UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – 1. stopnja

Mitja Rozman

Monade v funkcijskem programiranju

Delo diplomskega seminarja

Mentor: prof. dr. Andrej Bauer

Ljubljana, 2015

Kazalo

Monade v funkcijskem programiranju

Povzetek

V diplomskem delu so povzeti osnovni pojmi teorije kategorij. Definiran je pojem monade in podano je nekaj obsežnejših zgledov monad. Predstavljen je koncept funkcijskega programiranja v Haskellu in naveden je primer funkcije v Haskellu. Kot glavni rezultat je podana povezava med osnovno definicijo monade, definirano s pomočjo teorije kategorij, ter monado, definirano v Haskellu.

Monads in functional programming

Abstract

In this work we go through basic concepts of category theory. Monad is defined and some examples are given. There is also an introduction to functional programming in Haskell and an example of a function in Haskell. The main result is a link between the basic monad definition in the category theory and the one given in Haskell.

Math. Subj. Class. (2010): 18C15, 68N18

Ključne besede: kategorija, funktor, naravna transformacija, monada, Haskell, funkcijsko programiranje

Keywords: category, functor, natural transformation, monad, Haskell, functional programming

1. Osnovni pojmi teorije kategorij

1.1. **Kategorija.** Kategorija je struktura, s katero proučujemo morfizme, ti so ponavadi kar preslikave. Izkaže se, da se kategorije pojavijo na mnogih področjih matematike. Tako je na primer na področju topologije kompozitum zveznih funkcij spet zvezna funkcija, pri teoriji grup je kompozitum homomorfizmov spet homomorfizem in obakrat velja asociativnost. Ko te teorije obravnavamo s pomočjo teorije kategorij, dobimo popolnoma posplošene abstraktne rezultate, ki jih nato spet lahko prenesemo na konkretno teorijo. Na tak način proučujemo več različnih teorij s pomočjo splošnejše.

Definicija 1.1. *Kategorija* je struktura, ki zadošča naslednjim trem pogojem:

(1) Imamo razred objektov, ki ga označimo s $\underline{\mathbf{C}}$. Objekte tega razreda označimo z $A, B, \ldots \in \underline{\mathbf{C}}$.

(2) Za poljubna objekta $A, B \in \underline{\mathbf{C}}$ obstaja množica, ki jo označimo s $\mathrm{Hom}(A, B)$. Elemente te množice označimo $f, g, \ldots \in \mathrm{Hom}(A, B)$.

Za elemente $\mathrm{Hom}(A, B)$ v splošnem ne privzamemo, da so preslikave. Kljub temu pišemo: $f \in \mathrm{Hom}(A, B) \Leftrightarrow f : A \longrightarrow B \Leftrightarrow A \xrightarrow{f} B$ in jih imenujemo *morfizmi*, včasih tudi "puščice".

(3) Za poljubne tri objekte $A, B, C \in \underline{\mathbf{C}}$ obstaja operacija kompozitum $\circ$, ki morfizmoma $f : A \longrightarrow B$ in $g : B \longrightarrow C$ priredi morfizem $g \circ f : A \longrightarrow C$ in ima lastnosti:
   (a) *Asociativnost*:
   Za poljubne morfizme $f : A \longrightarrow B$, $g : B \longrightarrow C$ in $h : C \longrightarrow D$ velja: $h \circ (g \circ f) = (h \circ g) \circ f$, kjer je oboje $A \longrightarrow D$.
   (b) *Enota*:
   Za vsak $A \in \underline{\mathbf{C}}$ obstaja natanko določen morfizem id_A z lastnostma:
   - Za vsak $f : B \longrightarrow A$ je $id_A \circ f = f$.
   - Za vsak $g : A \longrightarrow C$ je $g \circ id_A = g$.

Podajmo nekaj primerov kategorij z njihovimi standardnimi oznakami:

- $\underline{\mathbf{Set}}$: Objekti so množice. Morfizmi so preslikave med njimi.
- $\underline{\mathbf{Mon}}$: Objekti so monoidi. Morfizmi so homomorfizmi med njimi.
- $\mathbf{Grp}$: Objekti so grupe. Morfizmi so homomorfizmi.
- $\underline{\mathbf{Vec}_O}$: Objekti so vektorski prostori nad obsegom O. Morfizmi so linearne transformacije.
- $\underline{\mathbf{Top}}$: Objekti so topološki prostori. Morfizmi so zvezne preslikave.

Pri vseh primerih je operacija kategorije kar kompozitum preslikav. Asociativnost povsod velja, enota pa je kar identiteta.

Sedaj poglejmo še primer kategorije, katere operacija ni kompozitum preslikav, morfizmi pa so vseeno preslikave. Taka je na primer kategorija $\underline{\mathbf{C}}$, katere edini objekt je $\mathbb{R}$. Morfizmi naj bodo raztegi, torej funkcije, ki pomnožijo realno število z nekim drugim realnim številom. Za morfizme $f, g, \ldots$ pa definiramo kompozitum kategorije $\bullet$ kot $g \bullet f = g \circ (\frac{1}{3} f)$, kjer je $\circ$ klasičen kompozitum funkcij. Zanimivo je, da enota v tem primeru ni klasična identiteta, ampak je enaka $id_{\mathbb{R}}(x) = 3x$. Asociativnost v kategoriji sledi iz asociativnosti množenja.

Primer 1.2. Razred množic z izbrano točko $\underline{\mathbf{Set}}_*$ je primer kategorije:

(1) Objekti v kategoriji $\underline{\mathbf{Set}}_*$ so $(A, a), (B, b), \ldots$ Pri tem so $A, B, \ldots \in \underline{\mathbf{Set}}$ in velja $a \in A$, $b \in B, \ldots$
(2) Za objekta $(A, a), (B, b)$ je množica $\mathrm{Hom}((A, a), (B, b))$, množica takih funkcij $f : A \longrightarrow B$, ki slikajo element a v b, torej $f(a) = b$.
(3) Operacijo kompozitum $\circ$ definiramo kot običajen kompozitum funkcij. Asociativnost tako sledi iz asociativnosti funkcij. Enota za $(A, a) \in \underline{\mathbf{Set}}_*$ pa je kar id_A, torej identiteta na množici A.

Primer 1.3. Delno urejena množica naravnih števil je primer kategorije:

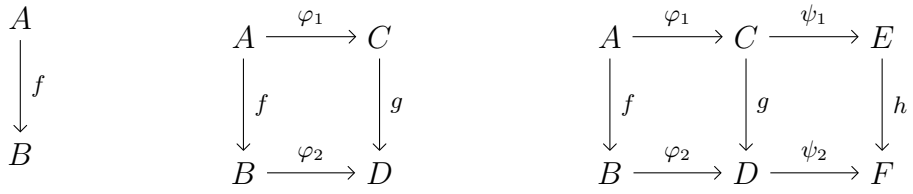
(1) Razred objektov je množica naravnih števil $\mathbb{N}_0$. Objekti so $0, 1, 2 \ldots \in \mathbb{N}_0$.
(2) Za objekta $n, m \in \mathbb{N}_0$ je množica $\mathrm{Hom}(n, m) = \{n \leq m\}$, če je $n \leq m$, sicer prazna. Element te množice je tako $n \leq m$ ali pa ga ni.
(3) Za poljubne tri objekte $n, m, k \in \mathbb{N}_0$ definiramo operacijo $\circ$, ki morfizmoma $n \leq m$ in $m \leq k$ priredi morfizem $n \leq k$, kar sledi iz tranzitivnosti relacije $\leq$. In res velja tudi:
   (a) Asociativnost: Saj za $a, b, c, d \in \mathbb{N}_0$, kjer je $a \leq b \leq c \leq d$, sledi $a \leq d$ ne glede na to, kako interpretiramo vrstni red.
   (b) Enota: Za vsak $a \in \mathbb{N}_0$ je enota kar $a \leq a$ in lastnosti enote veljata.

Opomba 1.4. Izkaže se, da ta struktura ni kategorija, če relacijo $\leq$ nadomestimo z $<$, zaplete se namreč pri enoti.

Opomba 1.5. Podobno kategorijo bi lahko skonstruirali za vse refleksivne in tranzitivne relacije.

1.1.1. *Kategorija puščic.* Oglejmo si kategorijo puščic $\underline{\mathbf{C}}^{\rightarrow}$ kategorije $\underline{\mathbf{C}}$. Objekti kategorije $\underline{\mathbf{C}}^{\rightarrow}$ so morfizmi kategorije $\underline{\mathbf{C}}$. Za vsaka objekta $f, g \in \underline{\mathbf{C}}^{\rightarrow}$ imamo množico morfizmov $\mathrm{Hom}(f, g)$, ki jo sestavljajo pari, za katere velja $(\varphi_1, \varphi_2) \in \mathrm{Hom}(f, g) \Leftrightarrow \varphi_2 \circ f = g \circ \varphi_1$ v kategoriji $\underline{\mathbf{C}}$. Definirajmo še operacijo kompozitum. Za morfizma (φ_1, φ_2), (ψ_1, ψ_2) je $(\psi_1, \psi_2) \circ (\varphi_1, \varphi_2) = (\psi_1 \circ \varphi_1, \psi_2 \circ \varphi_2)$. Asociativnost sledi iz asociativnosti v $\underline{\mathbf{C}}$. Enota za objekt $f : A \longrightarrow B$ pa je (id_A, id_B). Sledi še ponazoritev z diagrami.

Objekt f: Morfizem (φ_1, φ_2): Komponiranje morfizmov:



1.1.2. *Nasprotna kategorija.* Oglejmo si nasprotno kategorijo $\underline{\mathbf{C}}^{\mathrm{OP}}$ kategorije $\underline{\mathbf{C}}$. Objekti kategorije $\underline{\mathbf{C}}^{\mathrm{OP}}$ so kar objekti $\underline{\mathbf{C}}$. Za morfizme pa velja, da je $f = \widetilde{f} \in \mathrm{Hom}(A, B)$ v kategoriji $\underline{\mathbf{C}}^{\mathrm{OP}}$ natanko takrat, ko $f \in \mathrm{Hom}(B, A)$ v kategoriji $\underline{\mathbf{C}}$. Za razločevanje morfizme iz $\underline{\mathbf{C}}^{\mathrm{OP}}$ pišemo s "kačo". Operacijo $\circ_{\mathrm{OP}}$ definiramo kot $\widetilde{g} \circ_{\mathrm{OP}} \widetilde{f} = \widetilde{f \circ g}$. Taka definicija je dobra, saj iz $\widetilde{f} \in \mathrm{Hom}(A, B), \widetilde{g} \in \mathrm{Hom}(B, C)$ sledi $f \in \mathrm{Hom}(B, A), g \in \mathrm{Hom}(C, B)$. Torej $f \circ g \in \mathrm{Hom}(C, A)$. Preverimo asociativnost: $(\widetilde{h}\widetilde{g})\widetilde{f} = \widetilde{gh}\widetilde{f} = \widetilde{f(gh)} = \widetilde{(fg)h} = \widetilde{h}\widetilde{fg} = \widetilde{h}(\widetilde{g}\widetilde{f})$. Zapis operacije smo izpustili. Enota za A je id_A.

Definicija 1.6. Kategorija $\underline{\mathbf{C}}_1$ je *podkategorija* $\underline{\mathbf{C}}_2$, če so vsi objekti A kategorije $\underline{\mathbf{C}}_1$ tudi objekti kategorije $\underline{\mathbf{C}}_2$, ter če za vsak morfizem $f : A \longrightarrow B$ iz kategorije $\underline{\mathbf{C}}_1$ velja, da je tudi v kategoriji $\underline{\mathbf{C}}_2$. Dodatno mora tudi kompozitum v $\underline{\mathbf{C}}_1$ sovpadati s kompozitumom v $\underline{\mathbf{C}}_2$, prav tako tudi enote. Pišemo $\underline{\mathbf{C}}_1 \subseteq \underline{\mathbf{C}}_2$.

Kot primer si poglejmo nekaj podkategorij $\underline{\mathbf{Set}}$. Taka je na primer kategorija $\underline{\mathbf{Set}}_{\text{fin}}$, to je kategorija končnih množic in preslikav med njimi. Naslednji primeri so kategorije, kjer za objekte vzamemo vse objekte $\underline{\mathbf{Set}}$, le pri morfizmih se omejimo. Take so na primer kategorije, kjer imamo za morfizme samo injektivne, surjektivne ali bijektivne funkcije. Za dokaz, da so to res kategorije, je na primer treba preveriti, da je kompozitum injektivnih funkcij spet injektivna funkcija. Potem imamo tu še funkcije, kjer kot pogoj vzamemo, da ima praslika elementov določeno moč. Tako pri $f : A \longrightarrow B$, $b \in B$ dobimo različne kategorije za različne moči praslik $f^{-1}(b) \subseteq A$. Pri moči 1 je to ravno podkategorija $\underline{\mathbf{Set}}$ injektivnih funkcij. Za moč praslike 2 ne dobimo kategorije, ker ima praslika kompozicije dveh takih funkcij moč 4. Če rečemo, da mora biti praslika končna, spet dobimo kategorijo. Kompozitum funkcij s končno močjo praslike ima namreč spet končno moč praslike. Kategorija pa ni, če je moč praslike neskončna. Zatakne se namreč pri enoti.

Enostaven primer za kategorijo, vsebovano v $\underline{\mathbf{Grp}}$, je $\underline{\mathbf{Ab}}$, kategorija vseh abelovih grup. Morfizmi so seveda homomorfizmi. Kategorija $\underline{\mathbf{Grp}}$ pa je vsebovana v $\underline{\mathbf{Mon}}$. Tako imamo $\underline{\mathbf{Ab}} \subseteq \underline{\mathbf{Grp}} \subseteq \underline{\mathbf{Mon}}$.

1.2. **Funktor.** Glavni predmet proučevanja teorije kategorij so morfizmi. Ko želimo prenesti rezultate iz neke kategorije v drugo, to storimo z morfizmi med kategorijami. S funktorjem priredimo objektom in morfizmom prve kategorije objekte in morfizme druge kategorije. S tem da upoštevamo še usklajenost delovanja, si zagotovimo, da se lastnosti morfizmov prenesejo v drugo kategorijo.

Definicija 1.7. Naj bosta $\underline{\mathbf{C}}_1$ in $\underline{\mathbf{C}}_2$ kategoriji. Tedaj je $T : \underline{\mathbf{C}}_1 \longrightarrow \underline{\mathbf{C}}_2$ *funktor*, če velja:

(1) Funktor priredi vsakemu objektu $A \in \underline{\mathbf{C}}_1$ natančno določen objekt $T(A) \in \underline{\mathbf{C}}_2$.

(2) Funktor priredi vsakemu morfizmu $f : A \longrightarrow B$ natančno določen morfizem $T(f) : T(A) \longrightarrow T(B)$. Drugače povedano iz $f \in \mathrm{Hom}(A, B)$ sledi $T(f) \in \mathrm{Hom}(T(A), T(B))$.

(3) Delovanje funktorja je usklajeno z operacijo $\circ$ na $\underline{\mathbf{C}}_1$ in $\underline{\mathbf{C}}_2$:
- Za vse morfizme f, g velja $T(g \circ f) = T(g) \circ T(f)$.
- Za vsako enoto id_A velja $T(id_A) = id_{T(A)}$.

Opomba 1.8. Zaradi lepšega zapisa običajno pišemo delovanje funktorja na objektih kar TA namesto $T(A)$, podobno lahko storimo tudi za morfizme.

Primer 1.9. Oglejmo si primer funktorja za kategoriji:
$\underline{\mathbf{C}}_1 = (\mathbb{N}, |)$:

(1) Razred objektov je množica naravnih števil $\mathbb{N}$.
(2) Med objektoma a in b imamo en morfizem $a|b$, če a deli b, sicer nobenega.
(3) Pogojem za kategorijo je zadoščeno, saj je relacija $|$ tranzitivna in refleksivna.

