Programowanie funkcyjne

(wykład 7.)

Roman Dębski

Instytut Informatyki, AGH

21 grudnia 2021

Plan wykładu

Plan wykładu

Teoria kategorii jako abstrakcyjna teoria funkcji

"Teoria kategorii jako ogólny dział algebry wyrosła z prac Samuela Eilenberga i Saundersa MacLane'a: pionierską pracą jest tu General theory of natural equivalences, Transactions of the American Mathematical Society 58 (1945), str. 231-294 - autorzy wprowadzili tam pojęcie kategorii, funktora i naturalnej transformacji funktorów. [...] Od tamtego czasu zarówno język, jak i metody teorii kategorii spowodowały ujednolicenie i uproszczenie wielu pojęć algebry, topologii, geometrii algebraicznej, analizy funkcjonalnej, a także, ostatnimi czasy, logiki matematycznej i informatyki teoretycznej"

"Czy można prezentować różnorodne teorie matematyczne, badając jedynie własności przekształceń obiektów matematycznych będących przedmiotem zainteresowania danej teorii?"

"Teoria kategorii bada wspólne, uniwersalne własności zbiorów, grup, przestrzeni topologicznych, przestrzeni wektorowych, częściowych porządków, i tak dalej, wszystko w języku przekształceń danej struktury".

— `http://wazniak.mimuw.edu.pl/index.php?title=Teoria_kategorii_dla_informatyków`

Teoria kategorii: perspektywa programisty

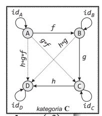
"'Category theory is a treasure trove of extremely useful programming ideas. Haskell programmers have been tapping into this resource for a long time, and the ideas are slowly being assimilated into other languages, but this process is too slow. We need to speed it up.

[...] category theory is the kind of Maths that is particularly well suited for the minds of programmers. This is because category theory – rather than dealing with particulars – deals with structure. It deals with the kind of structure that makes programs composable.

Composition is at the very root of category theory – it is a part of the definition of the category itself. [...] composition is the essence of programming. We have been composing things forever [...] Some time ago the principles of structural programming revolutionised programming because they made blocks of code composable. Then came object oriented programming which is all about composing objects. Functional programming is not only about composing functions and algebraic data structures — it makes concurrency composable – something that is virtually impossible with other programming paradigms."

— `https://bartoszmilewski.com/2014/10/28/category-theory-for-programmers-the-preface/`

Definicja kategorii



Kategoria $\mathbf{C}$ składa się z

- obiektów $A, B, C, ...$,
- morfizmów $f, g, h, ...$,
- dwóch operacji: dom i cod, przypisujących każdemu morfizmowi f obiekty $\mathrm{dom}(f)$ i $\mathrm{cod}(f)$,
- operacji id (lub $\mathbb{1}$) przypisującej każdemu obiektowi A morfizm id_A ($\mathbb{1}_A$) nazywany identycznością,
- operacji $\circ$ przypisującej każdej parze morfizmów f, g takich, że $\mathrm{cod}(g) = \mathrm{dom}(f)$ morfizm $f \circ g$ nazywany złożeniem

spełniających następujące aksjomaty:

- $\mathrm{dom}(id_A) = A = \mathrm{cod}(id_A)$; $\mathrm{dom}(f \circ g) = \mathrm{dom}(g)$; $\mathrm{cod}(f \circ g) = \mathrm{cod(f)}$
- $f \circ id_A = f = id_B \circ f$, gdzie $A = \mathrm{dom}(f)$ oraz $B = \mathrm{cod}(f)$
- jeśli f i g są składalne oraz g i h są składalne, to $(f \circ g) \circ h = f \circ (g \circ h)$

Uwaga: kolekcję obiektów kategorii $\mathbf{C}$ często oznacza się jako $\mathbf{C}_0$ zaś kolekcję jej morfizmów jako $\mathbf{C}_1$

— `http://wazniak.mimuw.edu.pl/index.php?title=Teoria_kategorii_dla_informatyków`

