PARADIGME DE PROGRAMARE

Curs 6

Abstractizare.

Abstractizare – Cuprins

- Abstractizare
- Abstractizarea proceselor
  - Funcții ca abstracțiuni procedurale care rezolvă o sarcină
  - Funcționale
- Abstractizarea datelor
  - Tipuri de date abstracte
  - Structuri
- Anatomia Racket

Programele sunt scrise pentru a fi înțelese

Scop

- Gestiunea complexității intelectuale a programelor (care trebuie întreținute și dezvoltate, nu doar executate pe mașini de calcul)

Mijloace

- Primitive
  - datele și funcțiile oferite de limbaj
- Mijloace de combinare
  - cum se pot pune primitivele împreună și construi obiecte mai complexe din ele
- Mijloace de abstractizare
  - cum se pot folosi combinările de elemente primitive ca și când ele însele ar fi primitive

Abstractizare – Cuprins

- Abstractizare
- Abstractizarea proceselor
  - Funcții ca abstracțiuni procedurale care rezolvă o sarcină
  - Funcționale
- Abstractizarea datelor
  - Tipuri de date abstracte
  - Structuri
- Anatomia Racket

Abstracțiuni procedurale

- Wishful thinking ca strategie de a scrie programe funcționale
  - problemele complexe se descompun în subprobleme rezolvabile prin funcții dedicate
  - descriem soluția în termeni de concepte dorite, înainte de a deține implementarea lor
  - implementarea este stratificată:
    - La nivel superior codul, puternic abstractizat, descrie planul logic al rezolvării
    - La nivelurile inferioare sunt implementate conceptele
- Fiecare funcție dedicată (abstracțiune procedurală) este ca o cutie neagră
  - pentru funcția principală contează CE face funcția dedicată, nu CUM o face
  - orice funcție care îndeplinește sarcina este la fel de bună

Abstracțiuni procedurale - Exemplu

Exemplu: primul număr mai mare sau egal decât start care este palindrom în minim b baze dintre bazele 2, 3, 4, 5, 6, 7, 8, 9, 10.

```
1.  (define (first-b-pal start b)
2.    (if (min-b-bases? start b '(2 3 4 5 6 7 8 9 10))
3.        start
4.        (first-b-pal (+ 1 start) b)))
5.
6.  (define (min-b-bases? n b Bases)
7.    (cond ((zero? b) #t)
8.          ((null? Bases) #f)
9.          ((palindrome? (num->base n (car Bases))) (min-b-bases? n (- b 1) (cdr Bases)))
10.         (else (min-b-bases? n b (cdr Bases)))))
```

wishful thinking: îmi imaginez că min-b-bases? există

apoi o implementez (se putea implementa și altfel, de exemplu numărând toate bazele)

Abstracțiuni procedurale – Observații

- O abstracțiune este cu atât mai interesantă cu cât are mai multe utilizări
- Nume sugestive conduc la programe expresive
  (apropie programarea de modul în care oamenii gândesc)
- Separare între utilizarea funcției și implementarea ei
  - Folosim funcțiile de bibliotecă fără a le cunoaște implementarea, ca pe niște primitive (ex: cons, +, equal?)
  - În propriile programe, creăm noi abstracțiuni care pot fi folosite ca niște primitive de un dezvoltator ulterior

Abstracțiuni utile – șabloane comune

Exemple la calculator:

- Adunarea numerelor naturale de la a la b
- Aproximarea lui e conform seriei $e = 1/0! + 1/1! + 1/2! + 1/3! + \ldots$
- Aproximarea lui $\pi^2/8$ conform seriei $\pi^2/8 = 1/1^2 + 1/3^2 + 1/5^2 + \ldots$

Observații

- Funcțiile au foarte mult cod în comun
- Este util să identificăm un șablon mai abstract din care derivă toate cele 3 funcții
- Pașii de urmat: Recunoaștere șablon → Definire → Reutilizare

Recunoaștere → Definire → Reutilizare

Recunoașterea șablonului

- Reproducem porțiunile de cod comune
- Porțiunile care diferă devin variabile
- Ex: șablonul comun pentru `'(2 2)`, `'(3 3)`, `'(4 4)` este `(list x x)`

Definire

- Șablonului identificat la pasul anterior i se atribuie un nume sugestiv, care face programul ușor de înțeles

