lambda x.\lambda y.y \to$ e o fct cu 2 arg care într-al 2-lea arg și ignoră pe primul

5. $(((\lambda x.\lambda y.\lambda z.y\ \lambda x.x)\ (\lambda z.\lambda t.z\ z))\ \Omega)$ $\downarrow_\beta$ — liber

nu am app lib a lui x ⇒ $((\lambda y.\lambda z.(y)\ (\lambda z.\lambda t.z\ z))\ \Omega) \to_\alpha$

$((\lambda y.\lambda w.y\ (\lambda z.\lambda t.z\ z))\ \Omega)$

dacă aș face β red: $(\lambda z.(\lambda z.\lambda t\ z\ \ z)\ \Omega)$ — legat

$\to_\beta (\lambda w.(\lambda z.\lambda t.z\ \ z)\ \Omega) \to (\lambda z.\lambda t.z\ \ z) \to_\beta (\lambda t.z$ — lib.

6. $0 := \lambda f.\lambda x.x$ $succ := \lambda n.\lambda f.\lambda x(f((n\ f)\ x))$

(succ 0)!

$(\lambda n.\lambda f.\lambda x(f\ ((n\ f)\ x))\ \lambda f.\lambda x.x) \rightarrow \lambda f.\lambda x(f((\lambda f.\lambda x.x\ f)x)) \rightarrow$

↳ nu aruncasa fără să mai param

$\rightarrow \lambda f.\lambda x(f(\lambda x.x\ x)) \rightarrow \lambda f.\lambda x(f\ x)$ nu mai am ce să-i mai fac

7. $true = \lambda x\ y.x$

$false = \lambda x\ y.y$

$\lambda x\ y.(x\ y\ true)$?

$\lambda x\ y\ z.exp\ a\ b\ c) = (((\lambda x.\lambda y.\lambda z.exp\ a)b)c)$

ca fără op. le true & false a.î. op e ... undeva...

$true = \lambda x.\lambda y.x$

$false = \lambda x.\lambda y.y$

$op = \lambda x.\lambda y\ (x\ y\ \lambda x.\lambda y.x)$

NOT → unar, restul binari

și aplic pe toate rând pos, și pe cu morgan.

$(op\ true\ y) \Rightarrow (\lambda x.\lambda y.(x\ y\ true)\ true\ y) \rightarrow (true\ y\ true) \rightarrow$

putem înlocui true cu def lui, dar o facem pentru să nu ne încurcăm în def.

o fac când e lungă. ei vor fi aplic. lui x cu param y și true

$\rightarrow (\lambda x.\lambda y.x\ \ y\ true) \rightarrow y$

def true

$(op\ false\ y) \rightarrow (\lambda x.\lambda y(x\ y\ true)\ false\ y) \rightarrow (false\ y\ true) \rightarrow$

$\rightarrow (\lambda x.\lambda y.y\ y\ true) \rightarrow true$

op ≡ "=>" → $p \Rightarrow q = \neg p \vee q$

⇓

$\neg p = false \vee y \Rightarrow y$

$p = F \Rightarrow true \vee y = true$

II SINTEZĂ DE TIP:

1) f xs = head xs xs.

Pt început pornesc de la premisa că f nu are constr., de la cu. gen.

Apoi văd ce constr. le trebuie

init: f :: a -> b

cum head xs => xs tb să fie o listă => xs :: [t1]; a = [t1]

merg mai departe & văd că nu am numai head xs =>

=> head xs xs = (head xs) xs

primul el din listă să-l aplic pe xs =>

=> el din xs tb să fie f-ţ.

el din xs pentru tipul t1 => t1 ≡ t2 -> t3

param. xs => t2 ≡ [t1]

=> t1 ≡ [t1] -> t3 ≡ [[t1]->t3] -> t3 -> eroare de sinteză de tip.

definiţie ciclică: t1 e exprimat cu ce depinde de t1

g xs = head xs 2 ce înt xs prin a param cu de tip Number

2) f x y = (head x, head y) ? f (tail x) (tail y)

↳ cons

pornesc de la cazul gen: adaug tipurile peste f ...

f :: a -> b -> c vez. -> pornesc de la cazul gen & încerc să obţ constr.

