

Paradigme de Programare

Conf. dr. ing. Andrei Olaru

andrei.olaru@upb.ro | cs@andreiolaru.ro
Departamentul de Calculatoare

2021

Cursul 3

Anexă: TDA pentru calcul λ

- 1 Introducere
- 2 Lambda-expresii
- 3 Reducere
- 4 Evaluare
- 5 Limbajul lambda-0 și incursiune în TDA
- 6 Racket vs. lambda-0
- 1 Anexă: TDA

Anexă: TDA

→ Folosim notația:

- $\lambda x_1. \lambda x_2. \dots \lambda x_n. E \rightarrow \lambda x_1 x_2 \dots x_n. E$
- $((\dots((E A_1) A_2) \dots) A_n) \rightarrow (E A_1 A_2 \dots A_n)$

→ Pentru definirea unui **Tip de Date Abstract** avem nevoie de:

- **Constructorii de bază** → un set minimal de funcții, prin aplicarea (eventual, repetată) cărora se poate construi oricare element din mulțimea de valori a tipului.
 - constructorii de bază construiesc **valorile**.
- **Operatori** → setul complet de funcții care pot lucra cu valorile din tipul de bază.
 - operatorii arată **ce operații** putem face cu valorile.
- **Axiome** → definesc rezultatul operatorilor pentru toate posibilele valori (ne ajută de constructorii de bază).
 - axiomele arată **ce rezultate** obținem din operații.

· Constructori: $\left\{ \begin{array}{l} T : \rightarrow Bool \\ F : \rightarrow Bool \end{array} \right.$

· Operatori: $\left\{ \begin{array}{l} not : Bool \rightarrow Bool \\ and : Bool^2 \rightarrow Bool \\ or : Bool^2 \rightarrow Natural \end{array} \right.$

· Axiome: $\left\{ \begin{array}{l} not : not(T) = F \\ \quad not(F) = T \\ and : and(T, a) = a \\ \quad and(F, a) = F \\ or : or(T, a) = T \\ \quad or(F, a) = a \end{array} \right.$



Intuiție: **selecția** între cele două valori, *true* și *false*

- $T \equiv_{\text{def}} \lambda x y.x$
- $F \equiv_{\text{def}} \lambda x y.y$
- Comportament de **selector**:
 - $(T a b) \rightarrow (\lambda x y.x a b) \rightarrow a$
 - $(F a b) \rightarrow (\lambda x y.y a b) \rightarrow b$

- $not \equiv_{\text{def}} \lambda x.(x F T)$
 - $(not T) \rightarrow (\lambda x.(x F T) T) \rightarrow (T F T) \rightarrow F$
 - $(not F) \rightarrow (\lambda x.(x F T) F) \rightarrow (F F T) \rightarrow T$

- $and \equiv_{\text{def}} \lambda x y.(x y F)$
 - $(and T a) \rightarrow (\lambda x y.(x y F) T a) \rightarrow (T a F) \rightarrow a$
 - $(and F a) \rightarrow (\lambda x y.(x y F) F a) \rightarrow (F a F) \rightarrow F$

- $or \equiv_{\text{def}} \lambda x y.(x T y)$
 - $(or T a) \rightarrow (\lambda x y.(x T y) T a) \rightarrow (T T a) \rightarrow T$
 - $(or F a) \rightarrow (\lambda x y.(x T y) F a) \rightarrow (F T a) \rightarrow a$

· Operator: $| \textit{if} : \textit{Bool} \times A \times A \rightarrow A$

· Axiome: $\left| \begin{array}{l} \textit{if}(T, a, b) = a \\ \textit{if}(F, a, b) = b \end{array} \right.$

● Implementare: $\textit{if} \equiv_{\text{def}} \lambda c t e. (c t e)$

● $(\textit{if} T a b) \rightarrow (\lambda c t e. (c t e) T a b) \rightarrow (T a b) \rightarrow a$

● $(\textit{if} F a b) \rightarrow (\lambda c t e. (c t e) F a b) \rightarrow (F a b) \rightarrow b$

