Programare declarativă

Categorii[1]

Ioana Leuștean
Traian Florin Șerbănuță

Departamentul de Informatică, FMI, UNIBUC
traian.serbanuta@unibuc.ro

7 decembrie 2017

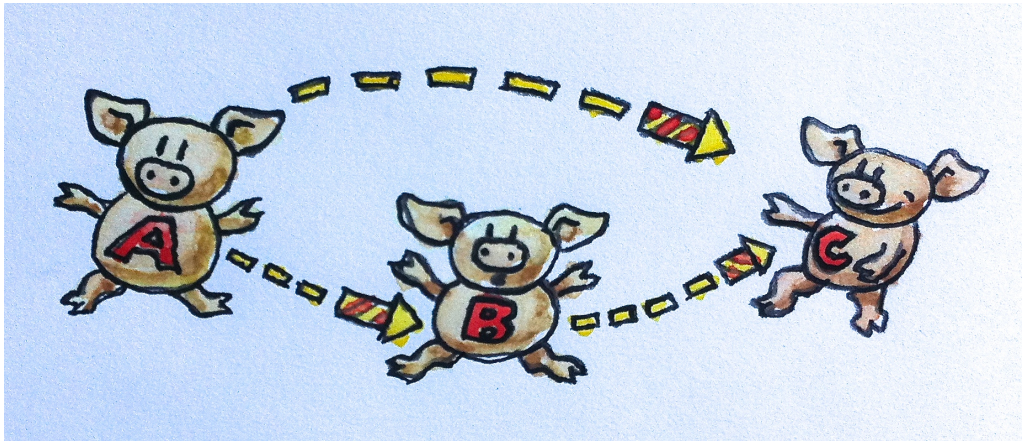
[1] bazat pe Categories for programmers

Categorii și Functori

Categorii

O categorie $\mathbb{C}$ este dată de:

- O clasă $|\mathbb{C}|$ a obiectelor
- Pentru oricare două obiecte $A, B \in |\mathbb{C}|$,
  o mulțime $\mathbb{C}(A, B)$ a săgeților „de la A la B”
  $f \in \mathbb{C}(A, B)$ poate fi scris ca $f : A \to B$
- Pentru orice obiect A o săgeată $id_A : A \to A$ numită identitatea lui A
- Pentru orice obiecte A, B, C, o operație de compunere a săgeților
  $\circ : \mathbb{C}(B, C) \times \mathbb{C}(A, B) \to \mathbb{C}(A, C)$



Bartosz Milewski — Category: The Essence of Composition

- Compunerea este asociativă și are element neutru id

Exemplu: Categoria $\mathbb{S}$et

- Obiecte: mulțimi
- Săgeți: funcții
- Identități: Funcțiile identitate
- Compunere: Compunerea funcțiilor

Exemplu: Categoria Hask

- Obiectele: tipuri
- Săgețiile: funcții între tipuri

```
f :: A -> B
```

- Identități: funcția polimorfică **id**

```
Prelude> :t id
id :: a -> a
```

- Compunere: funcția polimorfică (.)

```
Prelude> :t (.)
(.) :: (b -> c) -> (a -> b) -> a -> c
```

Subcategorii ale lui Hask date de tipuri parametrizate

- Obiecte: o clasă restânsă de tipuri din $|\mathbb{H}\mathrm{ask}|$
  - Exemplu: tipuri de forma [a]
- Săgeți: toate funcțiile din $\mathbb{H}$ask între tipurile obiecte
  - Exemple: **concat** :: [[a]] –> [a], **words** :: [**Char**] –> [**String**], **reverse** :: [a] –> [a]

Exemple

Liste	obiecte: tipuri de forma [a]
Optiuni	obiecte: tipuri de forma Maybe a
Arbori	obiecte: tipuri de forma Arbore a
Comenzi I/O	obiecte: tipuri de forma IO a
Funcții de sursă t	obiecte: tipuri de forma t -> a

De ce categorii?

(Des)compunerea este esența programării

- Am de rezolvat problema P
- O descompun în subproblemele $P_1, \ldots P_n$
- Rezolv problemele $P_1, \ldots P_n$ cu programele $p_1, \ldots p_n$
  - Eventual aplicând recursiv procedura de față
- Compun rezolvările $p_1, \ldots p_n$ într-o rezolvare p pentru problema inițială

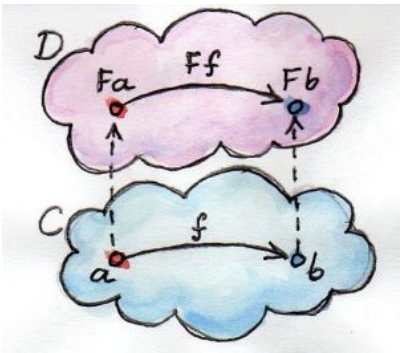
Categoriile rezolvă problema compunerii

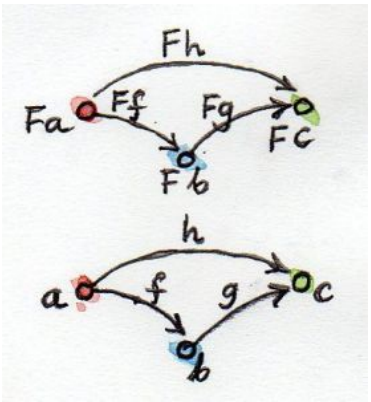
- Ne forțează să abstractizăm datele
- Se poate acționa asupra datelor doar prin săgeți (metode?)
- Forțează un stil de compunere independent de structura obiectelor

Functori

Date fiind două categorii $\mathbb{C}$ și $\mathbb{D}$, un functor $F : \mathbb{C} \to \mathbb{D}$ este dat de

- O funcție $F : |\mathbb{C}| \to |\mathbb{D}|$ de la obiectele lui $\mathbb{C}$ la cele ale lui $\mathbb{D}$
- Pentru orice $A, B \in |\mathbb{C}|$, o funcție $F : \mathbb{C}(A, B) \to \mathbb{D}(F(A), F(B))$
- Compatibilă cu identitățile și cu compunerea
  - $F(id_A) = id_{F(A)}$ pentru orice A
  - $F(g \circ f) = F(g) \circ F(f)$ pentru orice $f : A \to B, g : B \to C, h = g \circ f$





Bartosz Milewski — Functors

Functori în Haskell

În general un functor $F : \mathbb{C} \to \mathbb{D}$ este dat de

- O funcție $F : |\mathbb{C}| \to |\mathbb{D}|$ de la obiectele lui $\mathbb{C}$ la cele ale lui $\mathbb{D}$
- Pentru orice $A, B \in |\mathbb{C}|$, o funcție $F : \mathbb{C}(A, B) \to \mathbb{D}(F(A), F(B))$
- Compatibilă cu identitățile și cu compunerea
  - $F(id_A) = id_{F(A)}$ pentru orice A
  - $F(g \circ f) = F(g) \circ F(f)$ pentru orice $f : A \to B, g : B \to C, h = g \circ f$

În Haskell o instanță **Functor** m este dată de

- Un tip m a pentru orice tip a (deci m trebuie sa fie tip parametrizat)
- Pentru orice două tipuri a și b, o funcție

```
fmap :: (a -> b) -> (m a -> m b)
```

- Compatibilă cu identitățile și cu compunerea

```
fmap id == id
fmap (g . f) == fmap g . fmap f
```

pentru orice f :: a -> b și g :: b -> c