$\underline{\mathbf{C}}_2 = (\mathcal{P}(\mathbb{N}), \subseteq)$:

(1) Razred objektov je potenčna množica $\mathcal{P}(\mathbb{N})$. Objekti so podmnožice naravnih števil $A, B, C, \ldots \subseteq \mathbb{N}$.

(2) Med A in B imamo morfizem $A \subseteq B$, če je A vsebovana v B, sicer nobenega.
(3) Pogojem za kategorijo je zadoščeno, saj je relacija $\subseteq$ tranzitivna in refleksivna.

Definirajmo sedaj naslednji funktor $F : \underline{\mathbf{C}}_1 \longrightarrow \underline{\mathbf{C}}_2$:

(1) Funktor priredi vsakemu objektu $a \in \underline{\mathbf{C}}_1$ množico vseh deliteljev števila a, ki jo označimo z $\mathrm{del}(a) \in \underline{\mathbf{C}}_2$.
(2) Funktor priredi vsakemu morfizmu $a|b$ morfizem $\mathrm{del}(a) \subseteq \mathrm{del}(b)$. Res velja, da iz $a|b$ sledi $\mathrm{del}(a) \subseteq \mathrm{del}(b)$.
(3) Delovanje funktorja je usklajeno z operacijo $\circ$ na $\underline{\mathbf{C}}_1$ in $\underline{\mathbf{C}}_2$, saj je med objekti kategorije $\underline{\mathbf{C}}_1$ edini možni morfizem $x|y$, ki se vedno preslika v $\mathrm{del}(x) \subseteq \mathrm{del}(y)$.

Primer 1.10. Kot primer funktorja si oglejmo še *pozabljivi funktor*. Za kategoriji $\underline{\mathbf{Grp}}$ in $\underline{\mathbf{Set}}$, definirajmo funktor $U : \underline{\mathbf{Grp}} \longrightarrow \underline{\mathbf{Set}}$, podan takole:

(1) Funktor priredi vsaki grupi $(G, *) \in \underline{\mathbf{Grp}}$ množico $G \in \underline{\mathbf{Set}}$. To je kar enaka množica kot grupa, samo da ni opremljena z operacijo.
(2) Funktor priredi vsakemu homomorfizmu grup $f \in \mathrm{Hom}((G, *), (H, *))$ funkcijo $f \in \mathrm{Hom}(G, H)$. Ta ima enak predpis, ni pa več homomorfizem.
(3) Delovanje funktorja je usklajeno s kompozitumom $\circ$ na $\underline{\mathbf{Grp}}$ in $\underline{\mathbf{Set}}$, saj je $U(g \circ f) = g \circ f = U(g) \circ U(f)$. Na tem mestu bi lahko s "kačo" razločevali homomorfizme od funkcij. Tega ne bomo storili, zavedajmo pa se, da so, čeprav enako označeni, to različni morfizmi. Res se tudi identiteta grupe slika v identiteto.

Ta funktor je odstranil strukturo množice. Smiselno ga je uporabljati na kategorijah, katerih objekti so množice z neko dodatno stukturo. Take so na primer množice z operacijo v kategorijah monoidov $\underline{\mathbf{Mon}}$, grup $\underline{\mathbf{Grp}}$ ali celo dvema operacijama v kategoriji kolobarjev $\underline{\mathbf{Rng}}$. Pozabljivi funktor, uporabljen na teh kategorijah, odstrani operacijo. Lahko rečemo, da jo pozabi, zato pozabljivi. Pozabljivi funktor rečemo tudi funktorju, ki pri množicah, ki so delno urejene, opremljene s topologijo ali metriko, pozabi na dodatno strukturo. Tako postanejo navadne množice, brez strukture. Za vajo premislite, kako izgleda pozabljivi funktor $U : \underline{\mathbf{Set}}_* \longrightarrow \underline{\mathbf{Set}}$.

1.2.1. *Yonedov funktor.* Naj bo W izbran objekt kategorije $\underline{\mathbf{C}}$. Definirajmo funktor $\mathsf{Y}_W : \underline{\mathbf{C}} \longrightarrow \underline{\mathbf{Set}}$. Funktor Y_W slika objekte $A \in \underline{\mathbf{C}}$ v objekte $\mathrm{Hom}(W, A) \in \underline{\mathbf{Set}}$. Vsak objekt A se tako preslika v množico morfizmov iz objekta W v objekt A. Funktor slika morfizme $f : A \longrightarrow B$ v funkcije $\mathsf{Y}_W f : \mathrm{Hom}(W, A) \longrightarrow \mathrm{Hom}(W, B)$. Predpis funkcije $\mathsf{Y}_W f$ na elementih $\varphi \in \mathrm{Hom}(W, A)$ je naslednji: $\mathsf{Y}_W f(\varphi) = f \circ_{\underline{\mathbf{C}}} \varphi$. Tako smo predpisali delovanje funktorja na objektih in morfizmih. Poglejmo še, da je res usklajen s kompozitumom $\circ_{\underline{\mathbf{C}}}$ in kompozitumom $\circ_{\underline{\mathbf{Set}}}$. Naj bo $f : A \longrightarrow B$, $g : B \longrightarrow C$, torej $g \circ f : A \longrightarrow C$. Potem je $\mathsf{Y}_W f : \mathrm{Hom}(W, A) \longrightarrow \mathrm{Hom}(W, B)$, $\mathsf{Y}_W g : \mathrm{Hom}(W, B) \longrightarrow \mathrm{Hom}(W, C)$ ter $\mathsf{Y}_W (g \circ f) : \mathrm{Hom}(W, A) \longrightarrow \mathrm{Hom}(W, C)$. Dokazujemo $\mathsf{Y}_W (g \circ f) = \mathsf{Y}_W g \circ \mathsf{Y}_W f$. Domena in kodomena se res ujemata, preveriti moramo še na elementih. Vzemimo poljuben $\varphi \in \mathrm{Hom}(W, A)$, torej $\varphi : W \longrightarrow A$ in preverimo: $\mathsf{Y}_W (g \circ f)(\varphi) = (g \circ f) \circ \varphi$ in $\mathsf{Y}_W (g) \circ \mathsf{Y}_W (f)(\varphi) = \mathsf{Y}_W (g)(f \circ \varphi) = g \circ (f \circ \varphi)$, zaradi pogoja asociativnosti v $\underline{\mathbf{C}}$ sta izraza res enaka. Za vsak A moramo pokazati še $\mathsf{Y}_W (id_A) = id_{\mathrm{Hom}(W,A)}$. Res $\mathsf{Y}_W (id_A)(\varphi) = id_A \circ \varphi = \varphi$. Pri tem nismo neposredno preverili, da je $\mathsf{Y}_W (id_A)$ enota v $\underline{\mathbf{Set}}$. Preverili smo, da je $\mathsf{Y}_W (id_A)$ enak $id_{\mathrm{Hom}(W,A)}$, tako da smo to preverili za poljuben element $\varphi \in \mathrm{Hom}(W, A)$. Za id pa vemo, da je enota v $\underline{\mathbf{Set}}$.

1.2.2. *Funktorji kot morfizmi.* Kot bi mogoče lahko uganili, obstaja tudi kategorija kategorij, ki jo označimo z $\underline{\mathbf{Cat}}$. To je kategorija, katere objekti so kategorije. Za njene morfizme vzamemo funktorje. Torej je za $\underline{\mathbf{C}}_1, \underline{\mathbf{C}}_2 \in \underline{\mathbf{Cat}}$ funktor $F : \underline{\mathbf{C}}_1 \longrightarrow \underline{\mathbf{C}}_2$ morfizem kategorije $\underline{\mathbf{Cat}}$. Operacijo kompozitum definiramo kot kompozitum na komponentah funktorja. Vsak funktor ima namreč dve komponenti, funkciji. Prva slika objekte, druga pa morfizme. Kompozitum tako definiramo kot kompozitum na vsaki komponenti posebej. Hitro lahko premislimo, da je operacija kompozitum asociativna, kar sledi iz asociativnosti funkcij. Za enoto pa vzamemo funktor Id. Bolj natančno za kategorijo $\underline{\mathbf{C}}$ je enota $\mathrm{Id}_{\underline{\mathbf{C}}} : \underline{\mathbf{C}} \longrightarrow \underline{\mathbf{C}}$. To je funktor, ki ima identiteto tako na komponenti objektov, kot komponenti morfizmov. Ko ga uporabimo, se torej tako objekti kot morfizmi ohranijo.

1.2.3. *Funktor potenčne množice $\mathcal{P}$.* Oglejmo si funktor $\mathcal{P} : \underline{\mathbf{Set}} \longrightarrow \underline{\mathbf{Set}}$. Ta funktor slika objekte $\mathcal{P} : A \longmapsto \mathcal{P}(A)$, vsaki množici A tako priredi njeno potenčno množico, to je množica vseh podmnožic množice A. Funktor priredi vsaki funkciji $f : A \longrightarrow B$ funkcijo $\mathcal{P}(f) : \mathcal{P}A \longrightarrow \mathcal{P}B$. Funkcija $\mathcal{P}(f)$ pa ima za U podmnožico A, torej $U \in \mathcal{P}(A)$, predpis $\mathcal{P}f(U) = f[U] = \{f(x) \mid x \in U\}$. Preverimo še usklajenost s kompozitumom. Najprej preverimo $\mathcal{P}(g \circ f) = \mathcal{P}g \circ \mathcal{P}f$:

$$\begin{aligned}\mathcal{P}(g \circ f)(U) &= \{g(f(x)) \mid x \in U\} = \{g(y) \mid y \in f[U]\} \\ &= \{z \mid z \in g[f[U]]\} = g[f[U]] = \mathcal{P}g(\mathcal{P}f(U))\end{aligned}$$

Za boljše razumevanje pokomentirajmo zadnji izračun. Za vsak a iz A vemo, da velja $(g \circ f)(a) = g(f(a))$; ker je U podmnožica A, to velja tudi za vse elemete U. Ker je slika množice slika vseh njenih elementov, je zato jasno, da zgornje velja. Na tem mestu bi opozorili samo na to, da je včasih dosti lažje izračunati $h(a)$, kjer je $h = (g \circ f)$, kot $g(f(a))$. Primer je recimo, ko je g inverz f in je torej h identiteta, tu je izračun $h(a)$ trivialen. Pri $g(f(a))$ pa moramo najprej izračunati $y = f(a)$, nato pa še $g(y)$, da dobimo rezultat. Tako nam lahko poznavanje funkcij in kompozituma močno olajša delo. Preverimo še pogoj za enoto:

$$\mathcal{P}(id_A)(U) = id_A[U] = \{id_A(x) \mid x \in U\} = \{x \mid x \in U\} = U$$

Zadnja ugotovitev je trivialna. Če na elementih množice uporabimo identiteto, se ne spremenijo in imamo še vedno isto množico. Tako smo preverili, da je $\mathcal{P}$ res funktor.

1.3. **Naravna transformacija.**

Definicija 1.11. Imejmo funktorja $F, G : \underline{\mathbf{C}}_1 \longrightarrow \underline{\mathbf{C}}_2$. *Naravna transformacija* $\alpha : F \longrightarrow G$ je družina morfizmov $\alpha_A : FA \longrightarrow GA$, ki jim rečemo komponente, za katere za vsak morfizem $f : A \longrightarrow B$ velja: $Gf \circ \alpha_A = \alpha_B \circ Ff$, kar lahko ponazorimo tudi s komutativnim diagramom:

$$\begin{array}{ccc} FA & \xrightarrow{\alpha_A} & GA \\ \downarrow{\scriptstyle Ff} & & \downarrow{\scriptstyle Gf} \\ FB & \xrightarrow{\alpha_B} & GB \end{array}$$

Primer 1.12. Oglejmo si naravno transformacijo med Yonedovima funktorjema. V kategoriji $\underline{\mathbf{C}}$ fiksirajmo objekta W in V, med katerima imamo določen morfizem $\psi : V \longrightarrow W$. Sedaj konstruiramo dva Yonedova funktorja Y_W in Y_V. Velja $\mathsf{Y}_W, \mathsf{Y}_V : \underline{\mathbf{C}} \longrightarrow \underline{\mathbf{Set}}$. Delovanje Yoneda funktorja smo že pogledali pri 1.2.1 na strani 7. Naravna transformacija $\alpha : \mathsf{Y}_W \longrightarrow \mathsf{Y}_V$ je družina funkcij $\alpha_A : \mathrm{Hom}(W, A) \longrightarrow \mathrm{Hom}(V, A)$. V tem primeru imajo te funkcije za vsak objekt A kar enak predpis in sicer $\alpha_A(\varphi) = \varphi \circ \psi$. Vzemimo poljuben morfizem $f : A \longrightarrow B$ in preverimo veljavnost $\mathsf{Y}_V f \circ \alpha_A = \alpha_B \circ \mathsf{Y}_W f$. Vzemimo poljuben element $\varphi \in \mathrm{Hom}(W, A)$ in računajmo:

$$\mathsf{Y}_V f \circ \alpha_A(\varphi) = \mathsf{Y}_V f(\varphi \circ \psi) = f \circ (\varphi \circ \psi)$$
$$\alpha_B \circ \mathsf{Y}_W f(\varphi) = \alpha_B(f \circ \varphi) = (f \circ \varphi) \circ \psi$$

Zaradi asociativnosti v $\underline{\mathbf{C}}$ enakost res drži in pogoju naravnosti je zadoščeno. Za boljšo predstavo morfizem f in komponeti naravne transformacije α ponazorimo še z diagramom:

V kategoriji $\underline{\mathbf{C}}$:

$$\begin{array}{c} A \\ \downarrow{\scriptstyle f} \\ B \end{array}$$

V kategoriji $\underline{\mathbf{Set}}$:

$$\begin{array}{ccc} \mathrm{Hom}(W, A) & \xrightarrow{\alpha_A} & \mathrm{Hom}(V, A) \\ \downarrow{\scriptstyle \mathsf{Y}_W f} & & \downarrow{\scriptstyle \mathsf{Y}_V f} \\ \mathrm{Hom}(W, B) & \xrightarrow{\alpha_B} & \mathrm{Hom}(V, B) \end{array}$$

1.3.1. *Naravna transformacija kot morfizem.* Skonstruiramo lahko kategorijo, katere objekti so funktorji med kategorijama $\underline{\mathbf{C}}_1$ in $\underline{\mathbf{C}}_2$, torej so oblike $F : \underline{\mathbf{C}}_1 \longrightarrow \underline{\mathbf{C}}_2$, morfizmi pa so naravne transformacije med njimi. Za operacijo kompozitum vzamemo kar kompozitum iz kategorije $\underline{\mathbf{C}}_2$, ki deluje na komponentah naravne transformacije. Tako za naravni transformaciji $\alpha : F \longrightarrow G$ in $\beta : G \longrightarrow H$ definiramo kompozitum kot $\beta \circ \alpha$, kjer je komponenta te transformacije enaka $(\beta \circ \alpha)_A = \beta_A \circ \alpha_A$. Asociativnost sledi iz asociativnosti kategorij, nad katerimi delujejo funktorji. Enota pa je naravna transformacjia, ki ima na vseh komponentah identiteto in sploh ne spremeni funktorja.

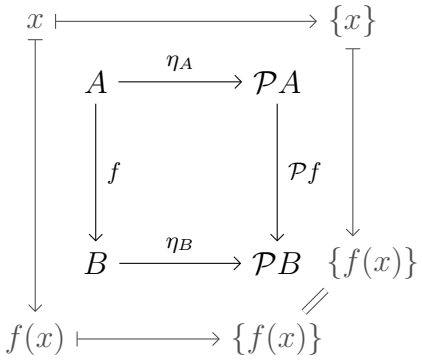
To je sicer le podkategorija kategorije, v kateri bi za objekte vzeli vse funktorje. Tam med funktorji različnih kategorij pač ne bi bilo naravnih transformacij, torej morfizmov v tem primeru.

