# Inducție Structurală

23 noiembrie 2017

## Tipuri de date abstracte (TDA)

Un TDA specifică:

- un set de **date (valori care aparțin tipului respectiv)**
- un set de **operații** care se pot face cu acele date

De ce "abstract"?

- **independent de implementare** (același TDA poate fi implementat în diverse moduri care respectă definiția)
- vizibil ca model matematic sau programat ca interfață

## Exemplu – Tipul de date abstract Nat

**Nume**: Nat
**Descriere**: numere naturale
**Tipuri importate**: Bool

---

**Constructori de bază**:

- $zero : \to Nat$ # orice numar natural este fie 0
- $succ : Nat \to Nat$ # fie succesorul altui numar natural

---

**Operatori**:

- $zero? : Nat \to Bool$ # testeaza daca un numar natural este 0
- $add : Nat * Nat \to Nat$ # aduna 2 numere naturale

---

**Axiome**:

- $zero?(zero) = true$
- $zero?(succ(n)) = false$
- $add(zero, n) = n$
- $add(succ(m), n) = succ(add(m, n))$

---

**Constructori de tip**

- toți operatorii unui tip t care **produc o valoare a lui t** se numesc constructori ai acelui tip
- **zero, succ, add** sunt, toți, constructori ai tipului Nat
- zero? nu este constructor pentru Nat, intrucat produce un Bool

**Constructori de bază**

- un set de constructori **necesari și suficienți** pentru a obține **toate valorile tipului**
- **zero** și **succ** sunt constructori de bază pentru Nat
- add nu este constructor de bază, întrucât nu e necesar; orice număr natural se poate obține din aplicări succesive de zero sau succ

**Constructorii unui tip t**, din punct de vedere al **parametrilor**

- **nulari**: au 0 argumente; fie $\Sigma_0$ mulțimea constructorilor de bază nulari (ex: zero pentru tipul Nat)
- **externi**: au un numar de argumente, dintre care **niciunul nu este de tip t**; fie $\Sigma_e$ mulțimea constructorilor de bază externi (nu există constructor extern în fișa tipului Nat)
- **interni**: au un numar de argumente, dintre care **cel puțin unul este de tip t**; fie $\Sigma_i$ mulțimea constructorilor de bază interni (ex: **succ** pentru tipul Nat)

**Axiome**

- rolul lor este să specifice **cum se comportă toți operatorii TDA-ului pe toate valorile TDA-ului**
- pentru aceasta, e suficient să specifice cum se comportă pe toți **constructorii de bază** (întrucât orice valoare a TDA-ului se poate obține numai din constructorii de bază)
- este extrem de important ca **axiomele să "acopere" toate valorile tipului**
- de asemenea ele trebuie sa fie scrise **neredundant** (de exemplu, la axiomele pentru add, am acoperit toate posibilitățile prin a varia doar primul parametru între zero și succ(m), nu era nevoie să procedăm la fel și pentru al doilea; în general, e foarte important (și ușurează mult demonstrațiile) să găsim exprimarea minimală și completă pentru axiome; în particular, când există mai mulți parametri de tip t, considerăm că scriem axiomele din punctul de vedere al unuia din ei, nu din al tuturor)

**Dimensiunea unei valori v a unui TDA t**

- reprezintă numărul aparițiilor constructorilor lui t în formula care desemnează pe v
- ex: succ(succ(zero)) are dimensiunea 3

# Inducție structurală

Cand este adecvată?