↓ x, ↓ y

head x => x :: [t1] => a ≡ [t1] nu-i toţi au acelaşi tip.

ibidem pt b: head y => b ≡ [t2]

Rez e tot o listă, de perechi: => [(t1,t2)] -> merge sau nu poate fi dată

⇓

f :: [t1] -> [t2] -> [(t1,t2)].

[[t1,t2]] ✗ ≠ t1 ≡ t2

eroare de sint de tip pt. că se aj la capătul de listă şi pune pe [] => head eroare din pct de vedere. ← de pb.

a sint de tip nu ved. gasp eroarea asta, e în rap din tip

3) f x y = if x == head y then [x] else f x (tail y)

↓ x tb să implem. Eq pt că am ==

$f :: a \to b \to c$

head y ⇒ y = listă ⇒ y :: [t1] ⇒ b = [t1]

x == head y ⇒ x :: t1 + constr Eq t1

pot deduce ceva despre c?

pe baza def, rez lui f este o listă cu x ⇒ c ≡ [t1] ≡ b.

tail y = tail [listă] ⇒ f :: Eq t1 ⇒ t1 → [t1] → [t1]

4) g f (x, y) = x ++ map f y

tuplu

$g :: a \to (b, c) \to d$

map f y ⇒ f este o fct ⇒ f ≡ t1 → t2

y e o listă ⇒ y = [t1] ≡ c

t2 tb să fie de tipul lui x pt că avem append + o listă ⇒

⇒ x ++ → x listă cu el din listă rez map ⇒ x :: [t2].

⇒ g :: (t1 → t2) → ([t2], [t1]) → [t2].

III. INSTANȚIERE

1) Suprascrie în Hs op comp pe listă a.î. o listă să fie <= alta dacă toate el ei primei < el ei a doua.

[2, 3, 1] < [5, 4, 3] dar nu [2, 4] <= [3, 5]

Aș putea suprascrie op. de comp pe liste

sau (filter pe fiecare el din prima și verif pe toate din a doua apoi length, apoi văd dacă length de filter = length listă

dar cum fac suprascr.? → inst tipul boolș pt care supr. op.

⇒ instance Ord a ⇒ Ord [a] where -- a tb să fie de ord. pt că tb să comparăm toate el din listă.

x1 <= x2 = maximum x1 <= minimum x2

Ord sau listă el de op → constr.

2) Evit o posib inst a clasei

```
class MyClass c where
  f :: c a -> a.
instance MyClass [] where
  f = head
```

(constructor -> c e de tip listă; liste fără constr.)

-- a e tipul el. din ocel c

-- f ia ca prim listă cu a-uri și întoarce un a => head

3) supraînc. op + = ||, * = && pt bool:

clasa cu + & * = Num -> cls pt care nu se face inst.

=> instance Num Bool where -- zic ce se întâmplă pt (+) & (*)
(tipul pt care nu se face inst.) -- gen cum o să fie

```
(+) = (||)   -- să fie sau
(*) = (&&)   -- nu te complica, ai făcut cu fiș-o cont
```

-- am pus bool pt că nu se face pt val booleene

c deo => tip de date param în fct de a => Maybe sau liste. Sau tip nou de date

4) generaliz lui filter pt op pe constr mai ce utiliz tipul a

filter :: a -> Bool -> [a] -> [a]

data Maybe a = Just a | Nothing

a) class Filterable c where -- fac o class + zic tipul de date

filter' :: (a -> Bool) -> c a -> c a (nu zic că e pe liste, ci un constr. care utiliz. tipul a)

b) instance Filterable [] where

```
filter' f xs = [x | x <- xs, f x]
```

-- obiș acum implem: cu list comprehension ori cu filter' = filter (pot să pun direct!!)

c) - similar pt Maybe, gen explic să se intr. mai ce vrei fac filter pe cele 2 variante

IV LOGICA CU PREDICATE

dem. explicit, neg & R.A.