1.3.2. *Naravne transformacije funktorjev* $\mathrm{Id}, \mathcal{P}, \mathcal{P} \circ \mathcal{P}$. Dokazali smo že, da je $\mathcal{P}$ funktor. Iz tega sledi, da je tudi $\mathcal{P} \circ \mathcal{P}$ funktor. Poiščimo nekaj naravnih transformacij med funktorji $\mathrm{Id}, \mathcal{P}, \mathcal{P} \circ \mathcal{P}$. Najprej si oglejmo naravne transformacije oblike $\alpha : \mathrm{Id} \longrightarrow \mathcal{P}$. Taka je recimo $\eta : \mathrm{Id} \longrightarrow \mathcal{P}$, definirana na komponentah kot:

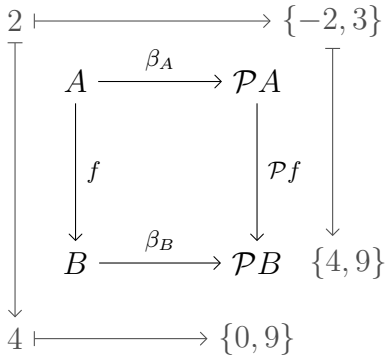
$$\eta_A : A \longrightarrow \mathcal{P}A$$
$$\eta_A : x \longmapsto \{x\}$$

Seveda je treba preveriti, da je η res naravna transformacija. To bomo naredili s pomočjo diagrama. Preveriti moramo, da diagram, ki ustreza pogoju $\mathcal{P}f \circ \eta_A =$

$\eta_B \circ f$, komutira. Izberimo poljuben $x \in A$ in dokažimo, da po obeh poteh dobimo isto:



Na koncu smo na obeh straneh dobili $\{f(x)\}$, torej diagram komutira, zato je η res naravna transformacija. Poskusimo poiskati še kakšno drugo naravno transformacijo istih funktorjev. Ena je recimo α, definirana na komponentah kot $\alpha_A : A \longrightarrow \mathcal{P}A$, s predpisom $\alpha_A : x \longmapsto \{\}$ na elementih A. Tudi to je naravna transformacija. Ko preverjamo komutativnost ustreznega diagrama, dobimo po obeh poteh na koncu prazno množico. Sedaj si oglejmo še primer, ki ni naravna transformacija, vsaj v splošnem ne. Vzemimo $\beta : \mathrm{Id} \longrightarrow \mathcal{P}$, definiran na komponentah kot: $\beta_A : A \longrightarrow \mathcal{P}A$, s predpisom $\beta_A : x \longmapsto A \setminus \{x\}$. To je naravna transformacija v podkategoriji **Set**, kjer za morfizme vzamemo samo bijektivne funkcije, v kategoriji **Set** pa ni, saj najdemo protiprimer. Vzemimo $A = \{-2, 2, 3\}$, $B = \{0, 4, 9\}$ ter $f(x) = x^2$. Narišimo diagram in preverimo komutativnost kar za ta konkreten primer:

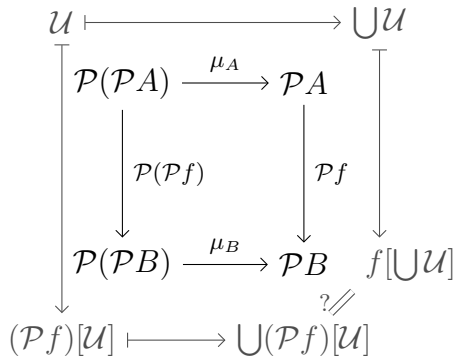


Ugotovimo, da diagram ne komutira, saj $\{0, 9\} \neq \{4, 9\}$. Torej β ni naravna transformacija. Oglejmo si še naravne transformacije oblike $\alpha : \mathcal{P} \circ \mathcal{P} \longrightarrow \mathcal{P}$. Vzemimo na primer $\mu : \mathcal{P} \circ \mathcal{P} \longrightarrow \mathcal{P}$, definirano na komponentah kot:

$$\mu_A : \mathcal{P}(\mathcal{P}A) \longrightarrow \mathcal{P}A$$
$$\mu_A : \mathcal{U} \longmapsto \bigcup \mathcal{U}$$

Preveriti je treba, da je μ naravna transformacija. To bomo naredili s pomočjo diagrama. Preveriti moramo, da diagram, ki ustreza pogoju $\mathcal{P}f \circ \mu_A = \mu_B \circ \mathcal{P}\mathcal{P}f$,

komutira. Izberimo neki $\mathcal{U} \in \mathcal{P}(\mathcal{P}A)$ in dokažimo, da po obeh poteh dobimo isto:

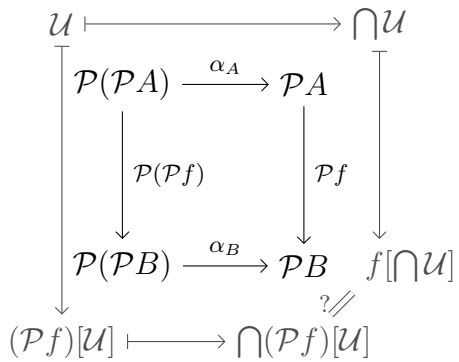


Po levi poti smo dobili $\bigcup \mathcal{P}f[\mathcal{U}]$, po zgornji pa $f[\bigcup\mathcal{U}]$. Za komutativnost diagrama želimo dokazati, da je to dvoje enako. Za dokaz enakosti množic, moramo preveriti, da vsebujeta iste elemente:

$$\begin{aligned} b \in f\left[\bigcup\mathcal{U}\right] &\Leftrightarrow \exists a \in \bigcup\mathcal{U}.f(a) = b \\ &\Leftrightarrow \exists U \in \mathcal{U}.\exists a \in U.f(a) = b \end{aligned}$$

$$\begin{aligned} b \in \bigcup(\mathcal{P}f)[\mathcal{U}] &\Leftrightarrow \exists V \in (\mathcal{P}f)[\mathcal{U}].b \in V \\ &\Leftrightarrow \exists V \in \{\mathcal{P}f(U) \mid U \in \mathcal{U}\}.b \in V \\ &\Leftrightarrow \exists V \in \{f[U] \mid U \in \mathcal{U}\}.b \in V \\ &\Leftrightarrow \exists U \in \mathcal{U}.b \in f[U] \\ &\Leftrightarrow \exists U \in \mathcal{U}.\exists a \in U.f(a) = b \end{aligned}$$

Množici vsebujeta iste elemente, s tem smo dokazali, da je μ res naravna transformacija. Poskusimo poiskati še kakšno naravno transformacijo $\alpha : \mathcal{P} \circ \mathcal{P} \longrightarrow \mathcal{P}$. Definirajmo α kot $\alpha_A(\mathcal{U}) = \bigcap\mathcal{U}$. Hitro ugotovimo, da pogoj naravnosti ne velja za funkcije, ki niso injektivne. Kaj pa za injektivne funkcije? Poglejmo, če ustrezen diagram komutira:



Zanima nas, ali je $f[\bigcap\mathcal{U}] = \bigcap(\mathcal{P}f)[\mathcal{U}]$. Preverimo, če množici vsebujeta iste elemente.

$$\begin{aligned} b \in f\left[\bigcap\mathcal{U}\right] &\Leftrightarrow \exists a \in \bigcap\mathcal{U}.f(a) = b \\ &\Leftrightarrow \exists a.\forall U \in \mathcal{U}.a \in U \wedge f(a) = b \end{aligned}$$

$$b \in \bigcap(\mathcal{P}f)[\mathcal{U}] \Leftrightarrow \forall V \in (\mathcal{P}f)[\mathcal{U}].b \in V$$
$$\Leftrightarrow \forall U \in \mathcal{U}.b \in f[U]$$
$$\Leftrightarrow \forall U \in \mathcal{U}.\exists a \in U.f(a) = b$$

To na prvi pogled ni enako, in res ni, če f ni injektivna. Za injektiven f pa velja $f(x_1) = f(x_2) \Rightarrow x_1 = x_2$. Ker je f injektivna na A, obstaja samo en a, da je $f(a) = b$. Pri pogoju $\forall U \in \mathcal{U}.\exists a \in U.f(a) = b$ tako ugotovimo, da obstaja vedno isti a, da je $f(a) = b$, zato lahko pišemo kar $\exists a.\forall U \in \mathcal{U}.a \in U \wedge f(a) = b$. Ravno to pa smo tudi želeli pokazati. Tako smo ugotovili, da je za podkategorijo $\underline{\mathbf{Set}}$ injektivnih funkcij presek naravna transformacija.

Obrazložimo še, kako pridemo na idejo, da bi presek za podkategorijo injektivnih funkcij lahko bila naravna transformacija. Za bijektivne funkcije je dokaj jasno. Lahko si predstavljamo, da bijekcija zlepi vse elemente, ki si bijektivno ustrezajo. Vse, kar se dogaja z enim elementom, se dogaja tudi z njegovim parom. Ko premislimo, da surjektivnost za presek v ničemer ne vpliva na končno sliko, ugotovimo, da je pomembna le injektivnost. Dobro je premisliti, zakaj smo pri komplementu kot naravni transformaciji potrebovali tudi surjektivnost funkcij.

2. Definicija monade

Definicija 2.1. *Monada* na kategoriji $\underline{\mathbf{C}}$ je trojica (T, η, μ), podana z:

(1) funktorjem $T : \underline{\mathbf{C}} \longrightarrow \underline{\mathbf{C}}$
(2) naravno transformacijo $\eta : \mathrm{Id} \longrightarrow T$
(3) naravno transformacijo $\mu : T \circ T \longrightarrow T$

Pri tem za vsak objekt A velja:

(a) $\mu_A \circ T\eta_A = id_{TA} = \mu_A \circ \eta_{TA}$
(b) $\mu_A \circ T\mu_A = \mu_A \circ \mu_{TA}$

Sedaj razpišimo pogoje. Zapišimo komponenti naravnih transformacij, uporabljenih na objektu A:

- $\eta_A : A \longrightarrow TA$
- $\mu_A : T(TA) \longrightarrow TA$

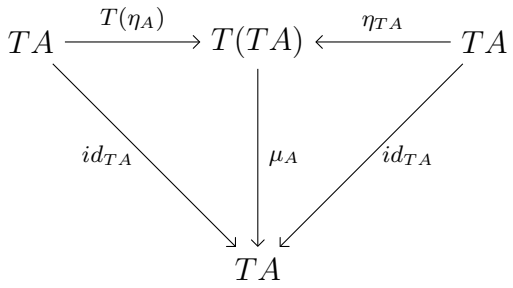
ter pripadajoči enačbi, ki veljata za naravni transformaciji za vsak morfizem $f : A \longrightarrow B$:

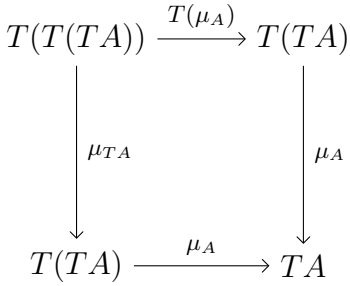
- $Tf \circ \eta_A = \eta_B \circ f$
- $Tf \circ \mu_A = \mu_B \circ T(Tf)$

kar lahko ponazorimo tudi s komutativnima diagramoma:

$$\begin{array}{ccc} A & \xrightarrow{\eta_A} & TA \\ \downarrow{\scriptstyle f} & & \downarrow{\scriptstyle Tf} \\ B & \xrightarrow{\eta_B} & TB \end{array} \qquad \begin{array}{ccc} T(TA) & \xrightarrow{\mu_A} & TA \\ \downarrow{\scriptstyle T(Tf)} & & \downarrow{\scriptstyle Tf} \\ T(TB) & \xrightarrow{\mu_B} & TB \end{array}$$

Ponazorimo sedaj še pogoja (a) in (b):





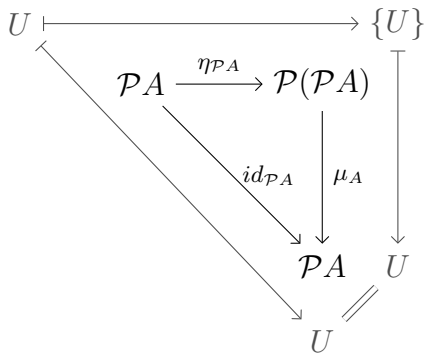
Primer 2.2. Za primer uporabimo funktor $\mathcal{P}$ ter naravni transformaciji, ki smo ju definirali v prejšnjem razdelku. Monada je trojica $(\mathcal{P}, \eta, \mu)$, podana z:

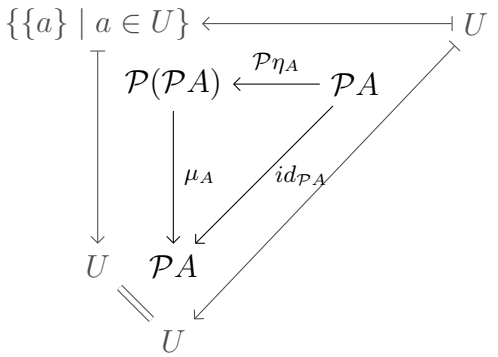
(1) funktorjem $\mathcal{P} : \underline{\mathbf{Set}} \longrightarrow \underline{\mathbf{Set}}$
(2) naravno transformacijo $\eta : \mathrm{Id} \longrightarrow \mathcal{P}$, ki slika $\eta_A(x) = \{x\}$
(3) naravno transformacijo $\mu : \mathcal{P} \circ \mathcal{P} \longrightarrow \mathcal{P}$, ki slika $\mu_A(\mathcal{U}) = \bigcup \mathcal{U}$

To, da je $\mathcal{P}$ funktor in η, μ naravni transformaciji, smo že preverili. Preveriti moramo še dodatna pogoja, torej veljavnost:

(a) $\mu_A \circ \mathcal{P}\eta_A = id_{\mathcal{P}A} = \mu_A \circ \eta_{\mathcal{P}A}$
(b) $\mu_A \circ \mathcal{P}\mu_A = \mu_A \circ \mu_{\mathcal{P}A}$

Za preverjanje pogojev si bomo pomagali z diagrami. Najprej preverimo komutativnost diagrama, ki ustreza pogoju $\mu_A \circ \mathcal{P}\eta_A = id_{\mathcal{P}A} = \mu_A \circ \eta_{\mathcal{P}A}$. Na levemu diagramu smo tako preverili pogoj $\mu_A \circ \eta_{\mathcal{P}A} = id_{\mathcal{P}A}$, na desnem pa pogoj $\mu_A \circ \mathcal{P}\eta_A = id_{\mathcal{P}A}$:



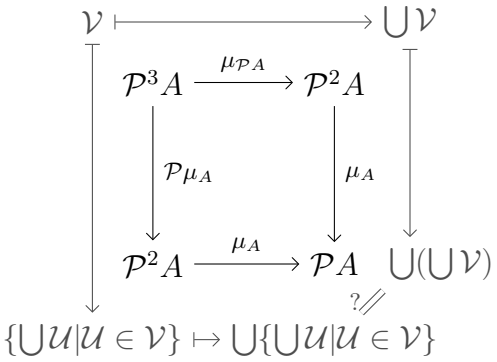


$\mu_A\{U\} = \bigcup\{U\} = U$

$\mu_A\{\{a\}|a \in U\} = \bigcup\{\{a\}|a \in U\} = U$

Ker za vsako množico U na koncu po obeh poteh dobimo isti rezultat U, diagram res komutira. S tem smo preverili, da je pogoju $\mu_A \circ \mathcal{P}\eta_A = id_{\mathcal{P}A} = \mu_A \circ \eta_{\mathcal{P}A}$ zadoščeno.