- când algoritmul (sau o parte din el) primește date de un tip t care satisfac o proprietate P (conform aserțiunii de intrare) și trebuie să producă date de tip t care satisfac, de asemenea, proprietatea P (conform aserțiunii de ieșire)
- faptul că proprietatea P este pastrată în urma aplicării algoritmului poate fi demonstrat prin inducție structurală

**Moduri de înțelegere a inducției structurale**

1. pentru a dovedi că proprietatea P va fi respectată în urma aplicării algoritmului de toate valorile posibile de tip t, va trebui sa arătăm că e respectată de **toți constructorii de bază**
2. pentru a dovedi ca proprietatea P va fi respectată în urma aplicării algoritmului de toate valorile posibile de tip t, vom face o **inducție după dimensiunea valorilor tipului t** (în cazul de bază vor fi constructorii **nulari** și **externi**, întrucât aceștia sunt singurii de dimensiune 1; **pasul de inducție** se va referi la **constructorii interni**, întrucât aceștia au proprietatea de a incrementa dimensiunea (P(n)→P(n+1), ca la inducția matematica))

Ambele viziuni sunt surprinse de aceeași schemă de inducție structurală:
**Cazuri de baza:**

$\forall \sigma \in \Sigma_0 \bullet P(\sigma)$

$\forall \sigma \in \Sigma_e, d \in Dom_\sigma \bullet P(\sigma(d))$

**Pas de inductie:**

$\forall x \in t \bullet P(x) \rightarrow P(\sigma(..., x, ...)), \forall \sigma \in \Sigma_i$

---

*atunci* $\forall x \in t \bullet P(x)$

# Exemplu

Fie TDA-ul List descris de fișa următoare:

**Nume:** List

**Tipuri importate:** Nat, Elem

**Constructori de bază:**

- $[] :\rightarrow List$ # orice listă este fie vidă
- $: : Elem * List \rightarrow List$ # fie un element adăugat la o altă listă
  # sintaxa (x:xs) = elem x adăugat la lista xs
  # constructorul este simbolizat prin simbolul ':', celelalalt simbol ':' semnifica ca urmeaza definirea operatorului

**Operatori:**

- $length : List \rightarrow Nat$ # lungimea unei liste
- $++ : List * List \rightarrow List$ # $A + +B$ = concatenarea listelor A și B

**Axiome:**

- $length([]) = 0$ (LEN1)
- $length(x : xs) = 1 + length(xs)$ (LEN2) # shortcut pentru succ(length(xs))
- $[] + +L = L$ (APP1)
- $(x : xs) + +L = x : (xs + +L)$ (APP2)

**Cerinta:** Să se demonstreze, prin inducție structurală, că ++ **e asociativ**.
Trebuie demonstrat, așadar, că (A++B)++C = A++(B++C), oricare ar fi A, B, C TDA-uri de tipul List Vom face inducție structurală dupa A.

**Caz de bază**:
$A = []$ (constructor nular)
Trebuie aratat că:
$([] + +B) + +C = [] + +(B + +C) \Leftarrow (APP1) \Rightarrow$
$B + +C = B + +C$
**(A)** oricare ar fi B si C Liste

**Pas de inducție**:
$A = (x : xs)$ (constructor intern)

**Ipoteza inductivă**:
$(xs + +B) + +C = xs + +(B + +C)$ $(II)$
Trebuie arătat că:
$((x : xs) + +B) + +C = (x : xs) + +(B + +C) \Leftarrow (APP2) \Rightarrow$
$(x : (xs + +B)) + +C = x : (xs + +(B + +C)) \Leftarrow (APP2) \Rightarrow$
$x : ((xs + +B) + +C) = x : (xs + +(B + +C)) \Leftarrow (II) \Rightarrow$
$x : (xs + +(B + +C)) = x : (xs + +(B + +C))$
**(A)**
Întrucât am epuizat toți constructorii de bază, rezultă că ++ e asociativ.

