MONADE V FUNKCIJSKEM PROGRAMIRANJU

MITJA ROZMAN

Fakulteta za matematiko in fiziko
Univerza v Ljubljani

Članek predstavi monado, eno pomembnejših struktur v programskem jeziku Haskell. Monada je za programski jezik Haskell bistvena, saj so z njeno pomočjo v Haskellu možne manipulacije z nečistimi izračuni, kot je stanje, na način, ki še vedno zagotavlja čistost ostalih izračunov. Vendar se izkaže, da uporaba monade omogoča še marsikaj drugega. Članek najprej na kratko predstavi osnovne pojme teorije kategorij, kot so kategorija, funktor, naravna transformacija in nazadnje monada. Podana je formalna definicija matematične in Haskllove monade. Na koncu se pokaže še, kako se Haskllova monada izpelje iz matematične, in dokaže njuno ekvivalenco.

MONADS IN FUNCTIONAL PROGRAMMING

A monad is one of the most important structures in the Haskell programming language, due to its ability to manipulate impure computations, for example state, in a way which guarantees purity of other computations. As shown in the article, its use is not limited to just this purpose. The article guides the reader through some basic concepts of category theory such as category, functor, natural transformation, and monad. A formal definition of mathematical and Haskell monad is also presented. At the end it is shown how to derive Haskell monad from mathematical monad, and that they are equivalent.

1. Uvod

V zadnjem času je v Sloveniji in na sploh po svetu zaznati trend naraščanja zanimanja za funkcijsko programiranje. Konkretno na fakultetah FRI in FMF UL je na magistrskem programu v zadnjem času zaznati naklonjenost funkcijskemu programiranju (v jezikih SML, Racket, OCaml, Haskell). K temu pripomore dejstvo, da je tako programiranje ponavadi zelo jasno in so napake v njem, za razliko od imperativnega načina, redkejše. Taka lastnost je zaželena predvsem na zelo občutljivih področjih, kjer si nikakor ne moremo dovoliti napak in vdorov – kot primer navedimo borzo. Zaradi tega se v zadnjem času tudi veliko imperativnih jezikov (Python, C++, Java) odloča za vpeljavo konceptov, ki so klasični v funkcijskem programiranju. V članku bomo pozornost namenili Haskellu, ki je čisti funkcijski jezik (pure functional language). To pomeni, da sprememba stanja ne vpliva na izračun rezultatov, kar je podobno kot pri matematiki, pomembni so le parametri, ki jih funkcija dobi. Izkaže se, da je za delo v takem jeziku zelo uporabna določena struktura, ki se vseskozi pojavlja na različnih mestih. Ustvarjalci so se te strukture zavedali že od vsega začetka, zato so jo vključili med njegove osnovne elemente. Strukturi, ki jo Haskell zelo pogosto uporablja, a mnogim novincem predstavlja veliko težavo pri učenju jezika, rečemo monada. V članku jo formalno matematično definiramo in pokažemo, da je ekvivalentna Haskllovi različici, ki je implementirana v smislu Kleislijeve trojice. Cilj članka je osvetliti nejasnosti na tem področju in predvsem matematični publiki omogočiti lažji prehod v funkcijsko programiranje s Haskellom.

2. Teorija kategorij

Teorija kategorij je matematično področje, na katerem imajo, za razliko od klasičnega pogleda teorije množic, glavno vlogo *morfizmi*, ki so ponavadi preslikave. Namesto proučevanja množic tako proučujemo morfizme, kar nam ponavadi da bolj abstrakten pogled na matematične pojave. Teorija kategorij je v svojih pojmih zelo splošna in zajema mnogo matematičnih področij. Opremi nas s skupnim pogledom in jezikom, s katerim lahko opisujemo značilnosti objektov, ki so si podobni le na zelo abstraktnem nivoju.

2.1 Kategorija

Definicija 1. *Kategorija* $\underline{\mathbf{C}}$ je struktura, ki zadošča naslednjim trem pogojem:

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

2. Za poljubna objekta $A, B \in \mathrm{ob}(\underline{\mathbf{C}})$ obstaja množica, ki jo označimo s $\hom(A, B)$. Elemente te množice označimo z $f, g, \ldots \in \hom(A, B)$. Tem elementom pravimo morfizmi in v splošnem niso preslikave, vseeno pišemo $f : A \to B$.

3. Za poljubne tri objekte $A, B, C \in \mathrm{ob}(\underline{\mathbf{C}})$ obstaja operacija kompozitum $\circ$, ki morfizmoma $f : A \to B$ in $g : B \to C$ priredi morfizem $g \circ f : A \to C$ in ima naslednji lastnosti:

   (a) *Asociativnost*: Za poljubne tri morfizme $f : A \to B$, $g : B \to C$ in $h : C \to D$ velja $h \circ (g \circ f) = (h \circ g) \circ f$.