**** można przyjąć, że kategorią jest "cokolwiek", co spełnia powyższą definicję :)***

Przykłady kategorii

- $\mathbf{Set}$: obiektami są zbiory, morfizmami funkcje.
  Uwaga: w teorii mnogości funkcja jest zdefiniowana jako zbiór par uporządkowanych takich, że $((x, y) \in f \ \wedge \ (x, y') \in f) \implies y = y'$ (1). Aby traktować funkcje jako morfizmy, musimy precyzyjnie znać $\mathrm{dom}(f)$ i $\mathrm{cod}(f)$. Formalnie, w języku teorii mnogości morfizmem będzie trójka (X,f,Y) taka, że $f \subseteq X \times Y$ spełnia równanie (1) oraz $x \in X \implies (\exists y \in Y \mid (x, y) \in f)$. Wtedy dom jest projekcją na pierwszą współrzędną $(X, f, Y) \mapsto X$, a cod projekcją na trzecią współrzędną.
- **Kategorie skończone**: $\mathbf{0}$, $\mathbf{1}$, $\mathbf{2}$,...
  Uwaga: skończoność dotyczy liczby istniejących morfizmów, choć nazwy tych kategorii odnoszą się do liczby obiektów.
- **Kategorie dyskretne**: nie ma innych morfizmów niż identyczności
  Uwaga: kategorie dyskretne możemy utożsamiać ze zbiorami (obiekty jako elementy zbioru)
- **Monoidy**: kategorie z jednym obiektem
  Niech $(M, *, e)$ będzie monoidem (e jest jego "*jedynką*"). Wówczas biorąc M jako jedyny obiekt, zaś elementy M jako morfizmy (z dziedziną i kodziedziną M), a działanie $*$ jako złożenie morfizmów, otrzymujemy kategorię.
- **Hask**: obiektami są typy, a morfizmami funkcje języka Haskell.

Izomorfizm, kategorie małe i lokalnie małe

Izomorfizm

Niech $\mathbf{C}$ będzie dowolną kategorią. Morfizm $f \colon A \to B$ jest izomorfizmem, jeśli istnieje morfizm $g \colon B \to A$ taki, że $f \circ g = id_B$ oraz $g \circ f = id_A$. Morfizm g nazywa się morfizmem odwrotnym do f. Jeśli dla obiektów A,B kategorii $\mathbf{C}$ istnieje izomorfizm $f \colon A \to B$, to obiekty A i B nazywamy izomorficznymi, co zapisujemy jako $A \cong B$.

Kategoria mała

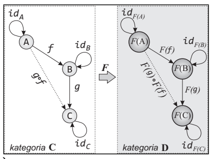
Kategorię $\mathbf{C}$ nazywamy małą, jeśli kolekcja wszystkich obiektów $\mathbf{C}_0$ i morfizmów $\mathbf{C}_1$ kategorii $\mathbf{C}$ są zbiorami. W przeciwnym przypadku $\mathbf{C}$ jest duża.

Kategoria lokalnie mała

Kategorię $\mathbf{C}$ nazywamy lokalnie małą, jeśli dla każdej pary obiektów A, B z $\mathbf{C}$ kolekcja $\mathrm{Hom}_{\mathbf{C}}(A, B) = \{f \in \mathbf{C}_1 \mid f \colon A \to B\}$ jest zbiorem. O takim zbiorze mówimy w skrócie homset (podobnie jak o zbiorze częściowo uporządkowanym przyjęło się mówić: poset)

uwaga na pojęcie: z dokładnością do izomorfizmu / up to isomorphism

Funktory



Funktor (funktor kowariantny)

Funktor (kowariantny) $F\colon \mathbf{C} \to \mathbf{D}$ z kategorii $\mathbf{C}$ do kategorii $\mathbf{D}$ jest dany poprzez:

- operację przypisującą jednoznacznie każdemu obiektowi $X \in \mathbf{C}_0$ obiekt $F(X) \in \mathbf{D}_0$
- operację przypisującą jednoznacznie każdemu morfizmowi $f \in \mathbf{C}_1$ morfizm $F(f) \in \mathbf{D}_1$ w ten sposób, że:
  jeśli $f\colon X \to Y$ w $\mathbf{C}$, to $F(f)\colon F(X) \to F(Y)$ w $\mathbf{D}$,
  $F(id_X) = id_{F(X)}$
  $F(f \circ g) = F(f) \circ F(g)$.