Preveriti moramo še pogoj $\mu_A \circ \mathcal{P}\mu_A = \mu_A \circ \mu_{\mathcal{P}A}$. Spet si pomagamo z diagramom:



Tokrat na koncu dobimo dve množici, $U_1 = \bigcup\{\bigcup\mathcal{U}|\mathcal{U} \in \mathcal{V}\}$ ter $U_2 = \bigcup(\bigcup\mathcal{V})$. Diagram komutira, če sta ti dve množici enaki. Na prvi pogled izgledata različni, izkaže pa se, da sta enaki. Poglejmo, katere elemente vsebujeta; če obe vsebujeta iste elemente, sta namreč enaki:

$$\begin{aligned} a \in \bigcup\left(\bigcup\mathcal{V}\right) &\Leftrightarrow \exists U \in \bigcup\mathcal{V}.a \in U \\ &\Leftrightarrow \exists\mathcal{U} \in \mathcal{V}.\exists U \in \mathcal{U}.a \in U \end{aligned}$$

$$\begin{aligned} a \in \bigcup\left\{\bigcup\mathcal{U} \mid \mathcal{U} \in \mathcal{V}\right\} &\Leftrightarrow \exists W \in \left\{\bigcup\mathcal{U} \mid \mathcal{U} \in \mathcal{V}\right\}.a \in W \\ &\Leftrightarrow \exists\mathcal{U} \in \mathcal{V}.a \in \bigcup\mathcal{U} \\ &\Leftrightarrow \exists\mathcal{U} \in \mathcal{V}.\exists U \in \mathcal{U}.a \in U \end{aligned}$$

Po izračunu ugotovimo, da je $a \in U_1$ natanko takrat, ko je $a \in U_2$. Torej sta množici res enaki. Tako smo preverili komutativnost diagrama in s tem pokazali, da je zadoščeno pogoju $\mu_A \circ \mathcal{P}\mu_A = \mu_A \circ \mu_{\mathcal{P}A}$. S tem je zadoščeno obema dodatnima pogojema. Res je tudi $\mathcal{P}$ funktor ter η in μ naravni transformaciji, zato je $(\mathcal{P}, \eta, \mu)$ monada.

Trditev 2.3. *Za monado velja $T\eta_A \circ \eta_A = \eta_{TA} \circ \eta_A$.*

Dokaz. Samo vstavimo η_A namesto f v definicijo naravne transformacije η za komponento objekta A. □

Lahko bi v definicijo naravne transformacije η vstavili tudi μ_A ali pa v definicijo μ vstavili η_A oziroma μ_A. Pogosto je to smiselno početi pri konkretnih naravnih transformacijah in preverjanju, če smo res našli monado.

Na tem mestu tudi odgovorimo na vprašanje ali je lahko med dvema funktorjema več naravnih transformacij. Kot smo videli v prejšnjem poglavju, obstaja med funktorjema Id in $\mathcal{P}$ več naravnih transformacij $\alpha : \mathrm{Id} \longrightarrow \mathcal{P}$. Zanima nas, ali so dobre za monado. Izkaže se, da naravna transformacija $\alpha : \mathrm{Id} \longrightarrow \mathcal{P}$, s predpisom $\alpha_A : a \longrightarrow \{\}$, ni dobra za monado, saj ne more zadostiti pogoju $\mu_A \circ \mathcal{P}\eta_A = id_{\mathcal{P}A} = \mu_A \circ \eta_{\mathcal{P}A}$. Obstajajo pa primeri, ko je med funktorji več naravnih transformacij, ki zadoščajo pogoju za monado. Kot bomo videli pri konstrukcijah nad vektorskimi prostori, jih je lahko celo neskončno.

3.1. **Predstavitev izjem.** Oglejmo si, kako lahko v kategoriji množic predstavimo izjeme. Izjemo bomo označili z $*$. Najprej moramo definirati disjunktno unijo množic. Definirajmo torej simbol $+$ kot disjunktno unijo množic $X + Y = \{0\} \times X \cup \{1\} \times Y = \{(0, x) \mid x \in X\} \cup \{(1, y) \mid y \in Y\}$.

Definirajmo funktor $T : \underline{\mathbf{Set}} \longrightarrow \underline{\mathbf{Set}}$, ki slika množice (objekte) A v $TA = \{*\} + A$ ter funkcije (morfizme) f v Tf, definirane kot $Tf : (1, a) \longmapsto (1, f(a))$ in $Tf : (0, *) \longmapsto (0, *)$. Pri tem je $a \in A$, kar bo veljalo za celoten zgled.

Preveriti moramo, da je T res funktor, torej usklajenost delovanja z operacijo kompozitum. Poračunajmo, da res velja $T(g \circ f) = Tg \circ Tf$ in $T(id_A) = id_{TA}$ za vse ustrezne funkcije. Naj bo $f : A \longrightarrow B$, $g : B \longrightarrow C$ ter $a \in A$, preverimo prvi pogoj:

$$T(g \circ f)(1, a) = (1, g(f(a))) = Tg(1, f(a)) = Tg(Tf(1, a))$$
$$T(g \circ f)(0, *) = (0, *) = Tg(0, *) = Tg(Tf(0, *))$$

Pokazali smo, da prvi pogoj res drži. Preverimo še pogoj za enoto:

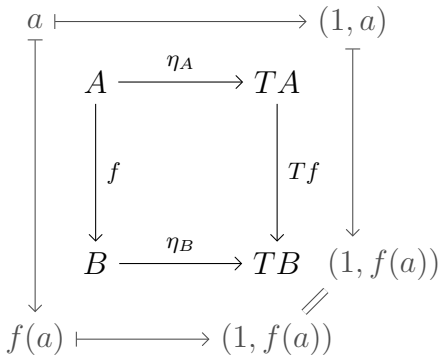
$$T(id_A)(1, a) = (1, id_A(a)) = (1, a)$$
$$T(id_A)(0, *) = (0, *)$$

Zadnji izračun sledi iz definicije funktorja, ker je za vsako funkcijo f predpis funkcije Tf enak $Tf(0, *) = (0, *)$. S tem smo preverili, da je T res funktor. Definirajmo še naravni transformaciji η in μ. Naj bo $\eta : \mathrm{Id} \longrightarrow T$ družina funkcij $\eta_A : A \longrightarrow TA$, s predpisom $\eta_A : a \longmapsto (1, a)$ za vsako množico A. Naravna transformacija $\mu : T \circ T \longrightarrow T$ pa naj bo družina funkcij $\mu_A : TTA \longrightarrow TA$, s predpisom $\mu_A : (1, (1, a)) \longmapsto (1, a)$, $\mu_A : (1, (0, *)) \longmapsto (0, *)$, in $\mu_A : (0, *) \longmapsto (0, *)$.

Preverimo, da sta to res naravni transformaciji. Vzemimo poljuben $f : A \longrightarrow B$ in pokažimo, da je η naravna transformacija. Veljati mora torej $\eta_B \circ f = Tf \circ \eta_A$:

$$n_B \circ f(a) = n_B(f(a)) = (1, f(a))$$
$$Tf \circ \eta_A(a) = Tf(1, a) = (1, f(a))$$

To smo pokazali tako, da smo za vsak $a \in A$ preverili veljavnost izraza $(\eta_B \circ f)(a) = (Tf \circ \eta_A)(a)$. Sedaj pa si poglejmo, kako bi to lahko videli s pomočjo diagrama:



Ko preverimo, da za vsak $a \in A$, tako po zgornji $a \mapsto (1, a) \mapsto (1, f(a))$ kot po levi $a \mapsto f(a) \mapsto (1, f(a))$ poti dobimo isti rezultat $(1, f(a))$, rečemo, da diagram komutira. To v našem primeru pomeni, da je enačbam zadoščeno in je η res naravna transformacija.

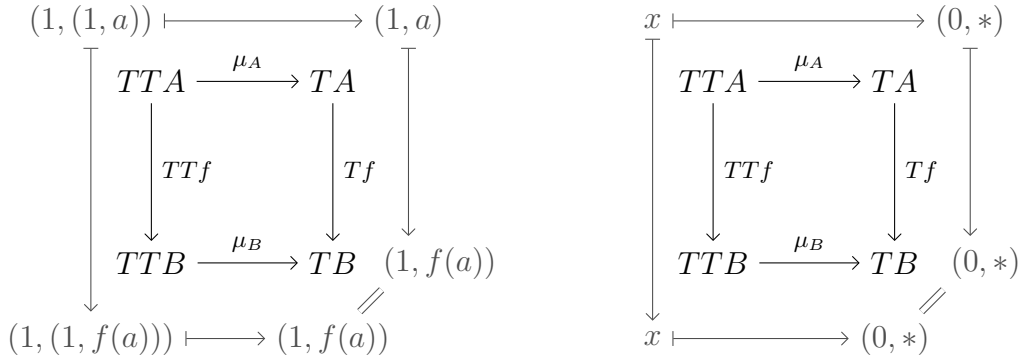
Sedaj preverimo, da je tudi μ naravna transformacija, najprej z računom, nato pa še z diagramom. Za vsak $x \in TTA$ preverimo veljavnost izraza $\mu_B \circ TTf(x) = Tf \circ \mu_A(x)$:

$$\mu_B \circ TTf(1,(1,a)) = \mu_B(1,Tf(1,a)) = \mu_B(1,(1,f(a))) = (1,f(a))$$
$$Tf \circ \mu_A(1,(1,a)) = Tf(1,a) = (1,f(a))$$

$$\mu_B \circ TTf(1,(0,*)) = \mu_B(1,Tf(0,*)) = \mu_B(1,(0,*)) = (0,*)$$
$$Tf \circ \mu_A(1,(0,*)) = Tf(0,*) = (0,*)$$

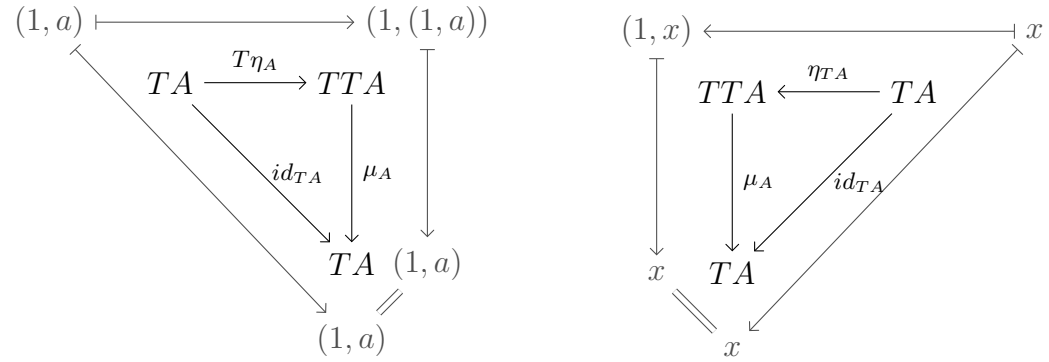
$$\mu_B \circ TTf(0,*) = \mu_B(0,*) = (0,*)$$
$$Tf \circ \mu_A(0,*) = Tf(0,*) = (0,*)$$

Podobno preverimo še komutativnost ustreznega diagrama, pri tem na desnem diagramu x predstavlja eno izmed obeh izjem:



Do sedaj imamo funktor T in naravni transformaciji η ter μ. Če želimo, da je (T, η, μ) monada, pa mora biti zadoščeno še dvema pogojema. Najprej preverimo pogoj $\mu_A \circ \eta_{TA} = id_{TA} = \mu_A \circ T\eta_A$. Prepričajmo se, da sta tako leva, kot desna stran izraza enaka identiteti:

$$\mu_A(T\eta_A(1,a)) = \mu_A(1,\eta_A(a)) = (1,a) \qquad \mu_A(\eta_{TA}(1,a)) = \mu_A(1,(1,a)) = (1,a)$$
$$\mu_A(T\eta_A(0,*)) = \mu_A(0,*) = (0,*) \qquad \mu_A(\eta_{TA}(0,*)) = \mu_A(1,(0,*)) = (0,*)$$



Tokrat smo vse preverili samo z računom. Pri levem diagramu namreč nismo naredili primera $(0,*)$. Na desnem diagramu pa sta upoštevani obe možnosti, samo za x enkrat vzamemo $(1,a)$ drugič pa $(0,*)$.

Sedaj je potrebno preveriti še pogoj $\mu_A \circ \mu_{TA} = \mu_A \circ T\mu_A$. Tu bomo naredili le izračun za $(1,(1,(1,a)))$:

$$\mu_A(\mu_{TA}(1,(1,(1,a)))) = \mu_A(1,(1,a)) = (1,a)$$
$$\mu_A(T\mu_A(1,(1,(1,a)))) = \mu_A(1,\mu_A(1,(1,a))) = \mu_A(1,(1,a)) = (1,a)$$

Za možnosti $(0,*)$, $(1,(0,*))$, $(1,(1,(0,*)))$ pa samo navedimo, da se na koncu vse slikajo v $(0,*)$, kar je sicer seveda potrebno preveriti. Tu je še diagram, za katerega moramo preveriti, da komutira, vendar tega na tem mestu ne bomo storili:

$$\begin{array}{ccc} T^3A & \xrightarrow{\mu_{TA}} & T^2A \\ \downarrow{\scriptstyle T\mu_A} & & \downarrow{\scriptstyle \mu_A} \\ T^2A & \xrightarrow{\mu_A} & TA \end{array}$$

Opomba 3.1. Povsem ekvivalentno je, ali preverjamo pogoje z računanjem enačb ali z dokazovanjem komutativnosti diagramov, ki jih te enačbe porodijo. Sicer je predvsem na začetku dobro narediti nekaj primerov z dokazovanjem komutativnosti diagramov, ker dobimo boljšo predstavo o tem, kaj te enačbe predstavljajo. Diagrami ponujajo več intuicije, žal pa jih ni možno vedno narisati, oziroma lahko niso pregledni. To je razlog, zakaj smo na tem primeru vse preverili na oba načina. Sicer si vedno lahko izberemo način, ki nam bolj ustreza.

Opomba 3.2. Na podoben način, kot smo tu predstavili izjeme, je v Haskellu definirana monada Maybe.

3.2. **Konstrukcije nad vektorskimi prostori.** Najprej si oglejmo monado nad kategorijo $\underline{\mathbf{Set}}$, kasneje bomo podobno monado definirali še nad kategorijo vektorskih prostorov $\underline{\mathbf{Vec}}$. Definirajmo funktor $T : \underline{\mathbf{Set}} \longrightarrow \underline{\mathbf{Set}}$, ki slika objekte $T : V \longmapsto V \times V$ in funkcije $T : f \longmapsto (f,f)$.