   (b) *Enota*: Za vsak $B \in \mathrm{ob}(\underline{\mathbf{C}})$ obstaja natanko določen morfizem id_B z lastnostma:

   - Za vsak $f \in \hom(A, B)$ je $id_B \circ f = f$.
   - Za vsak $g \in \hom(B, C)$ je $g \circ id_B = g$.

Kot primer naštejmo tri kategorije z njihovimi standardnimi oznakami: Kategorija $\underline{\mathbf{Set}}$, objekti so množice, morfizmi preslikave med množicami. Kategorija $\underline{\mathbf{Mon}}$, objekti so monoidi, morfizmi so homomorfizmi med monoidi. Kategorija $\underline{\mathbf{Top}}$, objekti so topološki prostori, morfizmi so zvezne preslikave. Pri vseh naštetih primerih je operacija kategorije kar kompozitum preslikav. Asociativnost povsod velja, enota pa je kar identiteta.

Sedaj si oglejmo še tri kategorije, pri katerih morfizmi niso preslikave, oziroma operacija ni kompozitum preslikav:

1. Kategorija $\underline{\mathbf{M}}$, kjer je $\underline{\mathbf{M}} \in \mathrm{ob}(\underline{\mathbf{Mon}})$ neki poljuben monoid. Njegovi objekti so $a, b, \ldots \in \mathrm{ob}(\underline{\mathbf{M}})$. Za te objekte imamo množico morfizmov $\hom(a, b)$, za katere velja $x \in \hom(a, b) \Leftrightarrow a \cdot x = b$. Operacija kompozitum je v tem primeru operacija monoida $\cdot$. Pogoja za asociativnost in enoto sledita iz definicije monoida. Za vsak $b \in \mathrm{ob}(\underline{\mathbf{M}})$ je enota $id_b = e$, kjer je e enota monoida.

2. Kategorija $\underline{\mathbf{C}}$, katere edini objekt je $\mathbb{R}$. Morfizmi naj bodo raztegi, torej funkcije, ki pomnožijo realno število z danim realnim številom. Za morfizme $f, g, \ldots$ 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.

3. Delno urejena množica naravnih števil. Razred objektov je množica naravnih števil $\mathbb{N}_0$. Za objekta $n, m \in \mathbb{N}_0$ je množica morfizmov $\hom(n, m) = \{\leq\}$, če $n \leq m$, sicer prazna. Za poljubne tri objekte n, m, k definiramo operacijo $\circ$, ki morfizmoma $n \leq m$ in $m \leq k$ priredi morfizem $n \leq k$. Zadoščeno je tudi pogoju za asociativnost in enoto. Podobne kategorije bi lahko skonstruirali tudi za ostale refleksivne in tranzitivne relacije.

Ti zgledi pokažejo, da je pojem kategorije zelo širok in zajema mnogo zanimivih matematičnih objektov.

2.2 Funktor

Definicija 2. Naj bosta $\underline{\mathbf{C}}_1$ in $\underline{\mathbf{C}}_2$ kategoriji. Tedaj je $T : \underline{\mathbf{C}}_1 \to \underline{\mathbf{C}}_2$ *funktor*, če velja:

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

2. Funktor priredi vsakemu morfizmu $f : A \to B$ natančno določen morfizem $T(f) : T(A) \to T(B)$. Drugače povedano, iz $f \in \hom(A, B)$ sledi $T(f) \in \hom(T(A), T(B))$.

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 $T(id_A) = id_{T(A)}$.

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

Oglejmo si nekaj primerov funktorjev:

1. Naj bo $\underline{\mathbf{C}}_1$ kategorija, ki za objekte vsebuje naravna števila, med objektoma a, b imamo morfizem $a|b$, če a deli b, sicer nobenega. Naj bo $\underline{\mathbf{C}}_2$ kategorija, katere razred objektov je potenčna množica $\mathcal{P}(\mathbb{N})$. Njeni objekti so podmnožice naravnih števil $A, B, C, \ldots \subseteq \mathbb{N}$, med objektoma A in B imamo morfizem $A \subseteq B$, če je A vsebovana v B, sicer nobenega.

   Primer funktorja med tema kategorijama je funktor $F : \underline{\mathbf{C}}_1 \to \underline{\mathbf{C}}_2$. Funktor priredi vsakemu objektu $a \in \mathrm{ob}(\underline{\mathbf{C}}_1)$ množico vseh deliteljev števila a, ki jo označimo z $\mathrm{del}(a) \in \mathrm{ob}(\underline{\mathbf{C}}_2)$. Funktor priredi vsakemu morfizmu $a|b$ morfizem $\mathrm{del}(a) \subseteq \mathrm{del}(b)$. Res velja, da iz $a|b$ sledi $\mathrm{del}(a) \subseteq \mathrm{del}(b)$. 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 $\mathrm{del}(x) \subseteq \mathrm{del}(y)$.