1) $\forall x.(P(x) \Rightarrow Q(x))$ are consec. log $\exists y. P(y) \Rightarrow \exists z. Q(z)$

a) aducem a primei prop la forma clauzală

în loc de $\Rightarrow$ avem $\neg p \vee q$

poiecuse & înțelese explicațiile din curs + ex. cu pași

elim implic

împing neg p̂nă în fața at.

redem. variab

dacă nu sunt $\forall$ înainte lui $\Rightarrow$ înlocuim cst, altfel îmi

iau o variab care depinde de aia

—vezi PDF

$\forall x(P(x) \Rightarrow Q(x))$ | $P \Rightarrow Q \equiv \neg P \vee Q$

renunț la implicație | $\Downarrow$ $\forall x.(\neg P(x) \vee Q(x))$

câte clauze avem? ← nu mai avem cu ce să facem până la

UNA | elim quantif univ.

cu const $\neg P(x)$ și $Q(x)$ $\Rightarrow$ clauze: $\{\neg P(x), Q(x)\}$.

b). neg a doua prop + ad. la forma clauzală

$\exists y\, P(y) \Rightarrow \exists z. Q(z) \Rightarrow$ primim dată o neg, pun NOT în față $\Rightarrow$

$\Rightarrow \neg(\exists y. P(y) \Rightarrow \exists z. Q(z))$ (apoi scap de implicație) $\Rightarrow$

$\Rightarrow \neg(\neg(\exists y. P(y)) \vee \exists z. Q(z))$ → dist sistem. din față $\Rightarrow$

$\Rightarrow \neg(\neg(\exists y. P(y)) \wedge \neg(\exists z. Q(z))$ $\quad \neg(P \vee Q) \equiv \neg P \wedge \neg Q$

↳ se duc $\Rightarrow \exists y. P(y) \wedge \forall z. \neg Q(z) \Rightarrow$ pot substitui $\Rightarrow$

$\neg\exists = \forall$ | câte clauze am? 1) $\exists y. P(y)$ & 2) $\forall z. \neg Q(z)$

clauze: $\{P(c)\}, \{\neg Q(z)\}$, $c =$ ct.

c) rezolutie pe clauzele obt. ant.

(1): $\{\neg P(x), Q(x)\}$

(2): $\{P(c)\}$

(3): $\{\neg Q(z)\}$

(1) & (2) unifică dar tb să folosesc o substit => $x \leftarrow c$ => îmi rămâne Q =>
=> (4) $\{Q(c)\}$

{ținem cont de cine e-n st stânga și cine-n dr și în dreapta }

Ce clauze unific? Ca să poată să unifice, ele trebuiau să ... Aplicăm un pas de rezoluție.

$\{p_1, \dots, r, \dots, p_n\}$

$\{q_1, \dots, \neg r, \dots, q_m\}$

$\{p_1, \dots, p_n, q_1, \dots, q_m\}$ fără r și fără ¬r

(1) & (3): $\neg P(x)$ (4)

(4) & (2) □

diferit = imag cu ... altfel ...

se leagă predicatul fără ... cuantificatorul

(3) și (4) unifică; din nou, am nevoie substit => $z \leftarrow c$ => ∅ □ ← clauza vidă ⇓ am demonstrat R.A ca caz

2) Cine învață, nu moare de foame. moareDe(ion, foame)

Ion nu învață => ∀ Cine. învață(Cine) => ¬moareDe(Cine, foame).

Cine = variabilă, foame = ct; ion ==

∀ Cine. învață(Cine) => ¬moareDe(Cine, foame).

moareDe(ion, foame) -> nu am ce să fac.

¬învață(ion)

pred. ∀x. învață(x) => ¬moareDe(x, foame) => elimin =>

=> ∀x (¬învață(x) ∨ ¬moareDe(x, foame)) => elim ∀ =>

=> Clauza: {¬învață(x), ¬moareDe(x, foame)} (1)

{moareDe(ion, foame)} (2)

neg concl. => {învață(ion)} (3)

(1) cu (2) unifică => $x \leftarrow ion$ => {¬învață(ion)} (4)

(3) cu (4) => □ => ¬învață(ion)

de principiu cerințe de teorie la Prolog

Haskell: documentație la sinteza de tip, instanțiere cu sp., implem.

implem predicate la Prolog cu predicate interne pt liste min, max, ultimul, primul el.

rezolvări ex. exam. private în ...