UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – 1. stopnja

Jure Slak

Induktivni in koinduktivni podatkovni tipi

Delo diplomskega seminarja

Mentor: prof. dr. Andrej Bauer

Ljubljana, 2015

Kazalo

Induktivni in koinduktivni podatkovni tipi

Povzetek

V programskih jezikih se pogosto srečamo z rekurzivnimi podatkovnimi tipi – tipi, ki se v svoji definiciji sklicujejo sami nase. Poseben primer so induktivni in koinduktivni tipi, ki jih obravnavamo v tem delu. Za te tipe razvijemo matematični model s pomočjo začetnih algeber in končnih koalgeber v okviru teorije kategorij. Obravnavo induktivnih tipov kot začetnih algeber predstavimo na primeru naravnih števila in seznamov, medtem ko teorijo koinduktivnih tipov kot končnih koalgeber predstavimo na primeru tokov in konaravnih števil. Podamo tudi primere definicij funkcij za delo z obema vrstama tipov in primere dokazov enostavnih trditev o njih. Na kratko se posvetimo tudi vprašanju eksistence začetnih algeber in končnih koalgeber. Za induktivne podatkovne tipe izpeljemo in formaliziramo principa rekurzije in indukcije, za koinduktivne pa principa korekurzije in koindukcije. Vse te pojme formaliziramo v okviru algeber in koalgeber za funktor ter prejšnje primere definicij funkcij prepišemo s pomočjo principov rekurzije in korekurzije. Podrobneje analiziramo tudi dokaze z indukcijo in koindukcijo ter s pomočjo pojmov bisimulacije in kongruence poudarimo njuno dualnost.

Inductive and Coinductive Data Types

Abstract

In programming languages, we often work with recursive data types – types that refer to themselves in their definitions. We dedicate this work to a notable special case of inductive and coinductive data types. For these types we develop a mathematical model treating them as initial algebras or final coalgebras, making use of category theory. We apply the theoretical view of inductive data types as initial algebras to lists and natural numbers, while the theory of coinductive data types is demonstrated using streams and conatural numbers. Examples of functions for manipulating both kinds of types are given, as well as examples of simple proofs. A question of existence of initial algebras and final coalgebras is also given some attention. We derive and formalize principles of recursion and induction for inductive data types and principles of corecursion and coinduction for coinductive data types. All aforementioned notions are formalised using algebras and coalgebras for a functor, and previous examles of function definitions are rewritten in a recursive or corecursive style. We analyse simple proofs using induction or coinduction and then even futrher emphasize their duality with the notions of bisimulation and congruence.

Math. Subj. Class. (2010): 68Q65

Ključne besede: začetna F-algebra, končna F-koalgebra, induktivni podatkovni tipi, koinduktivni podatkovni tipi, indukcija, koindukcija, rekurzija, korekurzija

Keywords: initial F-algebra, final F-coalgebra, inductive data type, coinductive data type, induction, coinduction, recursion, corecursion

1. Motivacija

Radi bi razvili matematični model enostavnih podatkovnih tipov. Tak matematični model nam koristi v prevajalnikih, pri dokazovanju lastnosti podatkovnih tipov na roke ali v avtomatskih dokazovalnikih. Poleg tega ponuja širši pogled na podatkovne tipe, ki je bolj v uporabi v funkcijskem programiranju. Za začetek si oglejmo motivacijo za študij tipov in neformalen uvod v njihovo obravnavo. Če ni drugače navedeno, bomo snov črpali iz [6].