2. Naj bosta $\underline{\mathbf{M}}_1, \underline{\mathbf{M}}_2 \in \mathrm{ob}(\underline{\mathbf{Mon}})$. Naj bo f homomorfizem med monoidoma $\underline{\mathbf{M}}_1, \underline{\mathbf{M}}_2$. Tedaj je f tudi funktor med kategorijama $\underline{\mathbf{M}}_1, \underline{\mathbf{M}}_2$; f vsak element $a \in \mathrm{ob}(\underline{\mathbf{M}}_1)$ preslika v $f(a) \in \mathrm{ob}(\underline{\mathbf{M}}_2)$. Prav tako vsak morfizem $x \in \hom_{M_1}(a, b)$, za katerega velja $a \cdot x = b$, slika v morfizem $f(x) \in \hom_{M_2}(f(a), f(b))$, za katerega velja $f(a) \cdot f(x) = f(b)$, saj za homomorfizem velja $f(b) = f(a \cdot x) = f(a) \cdot f(x)$. Delovanje funktorja pa je tudi usklajeno z operacijo $\cdot$, saj za morfizma x, y velja $f(x \cdot y) = f(x) \cdot f(y)$ ter za enoto e prvega monoida $f(e_{M_1}) = e_{M_2}$. Na tem mestu smo spet obakrat upoštevali, da je f homomorfizem.

3. Naj bo $\underline{\mathbf{C}}_1$ kategorija, ki za objekte vsebuje množice $\mathbb{Z}, \mathbb{R}, \mathbb{C}, \mathbb{Z}_2$, in $\underline{\mathbf{C}}_2$ kategorija z objekti $\mathbb{Z}^2, \mathbb{R}^2, \mathbb{C}^2, \mathbb{Z}_2^2$, v obeh primerih pa so morfizmi poljubne preslikave oblike $f : A \to B$, kjer sta A in B poljubni množici iz teh kategorij.

   Definiramo lahko funktor $F : \underline{\mathbf{C}}_1 \to \underline{\mathbf{C}}_2$, ki slika objekt A v A^2 in morfizem f v morfizem $Ff = \lambda : (x, y) \mapsto (f(x), f(y))$. Na tem mestu je Ff zapisan z anonimno funkcijo, ki pokaže, kako se Ff uporabi na nekem elementu. Velja namreč $Ff(x, y) = (f(x), f(y))$. Anonimna funkcija je podobna klasičnim funkcijam, le da je ne poimenujemo in je definirana na mestu uporabe. Delovanje funktorja je usklajeno z operacijo $\circ$ na $\underline{\mathbf{C}}_1$ in $\underline{\mathbf{C}}_2$, saj je

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

   podobno pa je izpolnjen tudi pogoj za enoto.

4. Za vsak komutativen kolobar K lahko definiramo množico obrnljivih matrik velikosti $n \times n$ nad tem kolobarjem. Ta množica tvori grupo za množenje, označimo jo z $\mathrm{GL}_n(K)$. Tako lahko definiramo funktor $F : \underline{\mathbf{CRng}} \to \underline{\mathbf{Grp}}$, ki vsakemu kolobarju K priredi grupo $\mathrm{GL}_n(K)$ in vsakemu homomorfizmu kolobarjev $f : K_1 \to K_2$ homomorfizem grup $Ff : \mathrm{GL}_n(K_1) \to \mathrm{GL}_n(K_2)$. Tu je Ff homomorfizem, ki uporabi homomorfizem f na vseh elementih matrike. Iz

tega, da je f homomorfizem kolobarjev, sledi, da je Ff homomorfizem grup. Enakost $Ff(AB) = Ff(A)Ff(B)$ preverimo za poljuben element matrike

$$Ff(AB)_{ij} = f(\sum_{k=1}^{n} A_{ik}B_{kj}) = \sum_{k=1}^{n} f(A_{ik})f(B_{kj}) = \sum_{k=1}^{n} Ff(A)_{ik}Ff(B)_{kj} = (Ff(A)Ff(B))_{ij}.$$

Usklajenost operacije $\circ$ pokažemo na enak način kot pri prejšnjem zgledu.

Trditev 1. *Naj bo $F : \underline{\boldsymbol{C}} \to \underline{\boldsymbol{C}}$ funktor, potem je funktor tudi $F^2 : \underline{\boldsymbol{C}} \to \underline{\boldsymbol{C}}$.*

Dokaz. Naj bo $A \in \mathrm{ob}(\underline{\mathbf{C}})$. Ker je F funktor, je $FA \in \mathrm{ob}(\underline{\mathbf{C}})$ in zato $F(FA) \in \mathrm{ob}(\underline{\mathbf{C}})$. Podoben sklep velja tudi za morfizme. Sedaj si oglejmo še usklajenost operacije kompozitum

$$F(F(f \circ g)) = F(F(f) \circ F(g)) = F^2(f) \circ F^2(g),$$
$$F(F(id_A)) = F(id_{FA}) = id_{F^2A}.$$

Ta trditev velja tudi v splošnejši obliki, kompozitum dveh funktorjev je funktor. Zato obstaja kategorija $\underline{\mathbf{Cat}}$, katere objekti so nekatere kategorije, morfizmi pa funktorji.