Sedaj poiščimo kakšno monado, katere sestavni del je ta funktor. Kot dokaj enostaven primer lahko najdemo monado (T,η,μ), kjer sta η in μ podana na naslednji način: $\eta_V(x) = (x,x)$ in $\mu_V((x,y),(z,w)) = (x,w)$. Preverimo, da sta to res naravni transformaciji in da skupaj s funktorjem T zadoščata definiciji monade. Naj bo $f : V_1 \longrightarrow V_2$ poljubna funkcija:

(1) η_V je naravna transformacija:

$$\eta_{V_2} \circ f(x) = \eta_{V_2}(f(x)) = (f(x),f(x))$$
$$Tf \circ \eta_{V_1}(x) = Tf(x,x) = (f(x),f(x))$$

(2) μ_V je naravna transformacija:

$$\mu_{V_2} \circ TTf((x,y),(z,w)) = \mu_{V_2}(Tf(x,y),Tf(z,w))$$
$$= \mu_{V_2}(f(x),f(y)),(f(z),f(w)) = (f(x),f(w))$$
$$Tf \circ \mu_{V_1}((x,y),(z,w)) = Tf(x,w) = (f(x),f(w))$$

(3) Zadoščeno je $\mu_V \circ \eta_{TV} = id_{TV} = \mu_V \circ T\eta_V$:

$$\mu_V \circ \eta_{TV}(x,y) = \mu_V((x,y),(x,y)) = (x,y)$$
$$\mu_V \circ T\eta_V(x,y) = \mu_V(\eta(x),\eta(y)) = \mu_V((x,x),(y,y)) = (x,y)$$

(4) Zadoščeno je $\mu_V \circ \mu_{TV} = \mu_V \circ T\mu_V$:

$$\mu_V \circ \mu_{TV}(((x_1, x_2), (x_3, x_4)), ((x_5, x_6), (x_7, x_8))) = \mu_V((x_1, x_2), (x_7, x_8)) = (x_1, x_8)$$

$$\mu_V \circ T\mu_V(((x_1, x_2), (x_3, x_4)), ((x_5, x_6), (x_7, x_8))) =$$
$$\mu_V(\mu_V((x_1, x_2), (x_3, x_4)), \mu_V((x_5, x_6), (x_7, x_8))) = \mu_V((x_1, x_4), (x_5, x_8)) = (x_1, x_8)$$

Torej je zadoščeno vsem pogojem in res smo našli monado. Če smo pozorni, ugotovimo, da lahko podobno monado skonstruiramo tudi nad kategorijo vseh vektorskih prostorov **Vec**. Za funktor $T : \underline{\mathbf{Vec}} \longrightarrow \underline{\mathbf{Vec}}$, ki slika objekte $T : V \longmapsto V \times V$ in linearne transformacije $T : f \longmapsto (f, f)$, sta naravni transformaciji η in μ definirani kot prej res linearni transformaciji in zadoščata pogojem za monado.

Sedaj pa bi radi poiskali še več monad, ki ustrezajo danemu funktorju. Iščemo torej monade (T, η, μ), kjer sta $\eta : \mathrm{Id} \longrightarrow T$ in $\mu : T \circ T \longrightarrow T$ naravni transformaciji, ki ustrezata dodatnim pogojem. Njune komponente so linearne transformacije:

$$\eta_V : V \longrightarrow V \times V$$
$$\eta_V(x) = (g_1(x), g_2(x))$$

$$\mu_V : (V \times V) \times (V \times V) \longrightarrow (V \times V)$$
$$\mu_V((x, y), (z, w)) = (h_1(x, y, z, w), h_2(x, y, z, w))$$

Izkaže se, da je iskanje splošnih linearnih transformacij η_V, μ_V, ki bi ustrezale pogojem za monado, dokaj zahtevno. Eden od problemov je recimo to, da bi morali za vektorske prostore različnih dimenzij različno definirati komponente naravne transformacije. Mi se bomo zaradi tega omejili le na iskanje linearnih transformacij, podanih kot:

$$\eta_V(x) = (\alpha x, \beta x)$$
$$\mu_V((x, y), (z, w)) = \widetilde{\mu_V}(x, y, z, w) = (a_1 x + a_2 y + a_3 z + a_4 w, b_1 x + b_2 y + b_3 z + b_4 w)$$

Pri tem so $\alpha, \beta, a_1, \ldots, b_4$ skalarji, neodvisni od vektorskega prostora V. Ponazorimo komponeti naravnih transformacij še v matričnem zapisu:

$$\eta_V = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

$$\widetilde{\mu_V} = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \end{bmatrix}$$

Na ta način definirani η in μ res zadoščata pogojem naravne transformacije. Pri tem je pomembno, da so morfizmi le linearne transformacije, sicer to ne drži. Pogoj za naravno transormacijo μ, uporabljen na $((x, y), (z, w))$, nam med drugim da enačbo: $f(a_1 x + a_2 y + a_3 z + a_4 w) = a_1 f(x) + a_2 f(y) + a_3 f(z) + a_4 f(w)$, ki pri splošnih koeficientih drži natančno takrat, ko je f linearna transformacija, zato podobne naravne transformacije ne moremo definirati v kategoriji **Set**, kjer so morfizmi poljubne funkcije. Sedaj fiksirajmo η_V in poglejmo, kakšnim enačbam mora zadostiti μ_V. Upoštevajmo pogoja:

(1) $$\mu_V \circ \eta_{TV} = id_{TV} \quad \text{in} \quad \mu_V \circ T\eta_V = id_{TV}.$$

Vzemimo $(x, y) \in V \times V$ in poglejmo, kakšen predpis mora imeti μ_V:

$$\mu_V \circ \eta_{TV}(x, y) = \mu_V(\alpha(x, y), \beta(x, y)) = \mu_V((\alpha x, \alpha y), (\beta x, \beta y)) = (x, y)$$
$$\mu_V \circ T\eta_V(x, y) = \mu_V(\eta_V(x), \eta_V(y)) = \mu_V((\alpha x, \beta x), (\alpha y, \beta y)) = (x, y)$$

Ko v enačbah μ_V ustrezno zamenjamo z $\widetilde{\mu_V}$, definirano kot splošno matriko, dobimo naslednje formule, ki jim morajo zadoščati koeficienti matrike:

x; prva:	y; prva:
$a_1\alpha + a_3\beta = 1$	$b_4\beta + b_2\alpha = 1$
$a_2\alpha + a_4\beta = 0$	$b_3\beta + b_1\alpha = 0$
x; druga:	y; druga:
$a_1\alpha + a_2\beta = 1$	$b_4\beta + b_3\alpha = 1$
$a_3\alpha + a_4\beta = 0$	$b_2\beta + b_1\alpha = 0$

Pri izračunu teh formul je dobro opaziti, da formule za b sledijo direktno iz tistih za a. In sicer na način $(\alpha, a_1, a_2, \ldots, b_4, \beta) \mapsto (\beta, b_4, b_3, \ldots, a_1, \alpha)$ prezrcalimo formulo. Oglejmo si, kakšne pogoje za koeficiente matrike nam porodijo enačbe, če vzamemo $\alpha \neq 0$ in $\beta \neq 0$:

$$\widetilde{\mu_V} = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \end{bmatrix} = \begin{bmatrix} \frac{1-a\beta}{\alpha} & a & a & -\frac{\alpha}{\beta}a \\ -\frac{\beta}{\alpha}b & b & b & \frac{1-b\alpha}{\beta} \end{bmatrix},$$

kjer sta a in b poljubna parametra. Upoštevajmo še pogoj:

(2) $\quad \mu_V \circ \mu_{TV} = \mu_V \circ T\mu_V$ na $(((x_1, x_2), (x_3, x_4)), ((x_5, x_6), (x_7, x_8))) \in V^8$.

Nastavimo enačbo in iščemo pogoje za predpis μ_V:

$$(\mu_V \circ \widetilde{\mu_{TV}})((x_1, x_2), (x_3, x_4), (x_5, x_6), (x_7, x_8)) = \\ (\mu_V \circ T\widetilde{\mu_V})((x_1, x_2, x_3, x_4), (x_5, x_6, x_7, x_8))$$

Kar nam po daljšem računu poda neslednje formule, ki jim morajo zadoščati koeficienti matrike $\widetilde{\mu_V}$:

$$\begin{aligned}
x_1&: \quad a_1a_1 + a_3b_1 = a_1a_1 + a_2b_1 \\
x_2&: \quad a_2a_1 + a_4b_1 = a_1a_2 + a_2b_2 \\
x_3&: \quad a_1a_2 + a_3b_2 = a_1a_3 + a_2b_3 \\
x_4&: \quad a_2a_2 + a_4b_2 = a_1a_4 + a_2b_4 \\
x_5&: \quad a_1a_3 + a_3b_3 = a_3a_1 + a_4b_1 \\
x_6&: \quad a_2a_3 + a_4b_3 = a_3a_2 + a_4b_2 \\
x_7&: \quad a_1a_4 + a_3b_4 = a_3a_3 + a_4b_3 \\
x_8&: \quad a_2a_4 + a_4b_4 = a_3a_4 + a_4b_4
\end{aligned} \qquad \begin{aligned}
x_8&: \quad b_4b_4 + b_2a_4 = b_4b_4 + b_3a_4 \\
&\vdots
\end{aligned}$$

Tokrat nismo zapisali vseh enačb. Enačbe na desni sledijo iz levih, samo prezrcalimo jih $(a_1, a_2, \ldots, b_4) \mapsto (b_4, b_3, \ldots, a_1)$. Sedaj združimo pogoja (1) in (2). Ker velja $a_2 = a_3 = a$, $b_2 = b_3 = b$ ter $a_2b_2 = a_4b_1 = ab$, se za večino enačb hitro vidi, da so izpolnjene, ko velja pogoj (1). Na levi moramo tako poračunati le enačbi pri x_4 in x_7, vendar se izkaže, da tudi ti ne data dodatne informacije o koeficientih matrike. Po premisleku ugotovimo, da tudi desna stran ne vrne dodatnih pogojev. Na desni bi morali sicer poračunati enačbi x_5 in x_2, kar pa ni potrebno, saj tudi rezultati enačb sledijo iz levih. Tako je pogoj (2) že popolnoma izpolnjen z upoštevanjem

pogoja (1). Torej je splošna oblika naravne transformacije μ za podano naravno transformacijo η s predpisom $\eta_V(x) = (\alpha x, \beta x)$, kjer sta $\alpha \neq 0$, $\beta \neq 0$, enaka:

$$\widetilde{\mu_V} = \begin{bmatrix} \frac{1-a\beta}{\alpha} & a & a & -\frac{\alpha}{\beta}a \\ -\frac{\beta}{\alpha}b & b & b & \frac{1-b\alpha}{\beta} \end{bmatrix}$$

Če parametra a in b nastavimo na 0, torej $a = b = 0$, dobimo:

$$\widetilde{\mu_V} = \begin{bmatrix} \frac{1}{\alpha} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\beta} \end{bmatrix}$$

Pri tem za $\eta_V(x) = (x, x)$ dobimo monado, ki smo jo našli že na začetku na strani 17. Sedaj za η spet vzemimo $\eta_V(x) = (x, x)$, torej $\alpha = \beta = 1$ ter nastavimo parametera $a = 1$ in $b = 0$. Tako dobimo $\mu_V((x, y), (z, w)) = (y + z - w, w)$. Bralec naj se sam prepriča, da je v tem primeru (T, η, μ) res monada, kar sicer sledi iz zgornjih formul.

Sedaj si moramo še ogledati, kakšno monado lahko dobimo, če sta α ali β enaka nič. Oba ne moreta biti enaka nič, ker mora veljati pogoj $\mu_V \circ \eta_{TV} = id_{TV}$. Pa si poglejmo samo možnost, da je $\beta = 0$. Možnost $\alpha = 0$ je namreč zelo podobna. Torej za $\alpha \neq 0$, $\beta = 0$ je $\eta_V(x) = (\alpha x, 0)$. S pomočjo formul iz pogoja (1) ugotovimo, da je μ oblike:

$$\widetilde{\mu_V} = \begin{bmatrix} \frac{1}{\alpha} & 0 & 0 & c_1 \\ 0 & \frac{1}{\alpha} & \frac{1}{\alpha} & c_2 \end{bmatrix}$$

Pri tem sta c_1 in c_2 poljubni konstanti. Sedaj uporabimo formule iz pogoja (2). Ko upoštevamo $a_2 = a_3 = b_1 = 0$ in $a_1 = b_2 = b_3$, ugotovimo, da formule ne dajo nobenih dodatnih omejitev. Izračun bomo izpustili, sicer pa je potrebno preveriti tako leve kot desne formule, pri čemer je izračun levih precej enostavnejši. Tako smo spet dobili družino možnih monad. Kot primer zapišimo η in μ za $\alpha = 2$, $c_1 = 5$ in $c_2 = 7$. Torej $\eta_V(x) = (2x, 0)$ in $\mu_V((x, y), (z, w)) = (\frac{1}{2}x + 5w, \frac{1}{2}y + \frac{1}{2}z + 7w)$.

Poiskali smo nekaj monad, katerih sestavni del je funktor, definiran kot $T : \underline{\mathbf{Vec}} \longrightarrow \underline{\mathbf{Vec}}$, ki slika vektorske prostore $T : V \longmapsto V \times V$ in linearne transformacije $T : f \longmapsto (f, f)$. Obstaja pa še mnogo drugih funktorjev $T : \underline{\mathbf{Vec}} \longrightarrow \underline{\mathbf{Vec}}$, z drugačnim predpisom, ki so tudi sestavni del monade. Dobri so recimo funktorji, ki slikajo vektorske prostore $T : V \longmapsto V^n$, kjer je $n \in \mathbb{N}$, in linearne transformacije $T : f \longmapsto (f, \ldots, f)$. Za te funktorje bi spet lahko poiskali primere monad, katerih sestavni del so.

Možno bi bilo definirati tudi funktor, ki slika vektorske prostore $T : V \longmapsto V \times A$, kjer je A izbran vektorski prostor, in linearne transformacije $T : f \longmapsto (f, id_A)$. Tudi tu lahko definiramo taki naravni transformaciji η, μ, da dobimo monado.

Treba pa je opozoriti, da ni vsaka ideja dobra. Oglejmo si primer, ki ne zadošča pogojem monade. Naj bo $F : \underline{\mathbf{Vec}} \longrightarrow \underline{\mathbf{Vec}}$. Naj F slika vektorske prostore $F : V \longmapsto V \times V$ in linearne transformacije $F : f \longmapsto (f, 3f)$. To ni monada, ker F ne zadošča pogojem za funktor. Komponenta F na morfizmih namreč ne slika enote v enoto.

Opomba 3.3. Potrebno je premisliti, kako je z obsegi, nad katerimi delamo. Najbolj zanesljivo je, če že na začetku namesto $\underline{\mathbf{Vec}}$ za kategorijo vzamemo $\underline{\mathbf{Vec}}_O$, kjer je O nek konkreten obseg. Razlog za to je, da moramo kot koeficiente matrike η in $\widetilde{\mu_V}$ vzeti elemente, ki so vsebovani v vseh obsegih, zajetih v kategoriji $\underline{\mathbf{Vec}}$.

Za več zgledov monade nad vektorskimi prostori si lahko ogledate [5], kjer najdete še mnogo drugih predvsem matematično zanimivih monad.

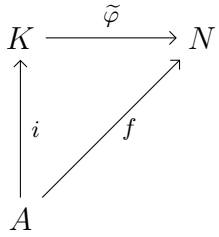
3.3. **Prosti monoid in njegova monada.** Najprej si poglejmo primer prostega monoida. Za množico A vzemimo abecedo, katere elementi so črke. Prosti monoid $M(A)$ nad množico $A = \{a, b, c, ...\}$ so vse možne besede, ki jih lahko tvorimo iz teh črk. Pogoj je, da so končne, ni pa nujno, da imajo kakšen pomen. Tako je na primer beseda tudi $[f, g, h, t, t, t, t]$. Oglate oklepaje pišemo, da lahko zapišemo enoto kot prazno besedo $[]$, tako brez črk. Na tej množici definiramo operacijo $\bullet$ kot stik besed. Oglejmo si primer $[a, v, t, o] \bullet [m, a, t] = [a, v, t, o, m, a, t]$. V splošnem definiramo stik dveh besed α, β kot $\alpha \bullet \beta = \alpha\beta$. Zlahka preverimo asociativnost. Torej je $M(A)$ res monoid, rečemo mu prosti monoid nad A.