Reutilizare

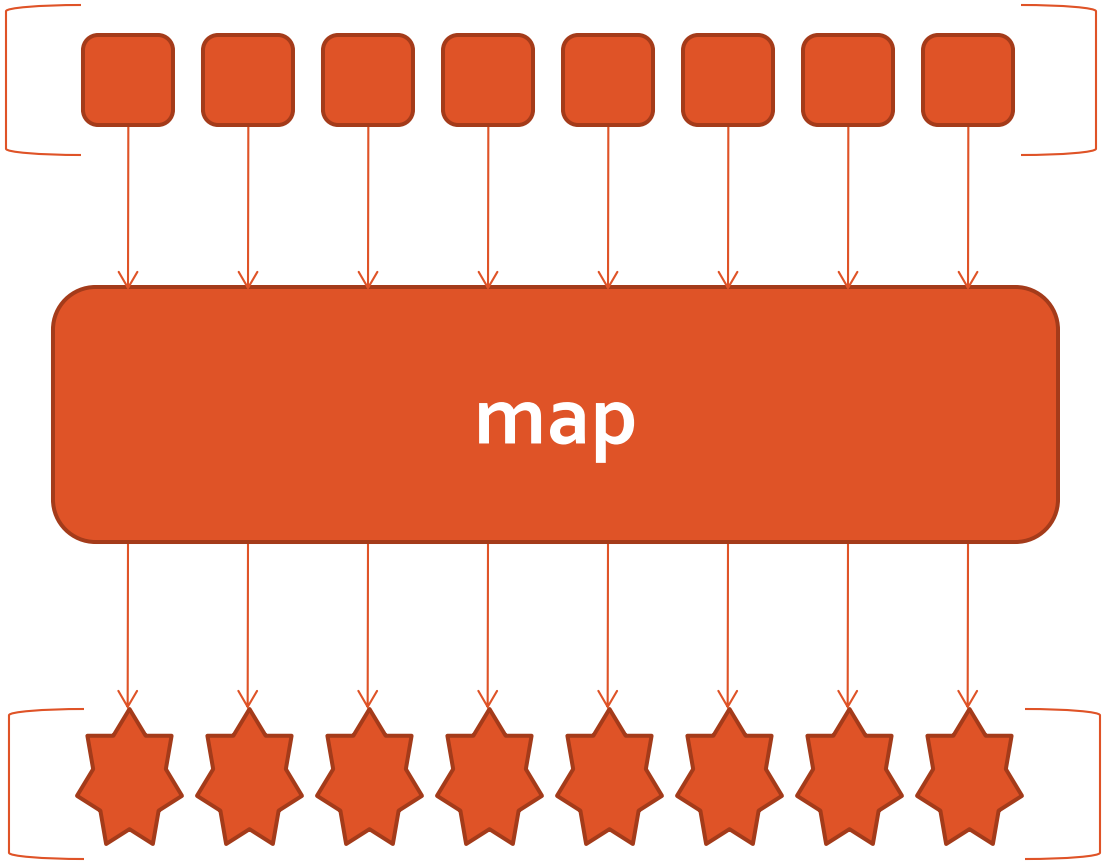
- Derivăm facil instanțe particulare ale șablonului, oricând avem ocazia

Abstractizare – Cuprins

- Abstractizare
- Abstractizarea proceselor
  - Funcții ca abstracțiuni procedurale care rezolvă o sarcină
  - Funcționale
- Abstractizarea datelor
  - Tipuri de date abstracte
  - Structuri
- Anatomia Racket

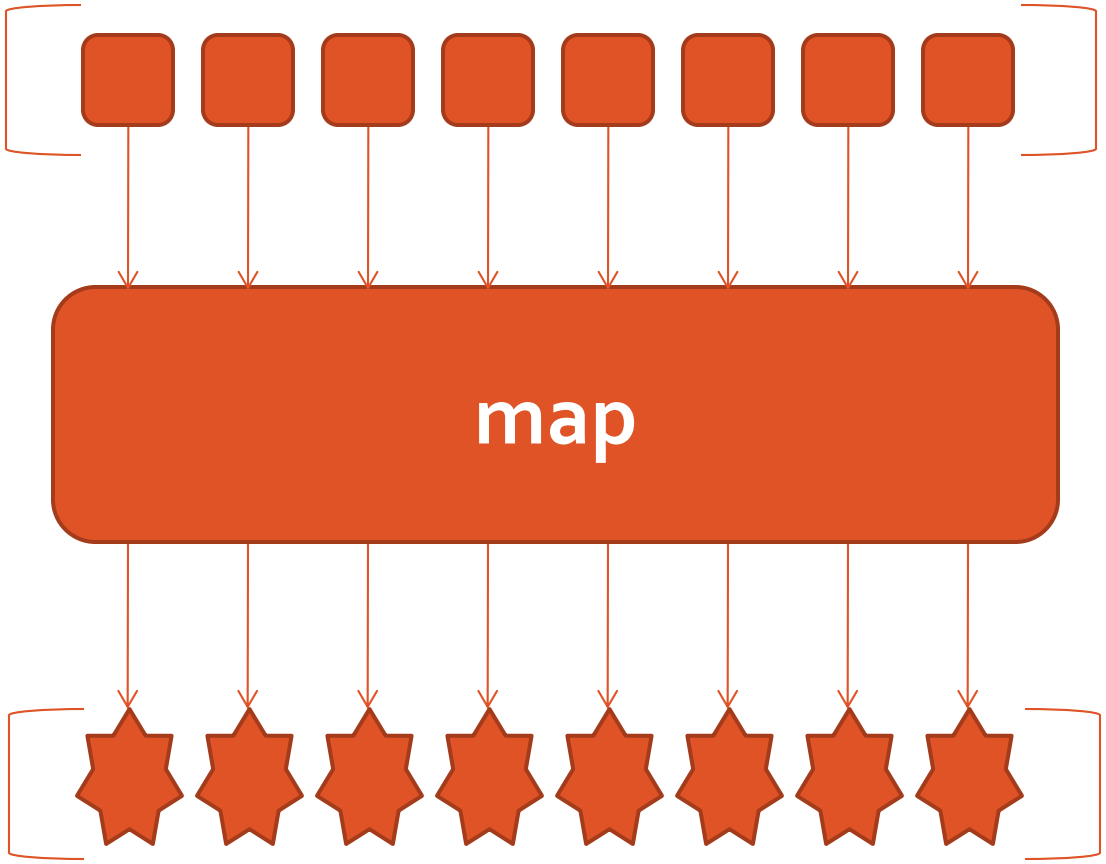
Funcționala map (map f L)

```
(map add1 (range 2 7))

(map sqr (range 2 7))

(map even? (range 2 7))

(map random (range 2 7))
```