2.3 Naravna transformacija

Definicija 3. Imejmo funktorja $F, G : \underline{\mathbf{C}}_1 \to \underline{\mathbf{C}}_2$. *Naravna transformacija* $\alpha : F \to G$ je družina morfizmov $\alpha_A : FA \to GA$ kategorije $\underline{\mathbf{C}}_2$, ki jim rečemo komponente, za katere za vsak morfizem $f : A \to B$ kategorije $\underline{\mathbf{C}}_1$ velja $Gf \circ \alpha_A = \alpha_B \circ Ff$, kar lahko ponazorimo tudi s komutativnim diagramom

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

Oglejmo si primer naravne transformacije za funktor $F : \underline{\mathbf{CRng}} \to \underline{\mathbf{Grp}}$, ki smo ga definirali v prejšnjem podpoglvju in funktor $G : \underline{\mathbf{CRng}} \to \underline{\mathbf{Grp}}$, ki vsak kolobar K preslika v grupo njegovih obrnljivih elementov $G(K) = \widetilde{K} = \{x \in K \mid \exists y. xy = yx = 1\}$, vsak homomorfizem kolobarjev $f : K_1 \to K_2$ pa zoži na homomorfizem grup $G(f) = \widetilde{f} : G(K_1) \to G(K_2)$.

Za tako definirana funktorja je determinanta $\det : F \to G$ naravna transformacija. Res je dobro definirana, saj vsako obrnljivo matriko slika v obrnljiv element kolobarja. Poleg tega je determinanta polinomska funkcija elementov in ker je f homomorfizem kolobarjev, je končni rezultat enak ne glede na smer računanja.

$$\begin{array}{ccc} \mathrm{GL}_n(K_1) & \xrightarrow{\det_{K_1}} & \widetilde{K_1} \\ \downarrow{\scriptstyle Ff} & & \downarrow{\scriptstyle \widetilde{f}} \\ \mathrm{GL}_n(K_2) & \xrightarrow{\det_{K_2}} & \widetilde{K_2} \end{array}$$

2.3.1 Funktor potenčne množice $\mathcal{P}$

Oglejmo si funktor $\mathcal{P} : \underline{\mathbf{Set}} \to \underline{\mathbf{Set}}$. Ta funktor slika objekte po predpisu $\mathcal{P} : A \mapsto \mathcal{P}(A)$, vsaki množici A tako priredi njeno potenčno množico, to je množica vseh podmnožic množice A. Funktor priredi vsaki funkciji $f : A \to B$ funkcijo $\mathcal{P}(f) : \mathcal{P}A \to \mathcal{P}B$. Funkcija $\mathcal{P}(f)$ pa množico $U \in \mathcal{P}(A)$ preslika po predpisu $\mathcal{P}f(U) = f[U] = \{f(x) \mid x \in U\}$. 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 elemente U. Ker je slika množice enaka množici slik vseh njenih elementov, je 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 na primer, 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

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

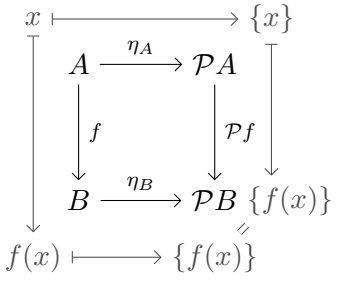
Zadnja ugotovitev je trivialna. Če na elementih množice uporabimo identiteto, se ne spremenijo in imamo še vedno isto množico. Tako smo preverili, da je $\mathcal{P}$ res funktor.

2.3.2 Naravne transformacije funktorjev $\mathrm{Id}, \mathcal{P}, \mathcal{P} \circ \mathcal{P} : \underline{\mathbf{Set}} \to \underline{\mathbf{Set}}$

Poiščimo nekaj naravnih transformacij med funktorji $\mathrm{Id}, \mathcal{P}, \mathcal{P} \circ \mathcal{P}$. Najprej si oglejmo naravne transformacije oblike $\alpha : \mathrm{Id} \to \mathcal{P}$. Taka je recimo $\eta : \mathrm{Id} \to \mathcal{P}$, definirana na komponentah kot

$$\begin{aligned}\eta_A &: A \to \mathcal{P}A, \\ \eta_A &: x \mapsto \{x\}.\end{aligned}$$

Preverimo, da je η naravna transformacija, torej da je izpolnjen pogoj $\mathcal{P}f \circ \eta_A = \eta_B \circ f$, oziroma da pripadajoči diagram komutira. Vzamemo poljuben $x \in A$ in pokažemo, da po obeh poteh dobimo isto

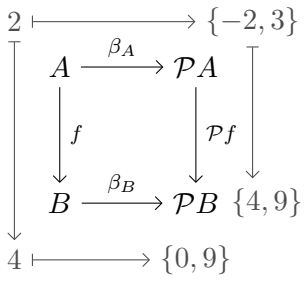


.