V zgornjem primeru smo za množico A vzeli množico črk abecede. V splošnem lahko za A vzamemo katerokoli množico. Tako tudi dobimo konstrukcijo splošnega prostega monoida. Poglejmo si, kako definiramo prosti monoid s pomočjo teorije kategorij. Najprej se spomnimo pozabljivega funktorja, ki ga bomo tokrat uporabili na **Mon**, da se znebimo operacije. Pozabljivi funktor $U : \underline{\mathbf{Mon}} \longrightarrow \underline{\mathbf{Set}}$ slika monoide $U : (N, *) \longmapsto N$ in homomorfizme $U : (K, *) \xrightarrow{\varphi} (N, *) \longmapsto K \xrightarrow{\widetilde{\varphi}} N$; zadnje zaradi jasnosti zapišimo še s Hom notacijo: $U : \varphi \in \mathrm{Hom}((K, *), (N, *)) \longmapsto \widetilde{\varphi} \in \mathrm{Hom}(K, N)$. Pozabljivi funktor slika homomorfizem φ v funkcijo $\widetilde{\varphi}$ z enakim predpisom. Zaradi potrebe razločevanja nad funkcijami pišemo "kačo". Sedaj imamo zadosti sredstev, da definiramo prosti monoid s pomočjo teorije kategorij.

Definicija 3.4. Monoid $(K, *)$ s funkcijo $i : A \longrightarrow K$ je *prost nad* A, če velja naslednja *univerzalna lastnost*: Za vsak monoid $(N, *)$ in za vsako funkcijo $f : A \longrightarrow N$, obstaja natančno določen homomorfizem $\varphi : (K, *) \longrightarrow (N, *)$, da velja $\widetilde{\varphi} \circ i = f$. Za preglednost vse ponazorimo še z diagramom:
Mon :

$$(K, *) \overset{\varphi}{\dashrightarrow} (N, *)$$

Set :



Univerzalna lastnost pove, da obstaja natanko en homomorfizem φ, da zgornji diagram v **Set** komutira.

Opomba 3.5. S $\widetilde{\varphi}$ je označena funkcija, v katero pozabljivi funktor slika homomorfizem φ. Spomnimo, da velja $\widetilde{\varphi}(x) = \varphi(x)$.

Izrek 3.6. *Monoid $M(A)$ zapišimo kot $(M_{(A)}, \bullet)$. Za funkcijo $\eta_A : A \longrightarrow M_{(A)}$ s predpisom $\eta_A : a \longmapsto [a]$ je $M(A)$ prosti monoid nad A.*

Dokaz. Za poljuben monoid $(N, *)$ in funkcijo $f : A \longrightarrow N$ definirajmo homomorfizem $\varphi : (M_{(A)}, \bullet) \longrightarrow (N, *)$ kot:

$$\varphi([]) = e_N$$
$$\varphi([a]) = f(a)$$

Definicija je smiselna, saj mora veljati $\widetilde{\varphi} \circ \eta_A = f$. Ko to uporabimo na a, dobimo $\widetilde{\varphi}([a]) = f(a)$. Vemo pa, da velja $\varphi([a]) = \widetilde{\varphi}([a])$. Poglejmo sedaj, kako je φ

definiran na splošnem elementu $[a_1, \ldots, a_n] \in (M_{(A)}, \bullet)$. Definicija sledi iz tega, da je homomorfizem:

$$\begin{aligned}\varphi([a_1, \ldots, a_n]) &= \varphi([a_1 \bullet \cdots \bullet a_n]) \\ &= \varphi([a_1]) * \cdots * \varphi([a_n]) \\ &= f(a_1) * \cdots * f(a_n)\end{aligned}$$

Upoštevali smo, da se vsak element lahko zapiše z elementi $a \in A$ in to na enoličen način. Do sedaj smo pokazali, da tak φ zares obstaja. Sedaj pokažimo, da je natanko eden. Recimo torej, da imamo še homomorfizem ψ, ki tudi naredi diagram komutativen. Potem velja $\psi([a]) = f(a)$. Upoštevajmo, da je ψ homomorfizem:

$$\psi([a_1, \ldots, a_n]) = \ldots = f(a_1) * \cdots * f(a_n)$$

Iz tega sledi $\psi([a_1 \ldots a_n]) = \varphi([a_1 \ldots a_n])$, zato je $\psi = \varphi$. S tem smo dokazali, da je φ res natančno določen in obstaja. Monoid $(M_{(A)}, \bullet)$ s funkcijo $\eta_A : A \longrightarrow M_{(A)}$ zato zadošča univerzalni lastnosti za množico A. □

Definirajmo sedaj funktor, ki množici A priredi njen prosti monoid. Naj bo $M : \underline{\mathbf{Set}} \longrightarrow \underline{\mathbf{Mon}}$, s predpisom na objektih $M : A \longrightarrow (M_{(A)}, \bullet)$ in morfizmih $M : f \longrightarrow Mf$. Pri tem je $(M_{(A)}, \bullet)$ prosti monoid nad A, homomorfizem Mf pa ima naslednji predpis $Mf([a_1, \ldots, a_n]) = [f(a_1), \ldots, f(a_n)]$. Preverimo, da je M res funktor. Res slika množice $A \in \underline{\mathbf{Set}}$ v monoide $(M_{(A)}, \bullet) \in \underline{\mathbf{Mon}}$. Preveriti moramo še, da se tudi funkcije f slikajo v homomorfizme. Torej moramo pokazati, da je Mf res homomorfizem:

$$\begin{aligned}Mf([a_1, \ldots, a_n] \bullet [b_1, \ldots, b_m]) &= Mf([a_1, \ldots, a_n, b_1, \ldots, b_m]) \\ &= [f(a_1), \ldots, f(a_n), f(b_1), \ldots, f(b_m)] \\ &= [f(a_1), \ldots, f(a_n)] \bullet [f(b_1), \ldots, f(b_m)] \\ &= Mf([a_1, \ldots, a_n]) \bullet Mf([b_1, \ldots, b_m])\end{aligned}$$

To je primer, ko je bilo potrebno preveriti, da funktor res slika morfizme v morfizme, kar je ponavadi očitno že iz njegovega predpisa. Sedaj preverimo še usklajenost operacije $\circ$. Najprej preverimo veljavnost $M(f \circ g) = Mf \circ Mg$:

$$\begin{aligned}M(f \circ g)([a_1, \ldots, a_n]) &= [f(g(a_1)), \ldots, f(g(a_n))] \\ &= Mf([g(a_1), \ldots, g(a_n)]) \\ &= Mf(Mg([a_1, \ldots, a_n]))\end{aligned}$$

Preverimo še pogoj za enoto $M(id_A) = id_{M(A)}$:

$$\begin{aligned}M(id_A)([a_1, \ldots, a_n]) &= [id_A(a_1), \ldots, id_A(a_n)] \\ &= [a_1, \ldots, a_n]\end{aligned}$$

S tem smo preverili usklajenost operacije in pokazali, da je M res funktor.

Če funktor M komponiramo s pozabljivim funktorjem $U : \underline{\mathbf{Mon}} \longrightarrow \underline{\mathbf{Set}}$, dobimo dva nova funktorja $F = M \circ U$ in $T = U \circ M$. Na tak način definirana $F : \underline{\mathbf{Mon}} \longrightarrow \underline{\mathbf{Mon}}$ in $T : \underline{\mathbf{Set}} \longrightarrow \underline{\mathbf{Set}}$ sta res funktorja, kar sledi iz tega, da so funktorji morfizmi v kategoriji $\underline{\mathbf{Cat}}$. V nadaljevanju bomo posvetili pozornost funktorju T, ki porodi sezname in je zato del pomembne monade.

3.3.1. *Monada prostega monoida.* Oglejmo si sedaj kategorijo množic in na njej definirajmo monado s pomočjo prostega monoida. Definirajmo funktor $T : \underline{\mathbf{Set}} \longrightarrow \underline{\mathbf{Set}}$, kot $T = U \circ M$. Da je T funktor, nam ni potrebno preverjati, saj vemo, da je kompozitum dveh funktorjev spet funktor. Oglejmo si le njegov predpis:

$$\begin{aligned} T &: \underline{\mathbf{Set}} \longrightarrow \underline{\mathbf{Set}} \\ T &: A \longmapsto U(M(A)) \\ T &: f \longmapsto \widetilde{Mf} \end{aligned}$$

Zaradi krajšega zapisa bomo vseeno raje pisali TA namesto $U(M(A))$ in Tf namesto $\widetilde{Mf}$. Sedaj si oglejmo predpis funkcije Tf za $f : A \longrightarrow B$:

$$\begin{aligned} Tf &: TA \longrightarrow TB \\ Tf &: [a_1, \ldots, a_n] \longmapsto [f(a_1), \ldots, f(a_n)] \end{aligned}$$

Definirajmo naravno transformacijo $\eta : \mathrm{Id} \longrightarrow T$. Predpišimo delovanje po komponentah. Za vsako množico A definirajmo η_A kot:

$$\begin{aligned} \eta_A &: A \longrightarrow TA \\ \eta_A &: a \longmapsto [a] \end{aligned}$$

Definirajmo še naravno transformacijo $\mu : T \circ T \longrightarrow T$. Predpišimo delovanje po komponentah. Za vsako množico A definirajmo μ_A kot:

$$\begin{aligned} &\mu_A : T(TA) \longrightarrow TA \\ &\mu_A : [[a_{11}, \ldots, a_{1n}], \ldots, [a_{k1}, \ldots, a_{km}]] \longmapsto [a_{11}, \ldots, a_{1n}, \ldots, a_{k1}, \ldots, a_{km}] \end{aligned}$$

Sedaj za obe definiciji preverimo, da sta res naravni transformaciji. Preveriti moramo, da η in μ ustrezata enačbi za naravno transformacijo. Vzemimo poljubno funkcijo $f : A \longrightarrow B$. Za η mora biti izpolnjen pogoj: $Tf \circ \eta_A = \eta_B \circ f$. Vzemimo poljuben $a \in A$ ter preverimo, če res drži:

$$\begin{aligned} Tf \circ \eta_A(a) &= Tf([a]) = [f(a)] \\ \eta_B \circ f(a) &= \eta_B(f(a)) = [f(a)] \end{aligned}$$

Ker res drži za vsak a, smo s tem preverili, da je η naravna transformacija. Preverimo še pogoj za naravno transformacijo μ. Veljati mora $Tf \circ \mu_A = \mu_B \circ TTf$. Vzemimo poljuben $x = [[a_{11}, \ldots, a_{1n}], \ldots, [a_{k1}, \ldots, a_{km}]] \in TTA$ ter preverimo pogoj:

$$\begin{aligned} Tf \circ \mu_A(x) &= Tf([a_{11}, \ldots, a_{1n}, \ldots, a_{k1}, \ldots, a_{km}]) \\ &= [f(a_{11}), \ldots, f(a_{1n}), \ldots, f(a_{k1}), \ldots, f(a_{km})] \end{aligned}$$

$$\begin{aligned} \mu_B \circ TTf(x) &= \mu_B([Tf([a_{11}, \ldots, a_{1n}]), \ldots, Tf([a_{k1}, \ldots, a_{km}])]) \\ &= \mu_B([[f(a_{11}), \ldots, f(a_{1n})], \ldots, [f(a_{k1}), \ldots, f(a_{km})]) \\ &= [f(a_{11}), \ldots, f(a_{1n}), \ldots, f(a_{k1}), \ldots, f(a_{km})] \end{aligned}$$

Res je tudi μ naravna transformacija. Če želimo, da je (T, η, μ) monada, morata biti izpolnjena še dva pogoja. Najprej preverimo pogoj $\mu_A \circ \eta_{TA} = id_{TA} = \mu_A \circ T\eta_A$. Vzemimo poljuben element $[a_1, \ldots, a_n] \in TA$ in preverimo pogoj:

$$\begin{aligned} \mu_A \circ \eta_{TA}([a_1, \ldots, a_n]) &= \mu_A([[a_1, \ldots, a_n]]) \\ &= [a_1, \ldots, a_n] \end{aligned}$$

$$\begin{aligned}\mu_A \circ T\eta_A([a_1, \ldots, a_n]) &= \mu_A([\eta_A(a_1), \ldots, \eta_A(a_n)]) \\ &= \mu_A([[a_1], \ldots, [a_n]]) \\ &= [a_1, \ldots, a_n]\end{aligned}$$

Pogoj je izpolnjen, preverimo še $\mu_A \circ \mu_{TA} = \mu_A \circ T\mu_A$. Vzemimo poljuben $x \in T^3A$ in preverimo, če drži enakost $\mu_A \circ \mu_{TA}(x) = \mu_A \circ T\mu_A(x)$. Tega na tem mestu za splošen x ne bomo naredili, ampak samo za konkreten primer. Recimo, da je $A = \mathbb{N}$ in $x = [[[1, 5, 7], [15], [2]], [[3]]]$. Izračunajmo, da za ta x res drži enakost.

$$\begin{aligned}\mu_A \circ \mu_{TA}(x) &= \mu_A([[1, 5, 7], [15], [2], [3]]) \\ &= [1, 5, 7, 15, 2, 3]\end{aligned}$$

$$\begin{aligned}\mu_A \circ T\mu_A(x) &= \mu_A([\mu_A([[1, 5, 7], [15], [2]]), \mu_A([[3]]))]) \\ &= \mu_A([[1, 5, 7, 15, 2], [3]]) \\ &= [1, 5, 7, 15, 2, 3]\end{aligned}$$

Enakost res drži. Podobno dobimo tudi za splošno množico A in poljuben $x \in T^3A$, le da je pri izračunu več pisanja. Predstavljamo pa si lahko, da se v splošnem odpravljajo oklepaji [] v različnem vrstnem redu, kar pa nas na koncu vseeno vedno pripelje do istega rezultata. Ker je zadoščeno tudi temu pogoju, je (T, η, μ) res monada.

4. Funkcijsko programiranje in Haskell

Haskell je funkcijski programski jezik. Njegovo bistvo so funkcije, ki se obnašajo tako kot funkcije v matematiki. To pomeni, da je za vsako funkcijo znano, kaj so njeni argumenti, torej kaj sprejme in kaj vrača. Pri tem ni nobenih zunanjih dejavnikov, ki bi vplivali na izračun rezultata. Pri istih argumentih tako Haskellove funkcije vedno vrnejo isti rezultat.

Za vsako funkcijo tudi natančno vemo, kakšnega tipa je. Tip funkcije nam pove kakšne tipe sprejema in kakšne vrača kot rezultat. Tudi za ostale izraze vemo, kakšnega tipa so. Če recimo napišemo `a=1`, kasneje ne moremo napisati `a="beseda"`, saj nismo vedno priredili istega tipa, velja celo še več, sploh ne moremo spreminjati vrednosti `a`. To, kakšnega tipa so vrednosti, nam ni treba eksplicitno povedati, kot moramo to storiti recimo v programskem jeziku C. Vendar pa se tipi med programom ne morejo spreminjati in ostanejo enaki. To omogoča prevajalniku delno preverjanje pravilnosti napisanega. Ena dobrih lastnosti Haskella je, da lahko kar vprašamo prevajalnik, kakšnega tipa naj bi funkcije bile, ta pa tip samostojno izračuna.

V osnovnih paketih Haskella kazalcev ni možno uporabljati. Tako ne moremo kar zamenjati n-ti element seznama s kakšnim drugim. Vsakič, ko želimo v Haskellu nek objekt spremeniti, moramo objekt na novo ustvariti. Na primer pri seznamu moramo za zamenjavo elementov seznama narediti celoten seznam na novo. Kot pozitivno lastnost seznamov omenimo, da imajo lahko tudi neskončno elementov, ki se izračunajo šele takrat, ko jih potrebujemo.

Haskell odlikuje tudi možnost programiranja z vzorci, kar nam lahko močno olajša delo. Jezik je tudi zelo hiter, predvsem zaradi zgoraj naštetih lastnosti. Vse našteto ga naredi za dober programski jezik. Primeren je predvsem za implementacijo matematičnih rezultatov, kar v mnogih programskih jezikih ni mogoče ali pa je zelo oteženo. Več o Haskellu si lahko preberete na spletu [8].