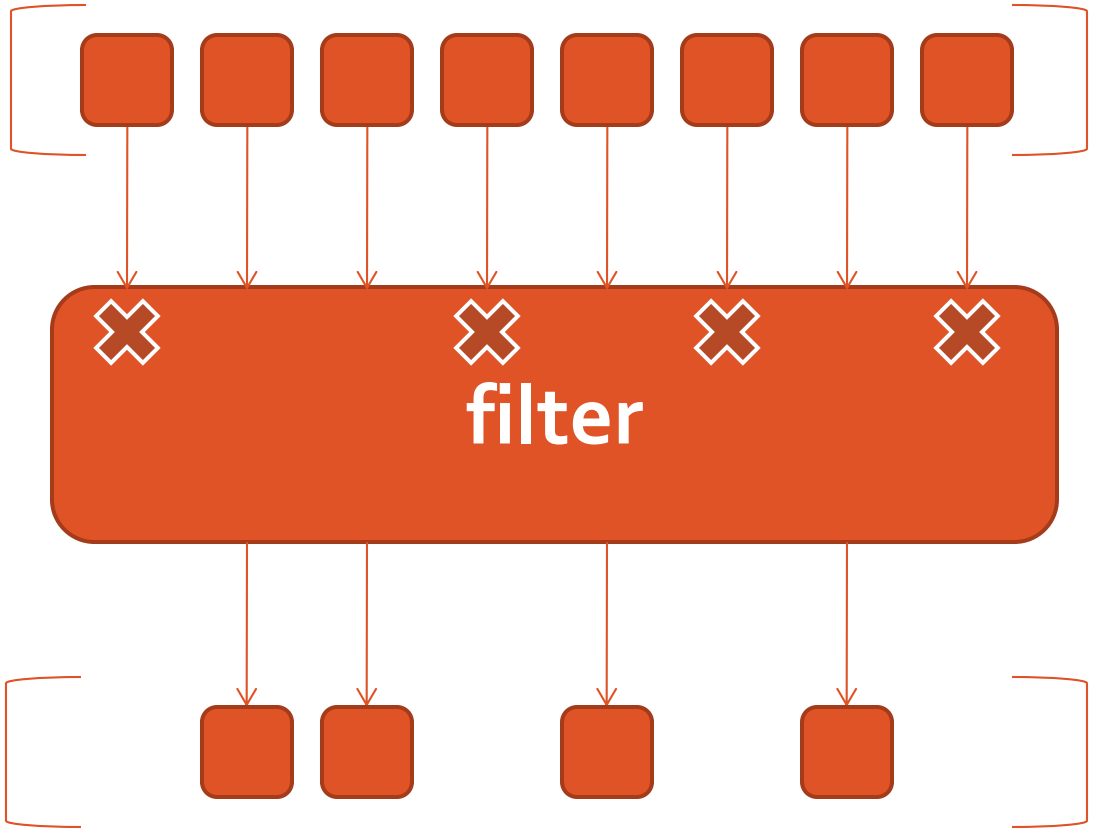
Funcționala map (map f L)

```
(map add1 (range 2 7))
    ;; '(3 4 5 6 7)
(map sqr (range 2 7))
    ;; '(4 9 16 25 36)
(map even? (range 2 7))
    ;; '(#t #f #t #f #t)
(map random (range 2 7))
    ;; surpriză ☺
```