Naravna transformacija med tema funktorjema je tudi α, definirana na komponentah kot $\alpha_A : A \to \mathcal{P}A$, s predpisom $\alpha_A : x \mapsto \{\}$ 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 $\beta : \mathrm{Id} \to \mathcal{P}$, definiran na komponentah kot $\beta_A : A \to \mathcal{P}A$, s predpisom $\beta_A : x \mapsto A \setminus \{x\}$. To je naravna transformacija v podkategoriji $\underline{\mathbf{Set}}$, 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



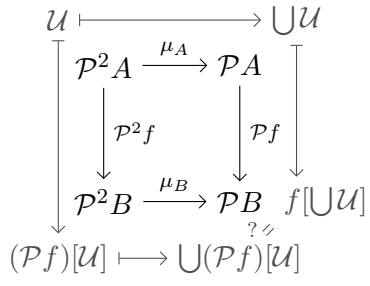
.

Ugotovimo, da diagram ne komutira, saj $\{0, 9\} \neq \{4, 9\}$. Torej β ni naravna transformacija.

Oglejmo si še naravne transformacije oblike $\alpha : \mathcal{P} \circ \mathcal{P} \to \mathcal{P}$. Vzemimo na primer $\mu : \mathcal{P} \circ \mathcal{P} \to \mathcal{P}$, definirano na komponentah kot

$$\mu_A : \mathcal{P}(\mathcal{P}A) \to \mathcal{P}A,$$
$$\mu_A : \mathcal{U} \mapsto \bigcup \mathcal{U}.$$

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 spodnji 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

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

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

Množici vsebujeta iste elemente, s tem smo dokazali, da je μ res naravna transformacija.

2.4 Monada

Definicija 4. *Monada* na kategoriji $\underline{\mathbf{C}}$ je trojica (T, η, μ), kjer je

(1) $T : \underline{\mathbf{C}} \to \underline{\mathbf{C}}$ funktor

(2) $\eta : \mathrm{Id} \to T$ naravna transformacija

(3) $\mu : T \circ T \to T$ naravna transformacija

Pri tem za vsak objekt A velja

(a) enotski zakon: $\mu_A \circ T\eta_A = id_{TA} = \mu_A \circ \eta_{TA}$

(b) zakon asociativnosti: $\mu_A \circ T\mu_A = \mu_A \circ \mu_{TA}$

Razpišimo pogoja, ki veljata za naravni transformaciji za vsak morfizem $f : A \to B$

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

kar lahko ponazorimo tudi s komutativnima diagramoma

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

Ponazorimo sedaj še enotski zakon in zakon asociativnosti

$$\begin{array}{ccccc} TA & \xrightarrow{\eta_{TA}} & T^2A & \xleftarrow{T\eta_A} & TA \\ & {\scriptstyle id_{TA}}\searrow & \downarrow{\scriptstyle \mu_A} & \swarrow{\scriptstyle id_{TA}} & \\ & & TA & & \end{array} \quad , \qquad \begin{array}{ccc} T^3A & \xrightarrow{T\mu_A} & T^2A \\ \downarrow{\scriptstyle \mu_{TA}} & & \downarrow{\scriptstyle \mu_A} \\ T^2A & \xrightarrow{\mu_A} & TA \end{array} \quad .$$

Primer 1. Za primer uporabimo funktor $\mathcal{P}$ ter naravni transformaciji, ki smo ju definirali v prejšnjem razdelku. Monada je trojica $(\mathcal{P}, \eta, \mu)$, kjer je

(1) $\mathcal{P} : \underline{\mathbf{Set}} \to \underline{\mathbf{Set}}$ funktor,

(2) $\eta : \mathrm{Id} \to \mathcal{P}$ naravna transformacija, pri kateri je $\eta_A(x) = \{x\}$,

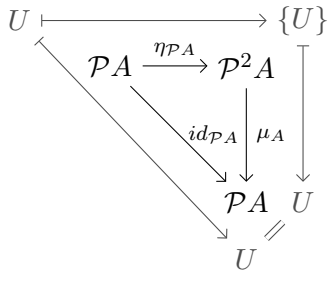
(3) $\mu : \mathcal{P} \circ \mathcal{P} \to \mathcal{P}$ naravna transformacija, pri kateri je $\mu_A(\mathcal{U}) = \bigcup \mathcal{U}$.