## Exerciții:

1. **O expresie aritmetică complet parantezată este definită astfel:**
   $0, 1, x, [e1 + e2], [e1 * e2], [-e2]$
   2 constructori nulari, 1 constructor extern, 3 constructori interni.
   Să notăm cu E acest tip de date.
   Se definesc operatorii:
   $\boldsymbol{eval(e, n) : E * N \to N}$

   - (E1) $eval(0, n) = 0$
   - (E2) $eval(1, n) = 1$
   - (E3) $eval(x, n) = n$
   - (E4) $eval([e1 + e2], n) = eval(e1, n) + eval(e2, n)$
   - (E5) $eval([e1 * e2], n) = eval(e1, n) * eval(e2, n)$
   - (E6) $eval([-e1], n) = -eval(e1, n)$

   $\boldsymbol{subst(e, f) : E * E \to E}$

   - (S1) $subst(0, f) = 0$
   - (S2) $subst(1, f) = 1$
   - (S3) $subst(x, f) = f$
   - (S4) $subst([e1 + e2], f) = [subst(e1, f) + subst(e2, f)]$
   - (S5) $subst([e1 * e2], f) = [subst(e1, f) * subst(e2, f)]$
   - (S6) $subst([-e], f) = [-subst(e, f)]$

   Sa se demonstreze prin inductie structurala ca, pentru orice e, f $\in$ E si n $\in \mathbb{N}$, proprietatea urmatoare este adevarata:
   **eval(subst(e, f), n)= eval(e, eval(f, n))**

2. **Fie tipul de date T LIST, definit prin constructorii de bază:**

   - $[] :\to TLIST$
   - $a : : l : T * TLIST \to TLIST$

   Se consideră **operatorii**:
   $\boldsymbol{head(l) : TLIST}\ []\ \boldsymbol{\to T}$

   - head(a::l) = a

   $\boldsymbol{range(l) : TLIST \to BOOL}$

   - $range([]) = false$
   - $range(a :: []) = (a == 1)$
   - $range(a :: l) = (a - 1 == head(l))\&\&range(l), pentrul[]$

   $\boldsymbol{sum(l) : TLIST \to} \mathbb{N}$

   - $sum([]) = 0$
   - $sum(a :: l) = a + sum(l)$

   Demonstraţi următoarea proprietate prin inducţie structurală:
   $\boldsymbol{range(l) \to sum(l) == head(l) * (head(l) + 1)/2}$

3. **Fie tipul de date LIST care reprezintă mulțimea listelor cu numere întregi definit prin următorii constructori de bază:**

   - $[] :\to LIST$
   - $(x : l) : INT * LIST \to LIST$

Fie FUN mulțimea funcțiilor definite astfel: $INT * INT \rightarrow INT$.
Se definesc funcțiile (operatori) următoare, cu axiomele aferente:

$\boldsymbol{fold : FUN * INT * LIST \rightarrow INT}$

- $fold(f, z, []) = z$
- $fold(f, z, x : l) = f(x, fold(f, z, l))$

$\boldsymbol{len : LIST \rightarrow INT}$

- $len([]) = 0$
- $len(x : l) = 1 + len(l)$

$\boldsymbol{(x, y) = y + 1}$
Să se demonstreze prin inducție structurală că P(l) este adevarată, pentru ∀ l ∈ LIST:
$\boldsymbol{P(l) = (len(l) = fold(inc, 0, l))}$

4. **Fie tipul de date asbtract BinTree definit prin constructorii de baza:**

   $empty : \rightarrow BinTree$ # Arborele binar vid
   $node : BinTree * Elem * BinTree \rightarrow BinTree$ # Subarborele stang, radacina, sub drept

   Se considera operatorii:

   $size : BinTree \rightarrow N$ # Numarul de elemente din arbore
   $all : (Elem \rightarrow Bool) * BinTree \rightarrow Bool$
   # Intoarce True daca p(x) este adevarat pentru toate elementele x din Tree, False altfel
   Axiomele pentru All:
   (All1): $all(p, empty) = True$
   (All2): $all(p, node(left, root, right)) = p(root)$ && $all(p, left)$ && $all(p, right)$
   $countif : (Elem \rightarrow Bool) * BinTree \rightarrow N$
   # countif(p, tree) = cate elemente din tree satisfac predicatul p(adica pentru cate elemente din tree p(x) este adevarat

   Scrieti axiomele pentru operatorii size si countif, apoi demonstrati prin inductie structurala:

$$\boldsymbol{all(p, t) \rightarrow countif(p, t) == size(t)} \tag{1}$$