1.1. **Osnovni tipi.** Najpogostejši in najosnovnejši primeri podatkovnih tipov v programiranju so cela števila `int32`, logične vrednosti `bool` ali pa tip `unit`, ki predstavlja tip z eno samo možno vrednostjo.

$$\begin{aligned}\texttt{int32}&: I = \{-2^{31}, \ldots, -1, 0, 1, \ldots, 2^{31} - 1\}\\ \texttt{bool}&: B = \{\texttt{false}, \texttt{true}\}\\ \texttt{unit}&: U = \{*\}\end{aligned}$$

Vsi ti podatkovni tipi imajo končne množice vrednosti, ki jih lahko zavzamejo, in so z njimi določeni. S sestavljanjem enostavnejših tipov lahko zgradimo veliko različnih in kompleksnejših tipov. Primer takega tipa je `triple`, ki predstavlja trojico števil. Poznamo dva osnovna načina sestavljanja, *produkte* in *vsote* tipov. Produktni tipi so taki, ki hranijo več vrednosti hkrati, na primer `triple`. Množica vrednosti produktnega tipa ustreza natanko kartezičnemu produktu množic vrednosti tipov, iz katerih je zgrajen. Za tip `triple` lahko torej zapišemo:

$$\texttt{triple}: T = I \times I \times I.$$

Kdaj pa si želimo imeti tip, ki lahko zavzame vrednost bodisi enega bodisi drugega tipa. Definirajmo si tip `result`, ki naj predstavlja rezultat nekega celoštevilskega izraza, lahko pa zavzame tudi vrednost $*$, če je šlo pri računanju kaj narobe. Takemu tipu, ki lahko zavzame le eno od naštetih vrednosti, bomo rekli *vsota tipov*. Množica vrednosti ustreza natanko disjunktni uniji množic vrednosti tipov, iz katerih je sestavljen. Tip `result` lahko zavzame katerokoli celoštevilsko vrednost ali vrednost $*$ in njegova množica vrednosti ustreza:

$$\texttt{result}: R = I + U.$$

Tipom, ki so zgrajeni iz osnovnih tipov s pomočjo vsot in produktov, pravimo *algebraični podatkovni tipi*. Pri vsakem algebraičnem tipu z definicijo pravzaprav podamo načine, na katere ga lahko zgradimo. Tip `triple` zgradimo iz treh števil tipa `int32`. Tip `result` zgradimo tako, da mu izberemo vrednost, ki je bodisi tipa `int32`, bodisi tipa `unit`. Tip `int32` pa ustvarimo tako, da mu izberemo eno izmed vrednosti. V tej luči lahko tip `int32` vidimo tudi kot vsoto tipov, ki so podobni `unit`. Vsak člen v vsoti lahko zavzame le eno vrednost, ki predstavlja eno izmed možnih vrednosti tipa `int32`.

$$\texttt{int32}: I = \{-2^{31}\} + \cdots + \{2^{31} - 1\}$$

1.2. **Rekurzivni tipi.** Množice vrednosti vseh tipov, ki smo jih do sedaj definirali, so bile vedno končne, tipi pa precej enostavni. Vendar nam zgolj enostavni tipi ne dajejo dovolj moči in fleksibilnosti; ne moremo namreč definirati nizov znakov, poljubno velikih števil in podobnih, še bolj kompleksnih tipov. Taki tipi so bolj zanimivi, saj jih najlažje definiramo *rekurzivno*, tj. s sklicevanjem samih nase.

Definirajmo rekurzivni tip, ki predstavlja seznam s celoštevilskimi elementi. Definicijo napišemo s pomočjo sklicevanja same nase, namreč: vsak *seznam* je bodisi prazen bodisi je sestavljen iz enega elementa in *seznama*. Če z L označimo množico vseh vrednosti tipa `list`, potem lahko L zapišemo podobno, kot smo zapisali ostale množice vrednosti tipov zgoraj:

$$\texttt{list}\colon L = 1 + I \times L,$$

kjer je 1 enoelementna množica, ki vsebuje prazen seznam. Dobili smo enačbo za L. Če želimo najti množico vrednosti tipa `list`, moramo poiskati rešitev množično-teoretične enačbe $L \cong 1 + I \times L$. Kot je pogosto v matematiki, bomo tudi tukaj objekte obravnavali le "do izomorfizma natančno". Ker delamo v kategoriji množic, to pomeni, da obravnavamo množice, med katerimi obstajajo bijekcije, kot izomorfne. V tem duhu interpretiramo tudi zgornjo enačbo. Iščemo torej tak L, za katerega bo med levo in desno stranjo obstajala bijekcija. Rešitev zgornje enačbe je veliko, mi pa želimo najti najmanjšo rešitev, saj želimo, da tip vsebuje le tisto, kar se iz začetnih vrednosti da skonstruirati, in ničesar drugega. Po konstrukciji pričakujemo, da bo rešitev enaka množici vseh celoštevilskih seznamov, ali ekvivalentno, množici $I^\star$, tj. množici vseh končnih besed nad abecedo I.