● Funcție **nestrictă!**

· Constructori: \mid $pair : A \times B \rightarrow Pair$

· Operatori: \mid $fst : Pair \rightarrow A$
 $snd : Pair \rightarrow B$

· Axiome: \mid $snd(pair(a, b)) = b$
 $fst(pair(a, b)) = a$



Intuiție: pereche \rightarrow funcție ce așteaptă **selectorul**, pentru a-l aplica asupra membrilor

- $pair \equiv_{\text{def}} \lambda x y z.(z x y)$
 - $(pair a b) \rightarrow (\lambda x y z.(z x y) a b) \rightarrow \lambda z.(z a b)$
- $fst \equiv_{\text{def}} \lambda p.(p T)$
 - $(fst (pair a b)) \rightarrow (\lambda p.(p T) \lambda z.(z a b)) \rightarrow (\lambda z.(z a b) T) \rightarrow (T a b) \rightarrow a$
- $snd \equiv_{\text{def}} \lambda p.(p F)$
 - $(snd (pair a b)) \rightarrow (\lambda p.(p F) \lambda z.(z a b)) \rightarrow (\lambda z.(z a b) F) \rightarrow (F a b) \rightarrow b$

· Constructori: $\left\{ \begin{array}{l} nil : \rightarrow List \\ cons : A \times List \rightarrow List \end{array} \right.$

· Operatori: $\left\{ \begin{array}{l} car : List \setminus \{nil\} \rightarrow A \\ cdr : List \setminus \{nil\} \rightarrow List \\ null? : List \rightarrow Bool \\ append : List^2 \rightarrow List \end{array} \right.$

· Axiome: $\left\{ \begin{array}{l} car : car(cons(e, L)) = e \\ cdr : cdr(cons(e, L)) = L \\ null? : null?(nil) = T \\ \quad \quad null?(cons(e, L)) = F \\ append : append(nil, B) = B \\ \quad \quad append(cons(e, A), B) = cons(e, append(A, B)) \end{array} \right.$



Intuiție: listă \rightarrow pereche (*head*, *tail*)

- $nil \equiv_{\text{def}} \lambda x. T$
- $cons \equiv_{\text{def}} pair$
 - $(cons\ e\ L) \rightarrow (\lambda x\ y\ z.(z\ x\ y)\ e\ L) \rightarrow \lambda z.(z\ e\ L)$
- $car \equiv_{\text{def}} fst$
- $cdr \equiv_{\text{def}} snd$
- $null? \equiv_{\text{def}} \lambda L.(L\ \lambda x\ y.F)$
 - $(null?\ nil) \rightarrow (\lambda L.(L\ \lambda x\ y.F)\ \lambda x.T) \rightarrow (\lambda x.T\ \dots) \rightarrow T$
 - $(null?(cons\ e\ L)) \rightarrow (\lambda L.(L\ \lambda x\ y.F)\ \lambda z.(z\ e\ L)) \rightarrow (\lambda z.(z\ e\ L)\ \lambda x\ y.F) \rightarrow (\lambda x\ y.F\ e\ L) \rightarrow F$

- $append \equiv_{\text{def}} \lambda A B. (if (null? A) B (cons (car A) (append (cdr A) B)))$

· Problemă: expresia **nu** admite formă închisă! → vezi eliminarea recursivității textuale.

· Constructori: $\left\{ \begin{array}{l} \text{zero} : \rightarrow \text{Natural} \\ \text{succ} : \text{Natural} \rightarrow \text{Natural} \end{array} \right.$

· Operatori: $\left\{ \begin{array}{l} \text{pred} : \text{Natural} \setminus \{\text{zero}\} \rightarrow \text{Natural} \\ \text{zero?} : \text{Natural} \rightarrow \text{Bool} \\ \text{add} : \text{Natural}^2 \rightarrow \text{Natural} \end{array} \right.$