5. Motivacija za monado v Haskellu

Proučevali bomo monade v funkcijskem programiranju. To je naraven koncept, ki se pojavi pri programiranju. Pri razvoju programiranja, predvsem funkcijskega, so že zgodaj odkrili, da se pri nekaterih operacijah v programskih jezikih vedno znova pojavljajo podobni koncepti. Ko odstranimo konkretnost nekaterih rešitev, se izkaže, da smo v ozadju programa uporabili koncept monade. Pri programskem jeziku Haskell je struktura monade pri problemih vidna, saj je jezik zgrajen s tem namenom, da se lahko čim več konceptov posploši. Oglejmo si nekaj primerov operacij v Haskellu, kjer je v ozadju ideja monade, kje točno, bomo ugotovili kasneje. Pri primerih pišemo dvojni enačaj za enake izraze. Vsa koda je delujoča in jo je možno prepisati v Haskell. Najprej si oglejmo delovanje funkcije `map`:

```
map (*3) [2,3,5] == [6,9,15]
```

Funkcija `map` v zgornjem primeru poskrbi, da se funkcija `(*3)` izvede znotraj oklepaja na vsakem elementu posebej. Na mestu funkcije `(*3)` imamo lahko poljubno funkcijo f, funkcija `map` pa vedno poskrbi, da se uporabi znotraj seznama na elementih. Sedaj si oglejmo delovanje funkcije `concat`:

```
concat [[1,2],[5]] == [1,2,5]
```

Funkcija `concat` deluje na seznamu seznamov. Kot rezultat vrne seznam, katerega elementi so elementi podseznamov prvotnega seznama. Oglejmo si še sintakso anonimne funkcije, abstrakcije lambda v Haskellu:

```
(λx→3*x^2) 7 == 147
((*3).(λx→x^2)) 7 == 147
```

V prvi vrstici anonimna funkcija uporabljena na 7 vrne $3 \cdot 7^2 = 147$. V drugem primeru najprej komponiramo dve funkciji, množenje s 3 in kvadriranje. Funkcijo, ki jo dobimo, nato uporabimo na 7. Seveda dobimo isti rezultat, saj je množenje s 3 komponirano s kvadriranjem ravno prvotna anonimna funkcija. Poglejmo si še nekaj primerov uporabe funkcije `map`:

```
map (λx → x^2) [2,3,5] == [4,9,25]
map (*3) (map (λx → x^2) [2,3,5]) == [12,27,75]
map ((*3).(λx → x^2)) [2,3,5] == [12,27,75]
```

Zadnji dve vrstici sugerirata, da je enako, če "mapiramo" dve funkciji vsako posebej ali pa najprej komponiramo funkciji in jih nato "mapiramo". Sledi še prikaz delovanja funkcij `concat` in `map` skupaj:

```
map (map (*3)) [[1,2],[5]] == [[3,6],[15]]
concat (map (map (*3)) [[1,2],[5]]) == [3,6,15]
map (*3) (concat [[1,2],[5]]) == [3,6,15]
```

Druga in tretja vrstica napeljujeta, da je enako, če najprej uporabimo funkcijo na elementih podseznamov seznama in nato uporabimo `concat`, ali pa najprej uporabimo `concat` ter nato "mapiramo" funkcijo na dobljen seznam. Sedaj si oglejmo še primer funkcije, ki jo lahko sprogramiramo sami:

```
kompleksniKoren:: Int → (Double,Double) → [(Double,Double)]
kompleksniKoren m (radij,kot) = resitev
    where
        n = (fromIntegral m)::Double    --pretvori v Double
        r = radij**(1/n)                --izracun n tega realnega korena
```

```
resitev = map (λx → (r,(kot+2*x*pi)/n)) [0..n-1]
```

```
f = kompleksniKoren 2
g = kompleksniKoren 3
```

To je delujoča koda za funkcijo, ki za kompleksno število, zapisano v polarnem zapisu, vrne seznam želenih korenov. V prvi vrstici je podan tip funkcije. Funkcija kot argument najprej sprejme naravno število m, to je m-ti koren, ki ga želimo izračunati. Nato kot argument sprejme dvojico, kjer je prva komponenta radij r, druga pa kot φ kompleksnega števila, zapisanega v polarnem zapisu. Funkcija kot rezultat vrne seznam kopleksnih števil, zapisanih v polarnem zapisu. Definirani sta tudi funkciji f in g. Funkcija f izračuna oba kvadratna korena kompleksnega števila, funkcija g pa izračuna tretje korene kompleksnega števila. Obe podata rezultate v obliki seznama. Oglejmo si izračun tretjih korenov števila 8:

```
g (8,0) == [(2.0,0.0),(2.0,2.0943951023931953),(2.0,4.1887902047863905)]
```

Zadnji rezultat v seznamu predstavlja polarni zapis $r = 2$, $\varphi = \frac{4\pi}{3}$ za kompleksno število $2 \cdot (\cos(\frac{4\pi}{3}) + i \cdot \sin(\frac{4\pi}{3}))$.

Oglejmo si sedaj naslednji problem. Če imamo realni funkciji $\widetilde{f}(x) = \sqrt[2]{x}$ in $\widetilde{g}(x) = \sqrt[3]{x}$, potem zlahka izračunamo šesti koren realnega števila x kot $(\widetilde{f} \circ \widetilde{g})(x) = \sqrt[6]{x}$. Kako pa lahko to naredimo za naši kompleksni funkciji f in g, ki vrneta seznam večih rezultatov? Ne moremo ju kar komponirati, saj se kodomena ene ne ujema z domeno druge. Lahko pa na rezultat prve "mapiramo" drugo funkcijo. Tako dobimo:

```
map f (g (8,0)) == [[(1.414213562373095,0.0),(1.41421356,3.14159265)],
                    [(1.41421356,1.04719755),(1.41421356,4.18879020)],
                    [(1.41421356,2.09439510),(1.41421356,5.23598775)]]
```

Nismo dobili tega, kar smo želeli, saj smo namesto seznama šestih korenov dobili seznam podseznamov šestih korenov. Rezultat spravimo v želeno obliko, če nad njim uporabimo funkcijo `concat`. Tako kot rezultat dobimo seznam šestih korenov kompleksnega števila 8. Do enakega rezultata bi prišli tudi z uporabo operacije $\gg\!=$ (bind), ki se za te namene v Haskellu pogosteje uporablja. Razlog je v tem, da je zapis krajši in še miselni tok za operacijo $\gg\!=$ je od leve proti desni.

```
(g (8,0)) >>= f == concat (map f (g (8,0)))
```

Več motivacije lahko najdete na spletu, recimo na [6].

6. Definicija monade v Haskellu

Ogledali si bomo le poenostavljeno definicijo Haskell monade, kot je podana tudi na spletu [7]. Celotna definicija vsebuje nekaj dodatnih operacij. Ena od takih je recimo `fail`, ki ponuja obravnavo izjem. Celotno definicijo Haskell monade si lahko ogledate na spletu [8]. Zreducirana in poenostavljana definicija Haskell monade je še vedno povsem uporabna in bolj pregledna:

```
class  Monad m  where
    (>>=)           :: m a → (a → m b) → m b
    return          :: a → m a
```

V definiciji predstavlja `a` oznako za osnovni tip oblike A, oznaka `m a` pa ponazarja tip, nad katerim je bil uporabljen funktor, torej tip oblike TA. Pri tem so zakoni, ki jim morata ustrezati `return` in $\gg\!=$ (bind), naslednji:

```
w >>= return  =  w
return x >>= f  =  f x
(w >>= f) >>= g  =  w >>= (λx → f x >>= g)
```

Malo razčlenimo definicijo monade. Najprej povejmo, da je $\gg\!=$ binarna operacija in sicer taka, ki na levi sprejme objekt na desni pa morfizem. Funkcija `return` pa deluje le na objektih, kar lahko razberemo tudi iz zapisa tipa, ki je za `return` enak `a → m a`. To v resnici pomeni naslednje: `return`: $A \longrightarrow TA$, kjer je T funktor. V Haskellu predstavljata `a` in `m a` tip. Torej funktor T slika tip objektov `a` v tip `m a`. Na konkretnem elementu pa je `return` komponenta naravne transformacije. To bomo dokazali kasneje iz Haskellovih zakonov za monado. Najlažje je, če na tem mestu kar definiramo `return` in $\gg\!=$ z matematičnim zapisom, v naslednjem poglavju pa bomo dokazali, da je to zares ustrezno.

```
return := η
```

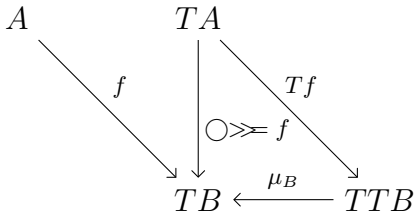
Res `return`: $A \longrightarrow TA$ in $\eta : A \longrightarrow TA$. Sedaj rečemo, da imata še enak predpis. Operacijo $\gg\!=$ lahko definiramo tudi kot funkcijo:

$$\gg\!=: TA \times (A \longrightarrow TB) \longrightarrow TB$$

To pove tudi tip v Haskellu `m a → (a → m b) → m b`, ki pravi, da je $(\gg\!=)$ funkcija, ki sprejme kot prvi argument element tipa `m a`, kot drugi argument funkcijo tipa `a → m b` in kot rezultat vrne element tipa `m b`. Vidimo, da je tipom res zadoščeno, ko definiramo $\gg\!=$ kot:

```
w >>= f := μ(Tf(w))
```

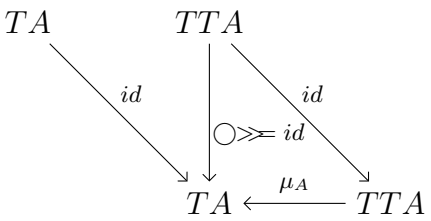
Za $f : A \longrightarrow TB$ ponazorimo to še z diagramom:



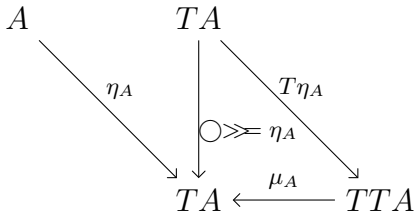
Tu je s krogcem ◯ označeno mesto, kjer $\gg\!= f$ sprejme svoj argument. Velja torej:

$$\bigcirc\!\gg\!= f = (\lambda : x \mapsto x \gg\!= f)$$

Oglejmo si še, kaj dobimo, če za f vzamemo kar id_{TA}. To smemo, saj je id_{TA} res tipa `a → m b`, za `a` namreč vzamemo TA, za `b` pa vzamemo A in res je `m b` enak TA. Prikažimo to še z diagramom:



Na ta način ugotovimo, da je $\mu_A = \bigcirc \gg\!= id_{TA}$. Poglejmo še, kaj dobimo, če za f vzamemo η_A:



Iz zakona monade $\mu_A \circ T\eta_A = id_{TA}$ sledi, da je $\bigcirc \gg\!= \eta_A = id_{TA}$. To pa je ravno prvi zakon Haskell monade:

```
w >>= return  =  w
```

S pomočjo diagramov lahko poiščemo še ostale relacije, ki veljajo za $\gg\!=$. Dokaze, ki jih bomo naredili v naslednjem poglavju, bi lahko dokazali tudi s pomočjo diagramov. Mi tega ne bomo storili, ker bi jih bilo preveč, so pa dobra pomoč, če bi pri dokazovanju slučajno zašli v težave.

6.1. **Primera Haskell monad.**

6.1.1. *Monada List.* Kot primer si oglejmo definicijo monade List v Haskellu:

```
instance Monad [] where
    return :: a → [a]
    return x = [x]

    (>>=) :: [a] → (a → [b]) → [b]
    xs >>= f = concat (map f xs)
```

Ker smo jo srečali že v motivaciji na strani 25, na tem mestu le pokomentirajmo delovanje funkcije `return`. Funkcija `return` vloži element v seznam. Na številu 8 deluje na primer takole: `return 8 = [8]`.

6.1.2. *Monada Maybe.* Oglejmo si še definicijo monade Maybe v Haskellu:

```
instance Monad Maybe where
    return :: a → Maybe a
    return x  = Just x

    (>>=) :: Maybe a → (a → Maybe b) → Maybe b
    Nothing >>= f = Nothing
    Just x >>= f = f x
```

Ta definicija v celoti ustreza monadi za predstavitev izjem na strani 15. Pri tem so elementi A vrednosti tipa a, elementi TA pa vrednosti tipa `Maybe a`. Poglejmo si še vzporednico med elementi TA in vrednostmi tipa `Maybe a`. Element $(0, *)$ lahko enačimo z `Nothing` in $(1, x)$ z `Just x`, tu je `x` nek konkreten element in ne več tip.

Ugotovimo tudi smiselnost definicije `return` kot η. Namreč `return` :: `a` → `m a` in $\eta : A \longrightarrow TA$, bijektivno pa ustrezajo tudi predpisi na elementih. Sedaj si oglejmo še delovanje operacije $\gg\!=$. Definiramo jo kot $w \gg\!= f = \mu_B(Tf(w))$, kjer so f funkcije oblike $f : A \longrightarrow TB$.

Poglejmo še konkreten primer, kako uporabimo monado Maybe pri programiranju. Recimo, da imamo rezultat neke funkcije `w = Just x`, pri čemer je `x` tipa `a` in zato `w` tipa `Maybe a`. Ta rezultat želimo podati funkciji `f` :: `a` → `m b`. Tega ne

moremo storiti direktno, ampak s pomočjo operacije `>>=`. Pri izračunu kot rezultat dobimo `f x`, ki je oblike `Just y` ali `Nothing`. Pri tem `Nothing` ponazarja, da je pri izračunu prišlo do napake oziroma izjeme. Tak rezultat lahko vrnemo recimo, če v matematičnem izračunu dobimo deljenje z 0.

7. Ekvivalentnost definicij

Oglejmo si lastnosti abstrakcije lambda, ki jih bomo potrebovali v dokazu. Naj bo $\lambda : x \mapsto f(x)$. Če dano abstrakcijo lambda uporabimo na argumentu a, dobimo $f(a)$. Naslednja lastnost je veljavnost $g \circ (\lambda : x \mapsto f(x)) = \lambda : x \mapsto (g \circ f)(x)$. Na lastnosti abstrakcije lambda se bomo sklicevali z (lam). Pokažimo še, da zadnja lastnost res drži:

$$\begin{aligned}(g \circ (\lambda : x \mapsto f(x)))(a) &= g((\lambda : x \mapsto f(x))(a)) \\ &= g(f(a)) = (g \circ f)(a) \\ &= (\lambda : x \mapsto (g \circ f)(x))(a)\end{aligned}$$

7.1. **Vpeljava Haskell monade.** S pomočjo matematične monade definirajmo programersko monado. Torej definirajmo monado, kot je definirana v Haskellu.

Definicija 7.1. Definirajmo Haskell monado z `return` in `>>=`, ki sta definirana s pomočjo matematične monade (T, η, μ) kot:

`return` $:= \eta$

`w >>= f` $:= \mu(Tf(w))$

Da je definicija res ustrezna, nam pove trditev 7.3.

Definicija 7.2. Definirajmo oznake zakonov monade, na katere se bomo sklicevali pri dokazovanju. Najprej sta označena pogoja za naravni transformaciji η, μ, nato sledita še dodatna pogoja:

(η) $Tf \circ \eta_A = \eta_B \circ f$
(μ) $Tf \circ \mu_A = \mu_B \circ T(Tf)$
(1) $\mu_A \circ T\eta_A = \mu_A \circ \eta_{TA} = id_{TA}$
(2) $\mu_A \circ T\mu_A = \mu_A \circ \mu_{TA}$

Pogoj, da je T funktor, pa bomo označili s (fun). Oznake so napisane na desni strani dokazov, kot pomoč, zakaj velja enakost pri dani vrstici.