$$L = I^\star = \coprod_{n=0}^{\infty} I^n$$

Poglejmo na tip `list` še z drugega zornega kota. Vrednost tipa `list` lahko dobimo po dveh pravilih. Prvo pravi: seznam je lahko prazen. Drugo pravi: če imamo število in seznam, lahko tvorimo nov seznam tako, da za prvi element vzamemo dano število, preostali elementi pa so elementi danega seznama. Prvo pravilo imenujmo nil, drugega pa cons. Pravilom, ki povedo, kako naredimo objekt nekega tipa, pravimo *konstruktorji*.

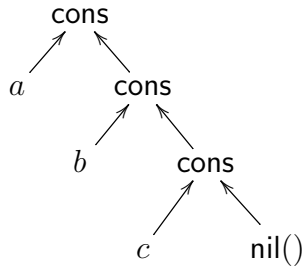
Sedaj lahko začnemo konstruirati množico L s konstruktorji, ki jih imamo. Drugega pravila ne moremo uporabiti, saj še nobenega seznama nimamo, zato pa lahko uporabimo prvega in dobimo prazen seznam $\mathsf{nil}()$. Na njem lahko uporabimo drugi konstruktor in tvorimo nove sezname, ki imajo na prvem mestu različna cela števila. Dobimo, da L vsebuje tudi $\mathsf{cons}(1, \mathsf{nil}())$, $\mathsf{cons}(5, \mathsf{nil}())$ in vse ostale enoelementne sezname. Postopek ponovimo, zopet uporabimo drugi konstruktor na novonastalih seznamih in dobimo na primer seznam $\mathsf{cons}(4, \mathsf{cons}(1, \mathsf{nil}()))$. S ponavljanjem tega postopka lahko dobimo vsak končen celoštevilski seznam, neskončnih pa ne. Za lažje pisanje bomo sezname označevali kar z naštevanjem elementov v oglatih oklepajih, kot je pogosto v navadi, toda zavedamo se, da zapis $[a, b, c]$ v resnici pomeni $\mathsf{cons}(a, \mathsf{cons}(b, \mathsf{cons}(c, \mathsf{nil}())))$. Shematični prikaz uporabe konstruktorjev pri nastajanju seznama $[a, b, c]$ si lahko ogledamo na sliki 1.

Na konstruktorje, ki smo jih definirali kot pravila, bolj matematično gledamo kot na preslikave iz neke množice v množico seznamov L. Oba konstruktorja za sezname lahko tako napišemo kot preslikavi:

$$\mathsf{nil}\colon 1 \to L$$
$$\mathsf{cons}\colon I \times L \to L$$

in ju združimo v eno samo preslikavo ζ, ki slika iz disjunktne unije 1 in $I \times L$.

$$\zeta\colon 1 + I \times L \to L$$



Slika 1. Uporaba konstruktorjev pri nastajanju seznama $[a, b, c]$.

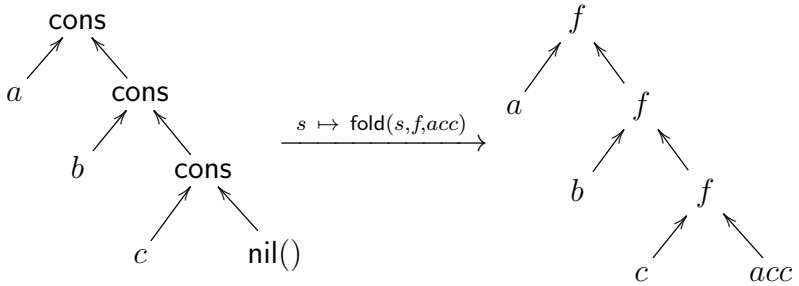
Množica vseh vrednosti tipa `list` je ravno najmanjša negibna točka preslikave ζ, tj. taka množica X, da velja $\zeta(X) \cong X$.