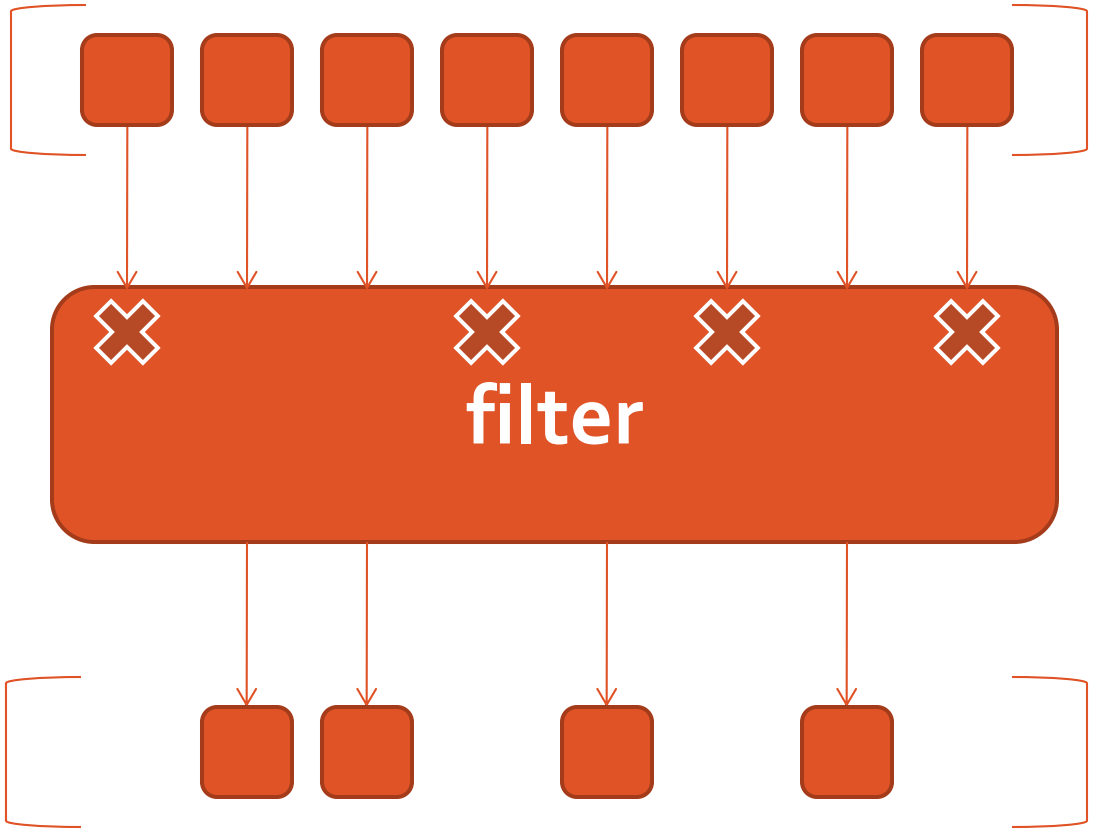
Funcționala filter (filter p L)

```
(filter even? (range 2 7))
```



Funcționala filter (filter p L)

```
(filter even? (range 2 7))

    ;; '(2 4 6)
```

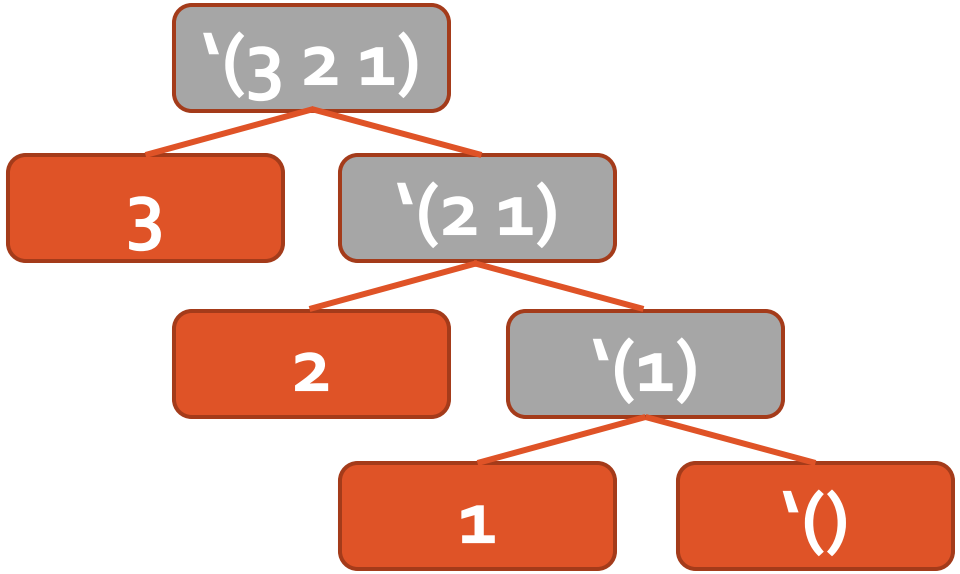


Funcționala foldl (fold left) (foldl f acc L)

- Folosește o funcție binară element-acumulator
- Elementele listei sunt prelucrate în ordinea stânga→dreapta
- Mai întâi se aplică funcția pe (first L) și acumulator, rezultând un nou acumulator
- Apoi pe (second L) și noul acumulator … etc.