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

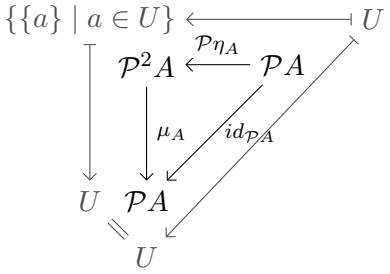
(a) $\mu_A \circ \mathcal{P}\eta_A = id_{\mathcal{P}A} = \mu_A \circ \eta_{\mathcal{P}A}$,

(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 levem diagramu smo tako preverili pogoj $\mu_A \circ \eta_{\mathcal{P}A} = id_{\mathcal{P}A}$, na desnem pa pogoj $\mu_A \circ \mathcal{P}\eta_A = id_{\mathcal{P}A}$



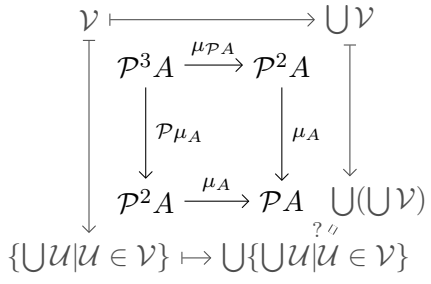
,



.

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

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 množici $\bigcup\{\bigcup\mathcal{U}|\mathcal{U} \in \mathcal{V}\}$ ter $\bigcup(\bigcup\mathcal{V})$, preverimo, če sta enaki

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

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

Ugotovimo, da sta enaki, zato velja $\mu_A \circ \mathcal{P}\mu_A = \mu_A \circ \mu_{\mathcal{P}A}$. S tem je zadoščeno obema dodatnima pogojema. Zato je $(\mathcal{P}, \eta, \mu)$ monada.

3. Haskllova monada

3.1 Motivacija

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 odmislimo 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. Na tem mestu naštejmo le nekaj osnovnih elementov Haskllove sintakse, kot so

komentarji `--`, navajanje tipov z `::`, seznami `[a,b,c,...]`, komponiranje funkcij s piko in uporaba funkcij na elementih brez pisanja oklepajev `f x`. Ostalo sintakso si lahko bralec ogleda na spletu [7]. 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` ("preslikaj")

```
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 v nespremenjenem vrstnem redu. 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. Oglejmo 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 preslikamo dve funkciji vsako posebej ali pa najprej komponiramo funkciji in jih nato preslikamo. 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 preslikamo funkcijo čez dobljeni seznam. Sedaj si oglejmo še primer funkcije, ki jo lahko sprogramiramo sami.