· Axiome: $\left\{ \begin{array}{l} \text{pred} : \text{pred}(\text{succ}(n)) = n \\ \text{zero?} : \text{zero?}(\text{zero}) = T \\ \text{zero?} : \text{zero?}(\text{succ}(n)) = F \\ \text{add} : \text{add}(\text{zero}, n) = n \\ \text{add} : \text{add}(\text{succ}(n), m) = \text{succ}(\text{add}(n, m)) \end{array} \right.$



Intuiție: număr \rightarrow **listă** cu lungimea egală cu valoarea numărului

- $zero \equiv_{\text{def}} nil$
- $succ \equiv_{\text{def}} \lambda n.(cons\ nil\ n)$
- $pred \equiv_{\text{def}} cdr$
- $zero? \equiv_{\text{def}} null$
- $add \equiv_{\text{def}} append$

- o funcție f are un **punct fix** dacă $\exists x$, a.î. $(f\ x) = x$
- un **combinator de punct fix** este o funcție (funcțională) Fix care pentru orice funcție f care are un punct fix întoarce un punct fix al acesteia:
 $(f\ (Fix\ f)) = (Fix\ f)$
- o astfel de funcție este:

$$Fix = \lambda f.(\lambda x.(f\ (x\ x))\ \lambda x.(f\ (x\ x)))$$

- definiția recursivă a unei funcții f conține numele funcției, deci nu este **expresie închisă**.
- pentru o funcție f , inițial scrisă recursiv textual $f \equiv_{\text{def}} \lambda x. \dots (f \dots) \dots$, ne punem problema crearea unei funcții F care primește pe f ca parametru, astfel încât:

$$F \equiv_{\text{def}} \lambda f. \underbrace{\lambda x. \dots (f \dots) \dots}_{\text{definiție } f}$$

- definiția lui F este expresie închisă, iar $(F f)$ va avea exact semnificația lui f , dar trebuie să-i transmitem lui F pe f ca parametru.
- dar dacă $(F f) = f$ (ca semnificație), atunci f este punct fix al lui F , deci pentru a obține f este suficient să apelăm $(\text{Fix } F)$, care este expresie închisă. **Done.**

Funcția $length \equiv_{\text{def}} \lambda L.(if (null L) zero (succ (length (cdr L))))$

cu $Length \equiv_{\text{def}} \lambda f L.(if (null L) zero (succ (f (cdr L))))$

$(Fix Length) = (\lambda f.(\lambda x.(f (x x)) \lambda x.(f (x x)))$
 $\lambda f.\lambda L.(((if (null L)) zero) (succ (f (cdr L)))))$

Verificăm axiomele pentru $(Fix Length)$:

$((Fix Length) nil)$

și pentru

$((Fix Length) L)$, cu $L = ((cons e) list)$

$$\text{length} = (\text{fix length}) = (\lambda f. (\lambda x. (f (x x)) \lambda x. (f (x x)))) \text{ len}$$

$$= (\lambda x. (\text{len } (x x)) \lambda x. (\text{len } (x x)))$$

$$= (\text{len } (\lambda x. (\text{len } (x x)) \lambda x. (\text{len } (x x))))$$

$$= \lambda h. (\text{if (null L) zero (succ (cdr L))}))$$

→ pt nil ⇒ zero

→ pt (cons x list)

$$\Rightarrow (\text{succ } (A \text{ (cdr (cons x list))}))$$

$$\Rightarrow (\text{succ } (A \text{ list}))$$

$$\Rightarrow (\text{succ } ((\lambda x. (\text{len } (x x)) \lambda x. (\text{len } (x x))) \text{ list}))$$

$$= (\text{fix length } \text{vari})$$

⇒ length (pt că length este punct fix al length)

$$= (\text{succ } (\text{length list})) \rightarrow \text{correct}$$

→ și termină?

$$(\lambda x. (\text{len } (x x)) \lambda x. (\text{len } (x x)))$$

$$= (\text{len } (\lambda x. (\text{len } (x x)) \lambda x. (\text{len } (x x))))$$

$$= \text{exact } \text{len pt list, care este (cdr L)}$$

→ cum arăsați 1 element

- dacă L are 1 element, list are 0 element (⇒ nil) și apelul de mai sus dă zero.