Exemplu

```
(foldl cons '() '(1 2 3))
```

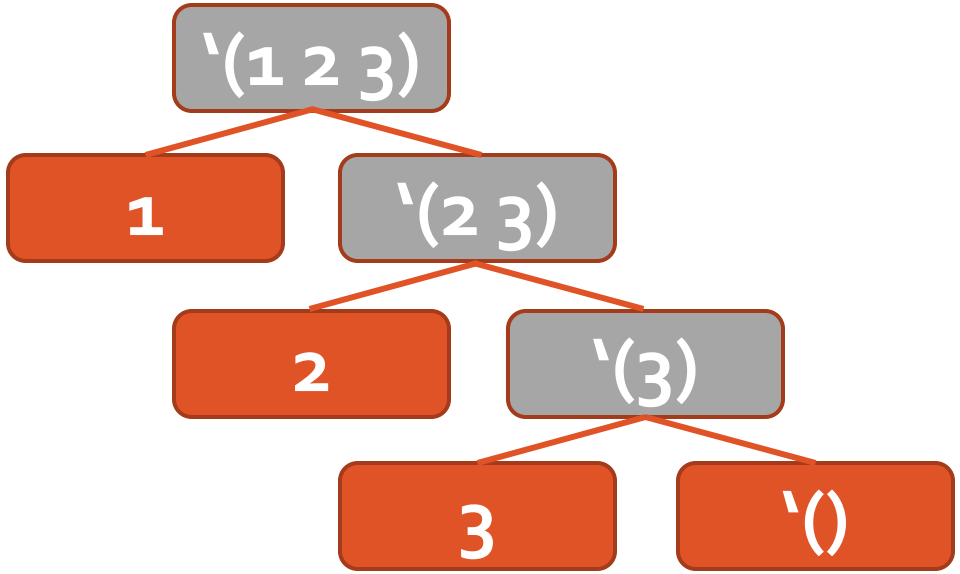


Funcționala foldr (fold right) (foldr f acc L)

- Folosește o funcție binară element-acumulator
- Elementele listei sunt prelucrate în ordinea dreapta→stânga
- Mai întâi se aplică funcția pe (last L) și acumulator, rezultând un nou acumulator
- Apoi pe penultimul și noul acumulator … etc.

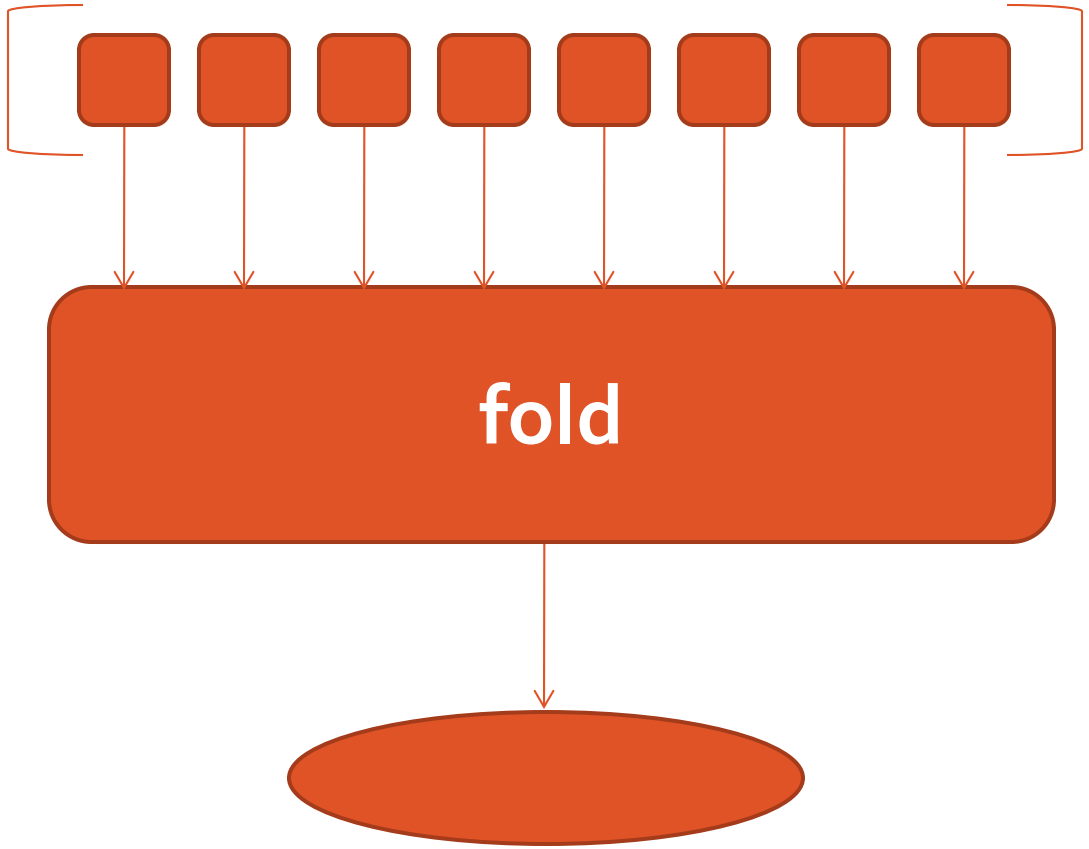
Exemplu

```
(foldr cons '() '(1 2 3))
```