Uwaga: funktor typu $\mathbf{C}^{op} \to \mathbf{D}$ nazywamy często *funktorem kontrawariantnym*, aby podkreślić, że jego dziedziną jest *kategoria dualna* do $\mathbf{C}$

Transformacje naturalne

$$\begin{tikzcd} A \arrow[d, "f"'] & F(A) \arrow[r, "\eta_A"] \arrow[d, "F(f)"'] & G(A) \arrow[d, "G(f)"] \\ B & F(B) \arrow[r, "\eta_B"'] & G(B) \end{tikzcd}$$

Transformacja naturalna

Dla równoległych funktorów* $F, G\colon \mathbf{C} \to \mathbf{D}$ *transformacją naturalną* η nazywamy przekształcenie przypisujące każdemu obiektowi $A \in \mathbf{C}$ strzałkę $\eta_A\colon F(A) \to G(A)$ w $\mathbf{D}$ takie, że dla dowolnego morfizmu $f\colon A \to B$ w $\mathbf{C}$ powyższy diagram komutuje. Rodzinę strzałek $(\eta_A)_{A \in \mathbf{C}_0}$ nazywamy *komponentami transformacji naturalnej* η.
Uwaga: Jeśli każdy komponent transformacji naturalnej η jest izomorfizmem w kategorii $\mathbf{D}$, wówczas transformację η nazywamy *naturalnym izomorfizmem*.

***** o tej samej dziedzinie i przeciwdziedzinie

Uwaga: każda funkcja polimorficzna Haskella da się opisać jako pewna transformacja naturalna w odpowiedniej kategorii funktorów (np. `Hask`), np. η - `safeHead :: [a] -> Maybe a`; F - `[]`; G - `Maybe`; $C = D =$ `Hask`

Monady

$$\begin{tikzcd} T \arrow[r, "\eta_T"] \arrow[dr, "id_T"'] & TT \arrow[d, "\mu"] & T \arrow[l, "T_\eta"'] \arrow[dl, "id_T"] \\ & T & \end{tikzcd} \qquad \begin{tikzcd} TTT \arrow[r, "T_\mu"] \arrow[d, "\mu_T"'] & TT \arrow[d, "\mu"] \\ TT \arrow[r, "\mu"'] & T \end{tikzcd}$$

Monada

Monadą w kategorii $\mathbf{C}$ nazywamy trójkę $\mathbb{T} = (T, \eta, \mu)$, gdzie $T\colon \mathbf{C} \to \mathbf{C}$ jest funktorem (endo-funktorem), zaś $\eta\colon id_{\mathbf{C}} \to T$ i $\mu\colon TT \to T$ są transformacjami naturalnymi takimi, że:

- $\mu \circ \eta_T = id = \mu \circ T_\eta$,
- $\mu \circ \mu_T = \mu \circ T_\mu$,

tzn. takimi, że powyższe diagramy komutują.

Plan wykładu

Warto poczytać o

- Idris (`http://www.idris-lang.org`)
- Agda (`http://wiki.portal.chalmers.se/agda/pmwiki.php`)
- Coq (`https://coq.inria.fr`)

Bibliografia

- `http://wazniak.mimuw.edu.pl/index.php?title=Teoria_kategorii_dla_informatyków`
- `https://bartoszmilewski.com/2014/10/28/category-theory-for-programmers-the-preface/`
- `http://www.math.mcgill.ca/triples/Barr-Wells-ctcs.pdf`
- `http://math.mit.edu/~dspivak/teaching/sp13/CT4S--static.pdf`
- `https://en.wikibooks.org/wiki/Haskell/Category_theory`