```
kompleksniKoren:: Int → (Double,Double) → [(Double,Double)]
kompleksniKoren m (absVrednost,kot) = resitev
    where
        n = (fromIntegral m)::Double   --pretvori v Double
        r = absVrednost**(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 stopnja korena, ki ga želimo izračunati. Nato kot argument sprejme dvojico, kjer je prva komponenta absulutna vrednost r, druga pa kot φ kompleksnega števila, zapisanega v polarnem zapisu. Funkcija kot rezultat vrne seznam kompleksnih 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 nenegativnega 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 več rezultatov? Ne moremo ju kar komponirati, saj se kodomena ene ne ujema z domeno druge. Lahko pa na rezultat prve preslikamo 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 na njem 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)))
```

3.2 Haskllova kategorija

Za formalno definicijo Haskllove monade moramo seveda vedeti, nad katero kategorijo jo bomo definirali. Na žalost je definicija Haskllove kategorije `Hask` dokaj ohlapna in je definirana kot kategorija, ki za objekte vsebuje vse Haskllove tipe in za morfizme Haskllove funkcije, pri katerih bo še zadoščeno pogojem za kategorijo. Izkaže se namreč, da ne moremo vzeti kar vseh funkcij in tipov, saj v tem primeru struktura ne zadošča pogojem za kategorijo. Več o tem si lahko preberete na spetu [8].

3.3 Definicija Haskllove monade

Ogledali si bomo le poenostavljeno definicijo Haskllove monade, kot je podana tudi na spletu [6]. Celotna definicija vsebuje nekaj dodatnih operacij. Ena od takih je recimo `fail`, ki ponuja obravnavo izjem. Celotno definicijo Haskllove monade si lahko ogledate na spletu [7]. Zreducirana in poenostavljena definicija Haskllove 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 $\gg\!=$ dvočlena operacija, rečemo ji `bind`, 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`: $A \to TA$, 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 Haskllovih 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` $:= \eta$

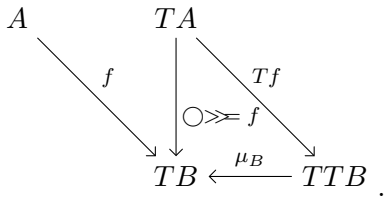
Kar se tiče tipov, je definicija sprejemljiva, saj sta tako `return` kot η tipa $A \to TA$. Operacijo $\gg\!=$ lahko definiramo tudi kot funkcijo

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

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

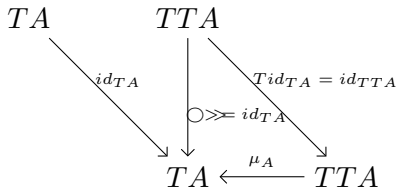
Za $f : A \to TB$ ponazorimo to še z diagramom



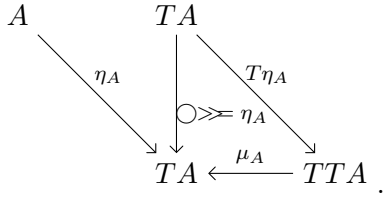
Tu je s krogcem ◯ označeno mesto, kjer $\gg\!= f$ sprejme svoj argument. Velja torej

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

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 B vzamemo A, za f pa 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 Haskllove monade

`w` $\gg\!=$ `return` $=$ `w`.

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

Primer 2. Oglejmo si primer monade `List` v Haskellu, ki smo jo že srečali v motivaciji na strani 8.

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

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

Na tem mestu pokomentirajmo le delovanje funkcije `return`. Funkcija `return` vloži element v seznam. Na številu 8 deluje na primer takole: `return 8 = [8]`.

4. Ekvivalentnost definicij

4.1 Definicija Haskllove monade preko matematične monade

Trditev 2. *Iz matematične monade* (T, η, μ) *lahko definiramo Haskllovo monado s predpisom za* `return` *in* `bind`

$$\texttt{return} := \eta$$
$$w \gg\!= f := \mu(Tf(w)).$$

Dokaz. Trditev bomo dokazali za vsak Haskllov zakon posebej. Na desni strani se sklicujemo na matematične zakone monade, s čimer utemeljimo, zakaj v dani vrstici enakost velja. Najprej dokažimo prvi zakon

`w >>= return = w.`

Izberimo objekt A dane kategorije ter element w tipa TA. Zapišimo levo stran izraza v matematičnem jeziku in ga izračunajmo

$$\begin{aligned} \mu_A(T\eta_A(w)) &= (\mu_A \circ T\eta_A)(w) \\ &= id_{TA}(w) && (\#\text{ enotski zakon}) \\ &= w. \end{aligned}$$

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

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

Imamo $f : A \to TB$. 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

$$\begin{aligned} \mu_B(Tf(\eta_A(x))) &= \mu_B((Tf \circ \eta_A)(x)) \\ &= \mu_B((\eta_{TB} \circ f)(x)) && (\#\ \eta \text{ naravna transformacija}) \\ &= ((\mu_B \circ \eta_{TB}) \circ f)(x) \\ &= (id_{TB} \circ f)(x) && (\#\text{ enotski zakon}) \\ &= f(x). \end{aligned}$$

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 $f : A \to TB$ ter $g : B \to TC$. Izračunamo levo stran

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

Poračunamo še desno stran

$$\begin{aligned}
[D] &= \mu_C((T(\lambda : x \mapsto \mu_C(Tg(f(x)))))(w)) \\
&= \mu_C((T(\mu_C \circ (\lambda : x \mapsto Tg(f(x)))))(w)) \\
&= \mu_C(((T\mu_C) \circ T(\lambda : x \mapsto Tg(f(x))))(w)) && (\#\ \text{lastnost funktorja}) \\
&= ((\mu_C \circ T\mu_C) \circ T(\lambda : x \mapsto Tg(f(x))))(w) \\
&= (\mu_C \circ \mu_{TC}) \circ T(\lambda : x \mapsto Tg(f(x))))(w) && (\#\ \text{zakon asociativnosti}) \\
&= ((\mu_C \circ \mu_{TC}) \circ T(Tg \circ (\lambda : x \mapsto f(x))))(w) \\
&= ((\mu_C \circ \mu_{TC}) \circ T(Tg \circ f))(w) \\
&= ((\mu_C \circ \mu_{TC}) \circ T(Tg) \circ Tf)(w) && (\#\ \text{lastnost funktorja}).
\end{aligned}$$

Leva in desna stran sta enaki, zato velja tudi tretji zakon. S tem smo zadostili vsem zakonom za Haskllovo monado in trditev drži. ∎

4.2 Izražanje T, η, μ s pomočjo `return` in `bind` ter obratno

S funkcijami iz Haskella izrazimo matematične objekte. Naravna transformacija η je kar `return`. Naravno transformacijo μ in funktor T pa izrazimo iz enakosti

$$\texttt{w} \gg\!= \texttt{f} = \mu(Tf(w)).$$

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

$$\texttt{w} \gg\!= \texttt{id\_TA} = \mu_A(T(id_{TA})(w)) = \mu_A(id_{TTA}(w)) = \mu(w).$$

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

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

Tu smo na predzadnjem koraku uporabili enotski zakon. Tako smo s pomočjo matematičnih zakonov izrazili matematične objekte. Sedaj pa s pomočjo Haskllovih zakonov za monado izrazimo Haskllov predpis za `return` in `bind`. Pokazati moramo, da se iz definicij

$\eta_A :=$ `return`
$\mu_A(w) :=$ `join w` $:=$ `w` $\gg\!=$ `id_TA`
$Tf(w) :=$ `fmap f w` $:=$ `w` $\gg\!=$ `return.f`

in Haskllovih zakonov

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

lahko izrazi Haskllove objekte. 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 Haskllovem zapisu ter ga s pomočjo Haskllovih zakonov zapisali v čim krajši obliki.

