Paradigme de Programare

Conf. dr. ing. Andrei Olaru

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

2020

Cursul 3

Anexă: TDA pentru calcul λ

Cursul 3: Anexă: TDA pentru calcul λ

Anexă: TDA

Notație

$\rightarrow$ Folosim notația:

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

Elemente TDA

λ

$\rightarrow$ Pentru definirea unui Tip de Date Abstract avem nevoie de:

- Constructori de bază $\rightarrow$ 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 $\rightarrow$ setul complet de funcții care pot lucra cu valorile din tipul de bază.
  - operatorii arată ce operații putem face cu valorile.
- Axiome $\rightarrow$ definesc rezultatul operatorilor pentru toate posibilele valori (ne ajutăm de constructorii de bază).
  - axiomele arată ce rezultate obținem din operații.

TDA *Bool*

Specificare

· Constructori:

$$\begin{array}{|ll} T: & \to Bool \\ F: & \to Bool \end{array}$$

· Operatori:

$$\begin{array}{|ll} not: & Bool \to Bool \\ and: & Bool^2 \to Bool \\ or: & Bool^2 \to Natural \end{array}$$

· Axiome:

$$\begin{array}{|ll} not: & not(T) = F \\ & not(F) = T \\ and: & and(T, a) = a \\ & and(F, a) = F \\ or: & or(T, a) = T \\ & or(F, a) = a \end{array}$$

TDA *Bool*

Implementarea constructorilor de bază

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 selectori:
  - $(T\ a\ b) \to (\lambda x\ y.x\ a\ b) \to a$
  - $(F\ a\ b) \to (\lambda x\ y.y\ a\ b) \to b$

TDA *Bool*
Implementarea operatorilor

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

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

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

TDA *Bool*
Condiționala *if*

- Operator: $\mid if : Bool \times A \times A \to A$
- Axiome: $\mid \begin{array}{l} if(T, a, b) = a \\ if(F, a, b) = b \end{array}$

  - Implementare: $if \equiv_{\text{def}} \lambda c\ t\ e.(c\ t\ e)$
    - $(if\ T\ a\ b) \to (\lambda c\ t\ e.(c\ t\ e)\ T\ a\ b) \to (T\ a\ b) \to a$
    - $(if\ F\ a\ b) \to (\lambda c\ t\ e.(c\ t\ e)\ F\ a\ b) \to (F\ a\ b) \to b$

  - Funcție nestrictă!

TDA *Pair*

Specificare

- Constructori: $\mid pair : A \times B \to Pair$
- Operatori: $\mid fst : Pair \to A$, $snd : Pair \to B$
- Axiome: $\mid snd(pair(a,b)) = b$, $fst(pair(a,b)) = a$

TDA *Pair*
Implementare

Intuiție: pereche → funcție ce așteaptă selectorul, pentru a-l aplica asupra membrilor

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

TDA *List*

Specificare

· Constructori:

$$\begin{array}{ll} nil: & \to List \\ cons: & A \times List \to List \end{array}$$

· Operatori:

$$\begin{array}{ll} car: & List \setminus \{nil\} \to A \\ cdr: & List \setminus \{nil\} \to List \\ null?: & List \to Bool \\ append: & List^2 \to List \end{array}$$

· Axiome:

$$\begin{array}{ll} car: & car(cons(e, L)) = e \\ cdr: & cdr(cons(e, L)) = L \\ null?: & null?(nil) = T \\ & null?(cons(e, L)) = F \\ append: & append(nil, B) = B \\ & append(cons(e, A), B) = cons(e, append(A, B)) \end{array}$$

TDA *List*
Implementare

Intuiție: listă → pereche (*head*, *tail*)

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

TDA *List*
Implementare *append*

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

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

TDA *Natural*

Specificare

· Constructori:

$$\begin{array}{l|ll} & zero: & \rightarrow Natural \\ & succ: & Natural \rightarrow Natural \end{array}$$

· Operatori:

$$\begin{array}{l|ll} & pred: & Natural \setminus \{zero\} \rightarrow Natural \\ & zero?: & Natural \rightarrow Bool \\ & add: & Natural^2 \rightarrow Natural \end{array}$$

· Axiome:

$$\begin{array}{l|ll} & pred: & pred(succ(n)) = n \\ & zero?: & zero?(zero) = T \\ & & zero?(succ(n)) = F \\ & add: & add(zero, n) = n \\ & & add(succ(n), m) = succ(add(n, m)) \end{array}$$

TDA *Natural*
Implementare

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

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

Eliminarea recursivității textuale

Combinator de punct fix

- o funcție f are un punct fix dacă $\exists x$, a.î. $(fx) = x$
- 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:

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

Eliminarea recursivității textuale

Recursivitate semantică

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

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

- 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 $(Fix\ \ F)$, care este expresie închisă. **Done**.

Eliminarea recursivității textuale
Exemplu

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

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 $L = ((cons\ \ e)\ \ list)$

length = (Fix Length) = (λf.(λx.(f (x x)) λx.(f (x x))) Len)

= (λx.(Len (x x)) λx.(Len (x x))

= (Len (λx.(Len (x x)) λx.(Len (x x)))) A

= λL.(if (null L) zero (succ (A (cdr L))))

→ pt nil ⇒ zero

→ pt (cons x list)

⇒ (succ (A (cdr (cons x list))))

⇒ (succ (A list))

⇒ (succ ((λx.(Len (x x)) λx.(Len (x x))) list))

= (Fix Length) verif

= length (pt că length este punct fix al Lengthi)

= (succ (length list)) → corect

→ se termină?

(λx.(Len (x x)) λx.(Len (x x)))

= (Len (λx.(Len (x x)) λx.(Len (x x))))

= exact ~~pt list~~ len pt list, care este (cdr L)

→ cu avansat 1 element.

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