Trditev 7.3. *Če privzamemo matematične zakone monade, torej zakone iz teorije kategorij, potem iz njih sledijo programerski zakoni iz Haskella:*

(a) `w >>= return = w`
(b) `return x >>= f = f x`
(c) `(w >>= f) >>= g = w >>= (λx → f x >>= g)`

Opomba 7.4. Zadnji pogoj se razume takole:

`(w >>= f) >>= g = w >>= (λx → ((f x) >>= g))`

Dokaz. Trditev bomo dokazali za vsako točko posebej. Pri dokazu bomo za objekte vzeli množice. Za ostale objekte morfizme namesto na elementih uporabimo direktno na objektih. Dokaz je nato enak. Lahko si tudi predstavljamo, da $a \in A$ pomeni, da je a tipa A. Dokaz bomo naredili ločeno po točkah (a), (b) in (c). Dokažimo, da velja prvi zakon:

`w >>= return = w`

Izberimo objekt A dane kategorije ter $w \in TA$. Zapišimo levo stran izraza v matematičnem jeziku in ga izračunajmo:

$$\begin{aligned} \mu_A(T\eta_A(w)) &= (\mu_A \circ T\eta_A)(w) \\ &= id_{TA}(w) && (\#1) \\ &= w \end{aligned}$$

Ker je rezultat enak desni strani, smo s tem prvi zakon dokazali. Sedaj dokažimo še drugi zakon:

`(return x) >>= f = f x`

Imamo $f : A \longrightarrow TB$. To pomeni, da za naravno transformacijo η velja $\eta_{TB} \circ f = Tf \circ \eta_A$, kar bomo tudi uporabili pri dokazu. Izračunamo levo stran:

$$\begin{aligned} \mu_B(Tf(\eta_A(x))) &= \mu_B((Tf \circ \eta_A)(x)) \\ &= \mu_B((\eta_{TB} \circ f)(x)) && (\#\eta) \\ &= ((\mu_B \circ \eta_{TB}) \circ f)(x) \\ &= (id_{TB} \circ f)(x) && (\#1) \\ &= f(x) \end{aligned}$$

Ker je leva stran enaka desni, smo dokazali tudi drugi zakon. Dokažimo še veljavnost tretjega zakona:

`(w >>= f) >>= g = w >>= (λx → f x >>= g)`

Naj bo $f : A \longrightarrow TB$ ter $g : B \longrightarrow TC$. Izračunamo levo stran:

$$\begin{aligned} [L] &= \mu_C(Tg(\mu_B(Tf(w)))) \\ &= (\mu_C \circ (Tg \circ \mu_B) \circ Tf)(w) \\ &= (\mu_C \circ (\mu_{TC} \circ T(Tg)) \circ Tf)(w) && (\#\mu) \\ &= ((\mu_C \circ \mu_{TC}) \circ T(Tg) \circ Tf)(w) \end{aligned}$$

Poračunamo še desno stran:

$$\begin{aligned} [D] &= \mu_C((T(\lambda : x \mapsto \mu_C(Tg(f(x)))))(w)) \\ &= \mu((T(\mu_C \circ (\lambda : x \mapsto Tg(f(x)))))(w)) && (\#\text{lam}) \\ &= \mu_C(((T\mu_C) \circ T(\lambda : x \mapsto Tg(f(x))))(w)) && (\#\text{fun}) \\ &= ((\mu_C \circ T\mu_C) \circ T(\lambda : x \mapsto Tg(f(x))))(w) \\ &= (\mu_C \circ \mu_{TC}) \circ T(\lambda : x \mapsto Tg(f(x))))(w) && (\#2) \\ &= ((\mu_C \circ \mu_{TC}) \circ T(Tg \circ (\lambda : x \mapsto f(x))))(w) && (\#\text{lam}) \\ &= ((\mu_C \circ \mu_{TC}) \circ T(Tg \circ f))(w) && (\#\text{lam}) \\ &= ((\mu_C \circ \mu_{TC}) \circ T(Tg) \circ Tf)(w) && (\#\text{fun}) \end{aligned}$$

Leva in desna stran sta enaki, zato velja tudi tretji zakon. S tem je vsem točkam trditve zadoščeno in trditev drži. □

Dokazali smo trditev 7.3, ki pove, da iz matematične definicije monade sledi Haskell definicija. Sedaj poskusimo s funkcijami iz Haskella izraziti matematične objekte. Naravna transformacija η je kar `return`. Kaj pa naravna transformacija μ in funktor T? Poskusimo ju izraziti iz enakosti:

`w` $\gg\!=$ `f` $= \mu(Tf(w))$

Najprej izrazimo μ. Naj bo $w \in TTA$, potem:

`w` $\gg\!=$ `id_TA` $= \mu_A(T(id_{TA})(w)) = \mu_A(id_{TTA}(w)) = \mu(w)$

Naj bo $g : A \longrightarrow B$ in $w \in TA$. Izrazimo še funktor T:

`w` $\gg\!=$ `return.g` $= \mu_B(T(\eta_B \circ g)(w)) = \mu_B((T\eta_B \circ Tg)(w)) = id_{TB} \circ Tg(w) = Tg(w)$

7.2. **Matematična monada iz Haskellove.** Poskusimo s pomočjo Haskellovih zakonov za monado dokazati matematične zakone. Za definicijo monade (T, η, μ), bomo vzeli kar rezultate, ki smo jih dobili pri izražavi matematičnih objektov.

Definicija 7.5. Definicija matematičnih objektov:

(i) $\eta :=$ `return`
(ii) $\mu(w) :=$ `join w := w` $\gg\!=$ `id`
(iii) $Tf(w) :=$ `fmap f w := w` $\gg\!=$ `return.f`

Definicija 7.6. Definirajmo oznake za zakone monade, ki veljajo v Haskellu:

```
(a) w >>= return  =  w
(b) return x >>= f  =  f x
(c) (w >>= f) >>= g  =  w >>= (λx → f x >>= g)
```

V dokazih se bomo na zakone sklicevali z (a), (b) in (c).

Trditev 7.7. *Če privzamemo programerske zakone monade, torej zakone, ki veljajo v Haskellu, potem iz njih sledi, da je T funktor.*

Dokaz. Potrebno je preveriti, da je T usklajen z operacijo $\circ$. Dokazujemo torej:

(1) $T(f \circ g) = T(f) \circ T(g)$
(2) $T(id_A) = id_{TA}$

Najprej dokažimo (1), torej:

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

Za levo stran samo razpišimo definicijo:

```
fmap (f.g) w = w >>= return.f.g
```

Sedaj izračunajmo desno stran:

```
fmap f (fmap g w) = (#def)
fmap f (w >>= return.g) = (#def)
(w >>= return.g) >>= return.f = (#c)
w >>= (λx → (return.g x >>= return.f)) = ()
w >>= (λx → (return (g x) >>= return.f)) = (#b)
w >>= (λx → return.f (g x)) = ()
w >>= (λx → return.f.g x) = (#lam)
w >>= return.f.g
```

Ker je leva stran identična desni, ugotovimo, da je pogoju (1) zadoščeno. Sedaj dokažimo še pogoj (2), torej pogoj za identiteto:

```
fmap id_A w = id_TA w
```

Izračunajmo levo stran:

```
fmap id_A w = (#def)
w >>= return.id_A = ()
w >>= return = (#a)
w
```

Izračunajmo še desno stran:

```
id_TA w = ()
w
```

Zadnji izračun se zdi trivialen, vendar moramo biti pozorni, da `id_TA w = w` zagotovo velja le v kategorijah, kjer so morfizmi preslikave, operacija kategorije klasičen kompozitum in je enota `id` kar identiteta. Haskell kategorija je taka, zato to drži. □

Trditev 7.8. *Iz Haskellovih zakonov sledi, da je η naravna transformacija, kar pomeni, da za vsak $f : A \longrightarrow B$ velja $Tf \circ \eta_A = \eta_B \circ f$.*

Dokaz. Zapišimo enačbo v Haskell sintaksi in dokažimo enakost:

```
(fmap f).return x = return.f x
```

Pokažimo, da je leva stran res enaka desni:

```
fmap f (return x) = (#def)
(return x) >>= return.f = (#b)
return.f x
```
□

Trditev 7.9. *Iz Haskellovih zakonov sledi, da je μ naravna transformacija, kar pomeni, da za vsak $f : A \longrightarrow B$ velja $Tf \circ \mu_A = \mu_B \circ T(Tf)$.*

Dokaz. Dokazati moramo enakost:

```
(fmap f).join w = join.(fmap (fmap f)) w
```

Izračunajmo levo stran:

```
fmap f (join w) = (#def)
(join w) >>= return.f = (#def)
(w >>= id) >>= return.f = (#c)
w >>= (λx → id x >>= return.f) = ()
w >>= (λx → x >>= return.f)
```

Izračunajmo še desno stran:

```
join (fmap (fmap f) w) = (#def)
join (w >>= return.(fmap f)) = (#def)
(w >>= return.(fmap f)) >>= id = (#c)
w >>= (λx → (return.(fmap f) x) >>= id) = ()
w >>= (λx → (return (fmap f x)) >>= id) = (#def)
w >>= (λx → (return (x >>= return.f)) >>= id) = (#b)
w >>= (λx → ( id (x >>= return.f) )) = ()
w >>= (λx → ( x >>= return.f ))
```

Ker sta enaki, trditev drži. □

Trditev 7.10. *Če privzamemo programerske zakone, torej zakone, ki veljajo za Haskell monado, potem iz njih sledi, da je* (T, η, μ) *monada.*

Dokaz. Preverili smo že, da je T funktor ter η in μ naravni transformaciji. Preverimo še dodatna pogoja:

(1) $\mu_A \circ T\eta_A = \mu_A \circ \eta_{TA} = id_{TA}$
(2) $\mu_A \circ T\mu_A = \mu_A \circ \mu_{TA}$

Vzemimo poljuben objekt A. Najprej preverimo prvi pogoj: $\mu_A \circ T\eta_A = id_{TA} = \mu_A \circ \eta_{TA}$. Dokazujemo:

```
join.(fmap return) w = w = join.return w
```

Najprej dokažimo levo stran:

```
join (fmap return w) = (#def)
join (w >>= return.return) = (#def)
(w >>= return.return) >>= id = (#c)
w >>= (λx → return.return x >>= id) = ()
w >>= (λx → return (return x) >>= id) = (#b)
w >>= (λx → id (return x)) = ()
w >>= (λx → return x) = (#lam)
w >>= return = (#a)
w
```

Sedaj dokažimo še desno:

```
join (return w) = (#def)
(return w) >>= id = (#b)
id w = ()
w
```

Dokažimo še drugi pogoj: $\mu_A \circ T\mu_A = \mu_A \circ \mu_{TA}$. V Haskell sintaksi torej dokazujemo:

```
join.(fmap join) n = join.join n
```

Izračunajmo levo stran:

```
join (fmap join n) = (#def)
join (n >>= return.join) = (#def)
(n >>= return.join) >>= id = (#def)
n >>= (λw → (return.join w >>= id)) = ()
n >>= (λw → (return (join w) >>= id)) = (#b)
n >>= (λw → (id (join w))) = ()
n >>= (λw → join w) = (#def)
n >>= (λw → (w >>= id))
```

Izračunajmo še desno stran:

```
join (join n) = (#def)
(join n) >>= id = (#def)
(n >>= id) >>= id = (#c)
n >>= (λw → (id w >>= id)) = ()
n >>= (λw → w >>= id)
```

Ker v obeh primerih dobimo enako, pogoj res drži. S tem smo dokazali, da je (T, η, μ) monada. □

7.3. **Izrek o ekvivalentnosti.**

Izrek 7.11 (Izrek o ekvivalentnosti)**.** *Ekvivalentno je, ali podamo monado na matematičen način kot* (T, η, μ) *ali kot Haskell monado s predpisom za* `return` *in* $\gg=$*. Monadi, ki ju pri tem dobimo, sta enaki.*

Dokaz. Dokazali smo že, da iz matematične definicije monade sledi, da sta `return` in $\gg=$ definirana kot `return` $= \eta$ in `w` $\gg=$ `f` $= \mu(Tf(w))$ ustrezna za Haskell monado. Potem smo z njima izrazili T, η, μ ter s programerskimi zakoni dokazali, da res ustrezajo matematični monadi (T, η, μ). Za dokaz ekvivalentnosti moramo samo še dokazati, da lahko s pomočjo programerskih zakonov izrazimo `return` in $\gg=$ na ustrezen način. Ustrezen način je ta, ki smo ga na začetku uporabili kot definicijo programerskih objektov.

Dokazati moramo, da se lahko s pomočjo programerskih zakonov iz naslednjih definicij matematičnih objektov izrazi programerske objekte na ustrezen način:

(i) $\eta :=$ `return`
(ii) $\mu(w) :=$ `join w := w` $\gg=$ `id`
(iii) $Tf(w) :=$ `fmap f w := w` $\gg=$ `return.f`

Za `return` je enostavno, le vzamemo η. Operacijo $\gg=$ pa si želimo na koncu dobiti izraženo kot `w` $\gg=$ `f` $= \mu(Tf(w))$. Izračunajmo, če to res dobimo. Izraz $\mu(Tf(w))$ bomo zapisali v Haskellovem zapisu ter ga s pomočjo programerskih zakonov zapisali v čim krajši obliki.

```
join (map f w) = (#def)
(map f w) >>= id = (#def)
(w >>= return.f) >>= id = (#c)
w >>= (λx → ((return.f x) >>= id))) = ()
w >>= (λx → ((return (f x)) >>= id)) = (#b)
w >>= (λx → (id (f x))) = ()
w >>= (λx → (f x))) = (#lam)
w >>= f
```

Res se $\gg=$ izrazi na ustrezen način. Zato sta monadi enaki. □

7.3.1. *Iz teorije v prakso.* Za konec zapišimo monado na množicah iz podpoglavja 3.2 na strani 17 v Haskell sintaksi:

```
instance Monad Dvojice where
    return :: a → (a,a)
    return x  = (x,x)

    ( >>= ) :: (a,a) → (a → (b,b)) → (b,b)
    (x,y) >>= f = (first f(x), second f(y))
```

Izrek o ekvivalentnosti nam zagotavlja, da je to res enaka monada kot (T, η, μ), saj sta `return` in $\gg=$ ustrezno definirana.

V članku [2] si lahko ogledate še, kako se definicije abstraktnih pojmov iz teorije kategorij zapiše v Haskellu. Več o teoriji kategorij si lahko preberete v knjigi [1]. Veliko o monadah lahko izveste v diplomskem delu [3], kjer je predstavljen Beckov izrek. Za matematično motivacijo Haskell monade pa si lahko ogledate članek [4], kjer je podana definicija monade s pomočjo Kleislove trojice.

Literatura

[1] S. Awodey, *Category Theory*, 2nd ed., Oxford logic guides **52**, Oxford University Press, Oxford, 2010.

[2] D. Elkins, *Calculating monads with category theory*, The Monad.Reader **13** (2009) 73–91.

[3] D. Lešnik, *Monade in Beckov izrek*, diplomsko delo, Fakulteta za matematiko in fiziko, Univerza v Ljubljani, 2005.

[4] E. Moggi, *Notions of computation and monads*, Inform. and Comput. **93** (1991) 55–92.

[5] A. Bauer, *Monade*, [ogled 9. 4. 2015], dostopno na `vimeo.com/channels/matfp2014`.

[6] D. Piponi, *You Could Have Invented Monads*, [ogled 1. 8. 2015], dostopno na `blog.sigfpe.com/2006/08/you-could-have-invented-monads-and.html`.

[7] *Category theory*, v: Wikibooks: Open books for open world, [ogled 9. 4. 2015], dostopno na `en.wikibooks.org/wiki/Haskell/Category_theory`.

[8] *Haskell*, [ogled 10. 8. 2015], dostopno na `www.Haskell.org`.