Inductie Structurala

Tipuri de date abstracte (TDA)
Un TDA specifica:

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

De ce "abstract"?

- **independent de implementare** (acelasi TDA poate fi implementat in diverse moduri care respecta definitia)
- vizibil ca definitie matematica sau programat ca interfata

Exemplul 1 – Tipul de date abstract Nat

Nume: Nat
Descriere: numere naturale
Tipuri importate: Bool

Constructori de baza:
zero:->Nat // orice numar natural este ori 0
succ:Nat->Nat // ori succesorul altui numar natural

Operatori:
zero?:Nat->Bool // testeaza daca un numar natural este 0
add:Nat*Nat->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

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

Constructori de baza

- acei constructori cu uzul carora se **obtin toate valorile tipului**
- zero si succ sunt constructori de baza pentru Nat
- add nu este constructor de baza, intrucat nu e necesar; orice numar natural se poate obtine din aplicari succesive de zero sau succ

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

- **nulari**: primesc 0 parametri; fie Σ_0 multimea constructorilor de baza nulari (ex: zero pentru tipul Nat)

- **externi**: primesc un numar de parametri dintre care niciunul nu este de tip t; fie Σ_e multimea constructorilor de baza externi (nu exista constructor extern in fisa tipului Nat)
- **interni**: primesc un numar de parametri dintre care cel putin unul este de tip t; fie Σ_i multimea constructorilor de baza interni (ex: succ pentru tipul Nat)

Axiome

- rolul lor este sa specifice **cum se comporta operatorii TDA-ului pe toate valorile TDA-ului**
- pentru aceasta, e suficient sa specifice cum se comporta pe **toti constructorii de baza** (intrucat orice valoare a TDA-ului se poate obtine numai din constructorii de baza)
- este extrem de important ca axiomele sa "acopere" toate valorile tipului
- de asemenea ele trebuie sa fie scrise neredundant (de exemplu, la axiomele pentru add, am acoperit toate posibilitatile prin a varia doar primul parametru intre zero si succ(m), nu era nevoie sa facem asta si pentru al doilea; in general e foarte important (si usureaza mult demonstratiile) sa gasim exprimarea minimala si completa pentru axiome; in particular, cand exista mai multi parametri de tip t, consideram ca scriem axiomele din punctul de vedere al unuia din ei, nu din al tuturor)

Dimensiunea unei valori v a unui TDA t

- reprezinta **numarul aparitiilor constructorilor** lui t in formula care desemneaza pe v
- ex: succ(succ(zero)) are dimensiunea 3

Exemplul 2 – Tipul de date abstract Stack

Nume: Stack
Tipuri importate: Elem, Bool

Constructori de baza:
void:->Stack // orice stiva e ori vida
push:Elem*Stack->Stack // ori un element deasupra unei alte stive

Operatori:
empty?:Stack->Bool // testeaza daca stiva e vida
pop:Stack\{void}->Stack // intoarce stiva fara primul ei element
top:Stack\{void}->Elem // intoarce primul element din stiva

Axiome:
empty?(void)=true
empty?(push(e,s))=false
pop(push(e,s))=s
top(push(e,s))=e

Pe acest al doilea exemplu avem:

- constructori de baza: void, push
- constructori care nu sunt de baza: pop
- constructori de baza nulari: void
- constructori de baza interni: push
- niciun constructor de baza extern
- axiome definite pe intreg domeniul de definitie al fiecarui operator (de aceea pop si top nu sunt definite pe void)

Inductia structurala

Cand este adecvata?