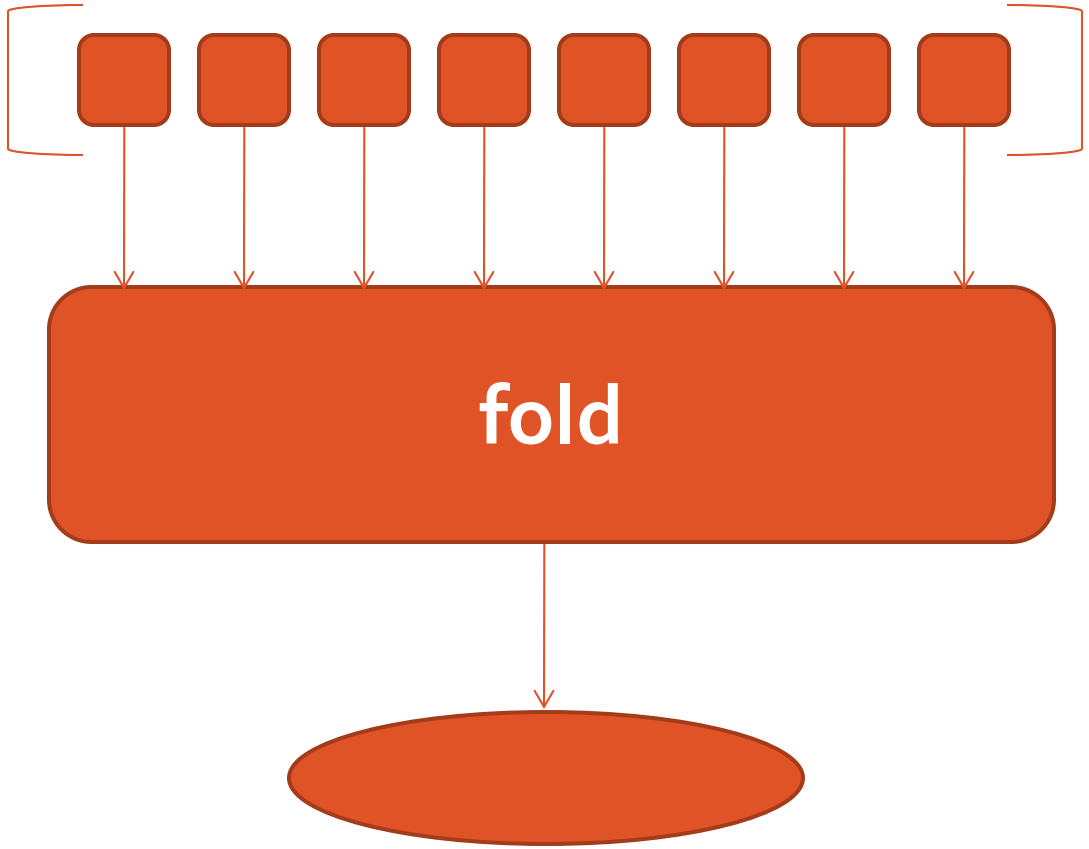
Funcționala foldl / foldr

```
(foldl + 0 (range 2 7))

(foldr max 0 (range 2 7))

(foldl cons '() (range 2 7))

(foldr cons '() (range 2 7))

(foldl list 1 (range 2 7))
```



Funcționala foldl / foldr

```
(foldl + 0 (range 2 7))
    ;; 20
(foldr max 0 (range 2 7))
    ;; 6
(foldl cons '() (range 2 7))
    ;; '(6 5 4 3 2)
(foldr cons '() (range 2 7))
    ;; '(2 3 4 5 6)
(foldl list 1 (range 2 7))
    ;; '(6 (5 (4 (3 (2 1)))))
```



Funcționala apply (apply f [...] L)

(apply f L) aplică funcția f pe argumentele din lista L

(apply f x y z L) aplică funcția f pe argumentele x y z și cele venite din lista L ... etc.

Exemple

```
(apply + (range 2 7))
(apply list -1 0 '(1 2 3))
(apply map list '((1 2 3)))
```

Funcționala apply (apply f [...] L)

(apply f L) aplică funcția f pe argumentele din lista L

(apply f x y z L) aplică funcția f pe argumentele x y z și cele venite din lista L ... etc.

Exemple

```
(apply + (range 2 7))                ;; 20
(apply list -1 0 '(1 2 3))           ;; '(-1 0 1 2 3)
(apply map list '((1 2 3)))          ;; '((1) (2) (3))
```

Funcționale (funcții de nivel înalt)

Funcționale (numite și funcții de nivel înalt)

- **Funcții care primesc ca argumente sau returnează funcții**
- map, filter, foldl, foldr, apply – funcționale Racket pentru procesarea listelor

Exemple la calculator:

- Maximul elementelor dintr-o listă de numere
- Extragerea inițialelor dintr-o listă de nume
- Extragerea pronumelor dintr-o listă de cuvinte
- Pătratele elementelor dintr-o listă de numere
- Numărul de elemente pare dintr-o listă
- Extragerea numerelor unei liste care sunt mai mici decât o valoare dată
- Concatenarea listelor dintr-o listă de liste

Funcționale (funcții de nivel înalt)

Funcționale (numite și funcții de nivel înalt)

- **Funcții care primesc ca argumente sau returnează funcții**
- map, filter, foldl, foldr, apply – funcționale Racket pentru procesarea listelor

Exemple la calculator:

- Maximul elementelor dintr-o listă de numere — fold
- Extragerea inițialelor dintr-o listă de nume — map
- Extragerea pronumelor dintr-o listă de cuvinte — filter
- Pătratele elementelor dintr-o listă de numere — map
- Numărul de elemente pare dintr-o listă — fold
- Extragerea numerelor unei liste care sunt mai mici decât o valoare dată — filter
- Concatenarea listelor dintr-o listă de liste — apply

Utilizare

map – calculează o listă de aceeași dimensiune, dar cu valori modificate

filter – calculează o listă mai scurtă, cu valori nemodificate

foldl/foldr – calculează un rezultat unic bazat pe toate valorile listei

apply – despachetează o listă ca argumente separate pentru o funcție

Observație

- Funcționalele de tip fold surprind procesul cel mai general
- map și filter pot fi implementate cu fold
- Pentru expresivitate, este de dorit să folosim
  - map când avem de transformat fiecare element (cuvânt cheie: fiecare)
  - filter când avem de selectat anumite elemente (cuvânt cheie: selecție)

și nu fold peste tot.