```
join (map f w) =                                    (# definicija)
(map f w) >>= id =                                  (# definicija)
(w >>= return.f) >>= id =                           (# tretji zakon)
w >>= (λx → ((return.f x) >>= id)) =
w >>= (λx → ((return (f x)) >>= id)) =              (# drugi zakon)
w >>= (λx → (id (f x))) =
w >>= (λx → (f x)) =
w >>= f
```

Res se tudi $\gg\!=$ izrazi na ustrezen način. Tako moramo za ekvivalentnost definicij Haskllove monade in matematične monade preveriti le še, da lahko iz Haskllovih zakonov za monado izpeljemo vse matematične zakone.

4.3 Definicija matematične monade preko Haskllove monade

Trditev 3. *Iz Haskllove monade s predpisom za* `return` *in* `bind` *in pripadajočimi zakoni lahko definiramo matematično monado* (T, η, μ) *z naslednjo definicijo komponent*

$\eta_A :=$ `return`
$\mu_A(w) :=$ `join w := w >>= id_TA`
$Tf(w) :=$ `fmap f w := w >>= return.f .`

Za dokaz potrebujemo še pomožno lemo, katere dokaz je na voljo v diplomskem delu [5], na tem mestu pa jo navedimo brez dokaza.

Lema 4. *Iz Haskllovih zakonov sledi*

- *da je* T *funktor,*
- *da je* η *naravna transformacija,*
- *da je* μ *naravna transformacija.*

Dokaz. Po zadnji lemi je T funktor in η, 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) =                              (# definicija)
join (w >>= return.return) =                        (# definicija)
(w >>= return.return) >>= id =                      (# tretji zakon)
w >>= (λx → (return.return x >>= id)) =
w >>= (λx → (return (return x) >>= id)) =           (# drugi zakon)
w >>= (λx → id (return x)) =
w >>= (λx → return x) =
w >>= return =                                      (# prvi zakon)
w  .
```

Sedaj dokažimo še desno

```
join (return w) =                                  (# definicija)
(return w) >>= id =                                (# drugi zakon)
id w =
w  .
```

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

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

Izračunajmo levo stran

```
join (fmap join n) =                               (# definicija)
join (n >>= return.join) =                         (# definicija)
(n >>= return.join) >>= id =                       (# tretji zakon)
n >>= (λw → (return.join w >>= id)) =
n >>= (λw → (return (join w) >>= id)) =            (# drugi zakon)
n >>= (λw → (id (join w))) =
n >>= (λw → join w) =                              (# definicija)
n >>= (λw → (w >>= id)) .
```

Izračunajmo še desno stran

```
join (join n) =                                    (# definicija)
(join n) >>= id =                                  (# definicija)
(n >>= id) >>= id =                                (# tretji zakon)
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. ∎

4.4 Izrek o ekvivalentnosti

Iz dokazov trditev zadnjih treh podpoglavij sledi naslednji izrek o ekvivalentnosti monad:

Izrek 5 (Izrek o ekvivalentnosti). *Ekvivalentno je, ali podamo monado na matematičen način kot (T, η, μ), ali kot Haskllovo monado s predpisom za* `return` *in* `bind`*. Monadi, ki ju pri tem dobimo, sta enaki.*

Dokaz izreka prepuščam bralcu. Za dokaz enakosti je treba pokazati, da lahko z matematično monado definiramo Haskllovo monado in nato s to monado prvotno matematično monado. Tako pokažemo, da gre res vedno za isto monado.

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] M. Rozman, *Monade v funkcijskem programiranju*, delo diplomskega seminarja, Fakulteta za matematiko in fiziko, Univerza v Ljubljani, 2015.

[6] *Category theory*, v: Wikibooks: Open books for open world, [ogled 7. 7. 2017], dostopno na `en.wikibooks.org/wiki/Haskell/Category_theory`.

[7] *Haskell*, [ogled 7. 7 2017], dostopno na `www.Haskell.org`.

[8] *Wiki Haskell*, [ogled 7. 7 2017], dostopno na `wiki.haskell.org/Hask`.