- cand algoritmul (sau o parte din el) primeste date de un tip t care satisfac o proprietate P (conform asertiunii de intrare) si trebuie sa produca date de tip t care satisfac de asemenea proprietatea P (conform asertiunii de iesire)
- faptul ca proprietatea P este pastrata in urma aplicarii algoritmului poate fi demonstrat prin inductie structurala

Moduri de intelegere a inductiei structurale

1. pentru a dovedi ca proprietatea P va fi respectata in urma aplicarii algoritmului de toate valorile posibile de tip t, va trebui sa aratam ca e respectata de **toti constructorii de baza**
2. pentru a dovedi ca proprietatea P va fi respectata in urma aplicarii algoritmului de toate valorile posibile de tip t, vom face o **inductie matematica dupa dimensiunea valorilor tipului t** (cazul de baza vor fi constructorii nulari si externi, intrucat acestia sunt singurii de dimensiune 1; pasul de inductie se va referi la constructorii interni, intrucat acestia au proprietatea de a incrementa dimensiunea (P(n)=>P(n+1), ca la inductia matematica))

Ambele viziuni sunt surprinse de aceeasi schema de inductie structurala.

Schema inductiei structurale

$$\frac{\text{caz de baza}: \begin{array}{c}\forall \sigma \in \Sigma_0 \bullet P(\sigma) \\ \forall \sigma \in \Sigma_e, d \in Dom(\sigma) \bullet P(\sigma(d))\end{array} \quad \text{pas de inductie}: \dfrac{\text{ipoteza inductiva}: x \in t \bullet P(x)}{P(\sigma(\ldots, x, \ldots))} \forall \sigma \in \Sigma_i}{\forall x \in t \bullet P(x)}$$

Exercitiu: identificati cele 2 viziuni de mai sus pe aceasta schema.

Exercitii:

<u>**Ex1**</u>: Fie TDA-ul List descris de fisa urmatoare:

Nume: List

Tipuri importate: Nat, Elem

Constructori de baza:

[]:->List // orice lista este ori vida
: :Elem*List->List // ori un element adaugat la o alta lista
// sintaxa (x:xs) = elem x adaugat la lista xs

Operatori:
length:List->Nat // lungimea unei liste
++:List*List->List // A++B = concatenarea listelor A si B

Axiome:
length([])=0 (LEN1)
length(x:xs)=1+length(xs) // shortcut pt succ(length(xs)) (LEN2)
[]++L=L (APP1)
(x:xs)++L=x:(xs++L) (APP2)

P1) Sa se demonstreze prin inductie structurala ca ++ e asociativ.

Trebuie demonstrat asadar ca (A++B)++C = A++(B++C).
Vom face inductie structurala dupa A.

Caz de baza: A=[] (constructor nular)
Trebuie aratat ca:
([]++B)++C = []++(B++C) <=(APP1)=>
B++C = B++C □

Pas de inductie: A=(x:xs) (constructor intern)
Ipoteza inductiva: (xs++B)++C = xs++(B++C) (II)
Trebuie aratat ca:
((x:xs)++B)++C = (x:xs)++(B++C) <=(APP2)=>
(x:(xs++B))++C = x:(xs++(B++C)) <=(APP2)=>
x:((xs++B)++C) = x:(xs++(B++C)) <=(II)=>
x:(xs++(B++C)) = x:(xs++(B++C)) □

Intrucat am epuizat toti constructorii de baza, rezulta ca ++ e asociativ.

P2) Sa se demonstreze prin inductie structurala ca:
length(A++B) = length(A)+length(B).

Vom face inductie structurala dupa A.

Caz de baza: A=[] (constructor nular)
Trebuie aratat ca:
length([]++B) = length([])+length(B) <=(LEN1)=>
length([]++B) = length(B) <=(APP1)=>
length(B) = length(B) □

Pas de inductie: A=(x:xs) (constructor intern)

Ipoteza inductiva: length(xs++B) = length(xs)+length(B) (II)

Trebuie aratat ca:

length((x:xs)++B) = length(x:xs)+length(B) <=(APP2)=>
length(x:(xs++B)) = length(x:xs)+length(B) <=(LEN2)=>
1+length(xs++B) = 1+length(xs)+length(B) <=(II)=>
1+length(xs++B) = 1+length(xs++B) □