Extensie Racket pentru map/fold

- În Racket, map, foldl și foldr pot procesa n liste în paralel, nu doar una
  - Toate listele trebuie să aibă aceeași lungime
  - Funcția dată ca parametru trebuie să fie
    - n-ară pentru map
    - (n+1)-ară pentru fold

  } astfel încât să își ia câte un argument din fiecare listă

```
(map + '(1 2 3) '(10 20 30) '(100 200 300))
(foldl (λ (x y acc) (cons (cons x y) acc))
       '()
       '(a b)
       '(1 2))
```

Extensie Racket pentru map/fold

- În Racket, map, foldl și foldr pot procesa n liste în paralel, nu doar una
  - Toate listele trebuie să aibă aceeași lungime
  - Funcția dată ca parametru trebuie să fie
    - n-ară pentru map
    - (n+1)-ară pentru fold

  } astfel încât să își ia câte un argument din fiecare listă

```
(map + '(1 2 3) '(10 20 30) '(100 200 300))     ; '(111 222 333)
(foldl (λ (x y acc) (cons (cons x y) acc))      ; '((b . 2) (a . 1))
       '()
       '(a b)
       '(1 2))
```

Abstractizare – Cuprins

- Abstractizare
- Abstractizarea proceselor
  - Funcții ca abstracțiuni procedurale care rezolvă o sarcină
  - Funcționale
- Abstractizarea datelor
  - Tipuri de date abstracte
  - Structuri
- Anatomia Racket

Abstractizarea datelor

- A abstractiza datele = a separa:
  - semantica (ce sunt datele și ce putem face cu ele)
  - reprezentarea fizică (cum sunt datele stocate în memorie)
- Bariera de abstractizare = zid virtual de protecție între cele două lumi:
  - Utilizatorul
    - Folosește datele ca pe niște obiecte abstracte, prin interfața (operatorii) acestora
    - Codul este robust: imun la schimbarea reprezentării datelor
  - Proiectantul
    - Gestionează detaliile de implementare
    - Poate optimiza performanța fără a afecta programele utilizatorului
  - O fisură în bariera de abstractizare (abstraction leak) descrie situația nedorită când detaliile din lumea proiectantului devin vizibile în lumea utilizatorului

Abstractizare – Cuprins

- Abstractizare
- Abstractizarea proceselor
  - Funcții ca abstracțiuni procedurale care rezolvă o sarcină
  - Funcționale
- Abstractizarea datelor
  - Tipuri de date abstracte
  - Structuri
- Anatomia Racket

Tipuri de date abstracte (TDA)

- Ca și procesele frecvente, tipurile de date frecvent utilizate sunt abstractizate
- Bariera de abstractizare
  - Deasupra: utilizatorul folosește TDA prin constructori și operatori,
    care se comportă conform contractului precizat în axiome
  - Dedesubt: proiectantul optează pentru orice implementare care respectă axiomele,
    putând optimiza implementarea în timp
- Abstractizare ierarhică
  - Tipuri primitive ⇒ tipuri complexe ⇒ tipuri și mai complexe
  - Fiecare nivel este material de construcție pentru nivelul următor
  - Interfața face utilizarea tipurilor complexe la fel de intuitivă ca utilizarea primitivelor
    ⇒ Astfel controlăm complexitatea intelectuală a unui program mare

Exemplu TDA – Numere complexe

Constructori

make-complex : R x R -> C

Operatori

get-real : C -> R

get-imag : C -> R

get-magnitude : C -> R

add-c : C x C -> C

etc.

Reprezentări posibile

coordonate carteziene sau polare

coordonate ca liste sau ca perechi

etc.

Utilizare numere complexe

make-complex get-real get-imag get-magnitude add-c

Implementare numere complexe (ex: perechi, dar irelevant la nivel superior)

cons car cdr

Implementare perechi (nu o știm și nu ne interesează)

Exemplu TDA – BST (Binary Search Trees)

Constructori

- empty-bst : -> BST
- make-bst : BST x Elem x BST -> BST

Operatori

- left : BST -> BST
- right : BST -> BST
- key : BST -> Elem
- bst-empty? : BST -> Bool
- insert-bst : Elem x BST -> BST
- list->bst : List -> BST

Reprezentări posibile

- liste
- perechi
- structuri

Abstractizare – Cuprins

- Abstractizare
- Abstractizarea proceselor
  - Funcții ca abstracțiuni procedurale care rezolvă o sarcină
  - Funcționale
- Abstractizarea datelor
  - Tipuri de date abstracte
  - Structuri
- Anatomia Racket

Structuri

- Feature Racket pentru construcția de TDA-uri cu barieră de abstractizare implicită
- Sintaxa: `(define-struct nume-struct (câmp1 câmp2 ...))`
  - Interfața se definește automat, minimizând riscul fisurilor (abstraction leaks):
    - Constructor: `make-nume-struct`
    - Predicat de tip: `nume-struct?`
    - Selectori: `nume-struct-câmp1, nume-struct-câmp2,` ... etc.

```
(define-struct complex-number (real imag))
(define (add-c-n C1 C2)
  (make-complex-number
   (+ (complex-number-real C1) (complex-number-real C2))
   (+ (complex-number-imag C1) (complex-number-imag C2))))
```

Abstractizare – Cuprins

- Abstractizare
- Abstractizarea proceselor
  - Funcții ca abstracțiuni procedurale care rezolvă o sarcină
  - Funcționale
- Abstractizarea datelor
  - Tipuri de date abstracte
  - Structuri
- Anatomia Racket

Anatomia Racket

- Primitive
  - Tipuri primitive (numere, valori boolene, simboluri, liste) și interfețele lor
- Mijloace de combinare
  - Perechi și liste – asamblează valori elementare producând valori mai complexe
  - Aplicația de funcție – combină procese și date pentru a produce rezultate noi
- Mijloace de abstractizare
  - Abstractizarea proceselor:
    - lambda, define – numesc și izolează un tipar de calcul
    - Currying – funcțiile devin șabloane instanțiabile
    - compose – abstractizează fluxul de calcul (ex: continuation passing style)
  - Abstractizarea datelor:
    - define – funcții care definesc interfața TDA-urilor
    - define-struct – creează TDA-uri noi cu barieră de abstractizare implicită

Rezumat

Abstractizare proces

Funcționale

Utilizare map

Utilizare filter

Utilizare fold

Utilizare apply

Abstractizare date

Structuri

Rezumat

Abstractizare proces: izolarea unui șablon de calcul pentru a fi utilizat ca o primitivă

Funcționale

Utilizare map

Utilizare filter

Utilizare fold

Utilizare apply

Abstractizare date

Structuri

Rezumat

Abstractizare proces: izolarea unui șablon de calcul pentru a fi utilizat ca o primitivă

Funcționale: funcții care primesc ca argumente sau returnează funcții

Utilizare map

Utilizare filter

Utilizare fold

Utilizare apply

Abstractizare date

Structuri

Rezumat

Abstractizare proces: izolarea unui șablon de calcul pentru a fi utilizat ca o primitivă

Funcționale: funcții care primesc ca argumente sau returnează funcții

Utilizare map: (map f L) aplică f pe fiecare element din L

Utilizare filter

Utilizare fold

Utilizare apply

Abstractizare date

Structuri

Rezumat

Abstractizare proces: izolarea unui șablon de calcul pentru a fi utilizat ca o primitivă

Funcționale: funcții care primesc ca argumente sau returnează funcții

Utilizare map: (map f L) aplică f pe fiecare element din L

Utilizare filter: (filter p L) păstrează doar elementele din L care satisfac predicatul p

Utilizare fold

Utilizare apply

Abstractizare date

Structuri

Rezumat

Abstractizare proces: izolarea unui șablon de calcul pentru a fi utilizat ca o primitivă

Funcționale: funcții care primesc ca argumente sau returnează funcții

Utilizare map: (map f L) aplică f pe fiecare element din L

Utilizare filter: (filter p L) păstrează doar elementele din L care satisfac predicatul p

Utilizare fold: (foldl/foldr f acc L) parcurge L de la stânga/dreapta, aplicând (f x acc), (f y acc')…

Utilizare apply

Abstractizare date

Structuri

Rezumat

Abstractizare proces: izolarea unui șablon de calcul pentru a fi utilizat ca o primitivă

Funcționale: funcții care primesc ca argumente sau returnează funcții

Utilizare map: (map f L) aplică f pe fiecare element din L

Utilizare filter: (filter p L) păstrează doar elementele din L care satisfac predicatul p

Utilizare fold: (foldl/foldr f acc L) parcurge L de la stânga/dreapta, aplicând (f x acc), (f y acc')…

Utilizare apply: (apply f […] L) aplică f pe argumentele care urmează, despachetând lista L

Abstractizare date

Structuri

Rezumat

Abstractizare proces: izolarea unui șablon de calcul pentru a fi utilizat ca o primitivă

Funcționale: funcții care primesc ca argumente sau returnează funcții

Utilizare map: (map f L) aplică f pe fiecare element din L

Utilizare filter: (filter p L) păstrează doar elementele din L care satisfac predicatul p

Utilizare fold: (foldl/foldr f acc L) parcurge L de la stânga/dreapta, aplicând (f x acc), (f y acc')…

Utilizare apply: (apply f […] L) aplică f pe argumentele care urmează, despachetând lista L

Abstractizare date: separarea semanticii datelor de detaliile reprezentării fizice

Structuri

Rezumat

Abstractizare proces: izolarea unui șablon de calcul pentru a fi utilizat ca o primitivă

Funcționale: funcții care primesc ca argumente sau returnează funcții

Utilizare map: (map f L) aplică f pe fiecare element din L

Utilizare filter: (filter p L) păstrează doar elementele din L care satisfac predicatul p

Utilizare fold: (foldl/foldr f acc L) parcurge L de la stânga/dreapta, aplicând (f x acc), (f y acc')…

Utilizare apply: (apply f […] L) aplică f pe argumentele care urmează, despachetând lista L

Abstractizare date: separarea semanticii datelor de detaliile reprezentării fizice

Structuri: TDA-uri cu interfața generată automat (constructor, predicat de tip, selectori)