Rekurzivni tipi so lahko zelo zapleteni, kot na primer tip T, ki ustreza enačbi $T = 1 + B^{1+A\times T} \times A + T^A$, za neki množici A in B z vsaj dvema elementoma. Zaradi tega se bomo omejili le na enostavnejše rekurzivne podatkovne tipe, ki jih lahko modeliramo že z množicami, to so *induktivni* in *koinduktivni* podatkovni tipi. Tudi za splošne rekurzivne tipe, kot je tip T zgoraj, obstaja matematični model, ki uporablja topološke prostore in zvezne preslikave. Več o tem lahko bralec prebere v [2, poglavje 7, str. 144].

1.3. **Fold.** Pomembna operacija na seznamih je fold, reduce ali accumulate, kot se ji pogosto pravi v programskih jezikih. To je funkcija, ki sprejme seznam s, funkcijo dveh spremenljivk f in začetno vrednost acc. Funkcija fold zaporedoma uporabi f od desne proti levi na s. Na začetku fold vzame zadnji element seznama in acc ter ju poda funkciji f in si zapomni izračunano vrednost. Na naslednjem koraku vzame predzadnji element in prej izračunano vrednost, ju zopet poda za argumenta f in s tem izračuna novo vrednost. Postopek ponavlja, dokler ne akumulira celega seznama v eno samo vrednost, ki jo tudi vrne kot rezultat. Poglejmo si splošen in konkreten primer:

$$\mathsf{fold}([a, b, c, d], f, acc) = f(a, f(b, f(c, f(d, acc))))$$
$$\mathsf{fold}([1, 2, 3, 4, 5], -, 0) = 1 - (2 - (3 - (4 - (5 - 0)))) = 3.$$

Oglejmo si še diagram operacij, ki jih izvede $\mathsf{fold}([a, b, c], f, acc)$, prikazan na sliki 2. Ta diagram in še marsikaj drugega si lahko bralec podrobneje pogleda v [8]. Iz



Slika 2. Diagram uporabe funkcije $\mathsf{fold}(\,\cdot\,, f, acc)$ na seznamu $[a, b, c]$.

slike 2 vidimo, da fold zamenja konstruktorje s svojimi parametri in ohrani strukturo gnezdenja argumentov. Sledi, da je funkcija $\mathsf{fold}(\,\cdot\,, \mathsf{cons}, \mathsf{nil}())$ identična funkcija

na množici L, namreč, če konstruktorje zamenjamo same s seboj, nismo ničesar spremenili.

V nadaljevanju dela bomo razvili teorijo v ozadju rekurzivnih podatkovnih tipov in si ogledali še več primerov. Izpeljali bomo, kako posplošiti fold na poljuben induktivni tip in kako nam to pomaga pri delu z induktivnimi podatkovnimi tipi, ko jih implementiramo in ko o njih razmišljamo teoretično. Ogledali si bomo, kako lahko na induktivnih podatkovnih tipih formaliziramo princip indukcije in rekurzije. Poleg tega bomo obravnavali tudi dualna pojma koinduktivnih tipov in koindukcije, ki nam pomagata pri konstrukciji in obravnavi neskončnih struktur, kot so neskončni seznami in tokovi.

2. Notacija

Skozi celotno delo bomo vse trditve, izreke in primere, če ni drugače navedeno, interpretirali v kategoriji množic, ki jo bomo označevali s $\mathcal{S}et$. Produkt objektov A in B bomo označevali z $A \times B$, kanonični projekciji na komponenti pa s p_1 in p_2. Koprodukt (v $\mathcal{S}et$ je to kar disjunktna unija) bomo pisali z $A+B$, kanonični injekciji pa bomo označevali z ι_1 in ι_2. Produkt morfizmov f in g bomo označevali z $\langle f, g\rangle$, koprodukt pa z $[f, g]$. Oznake naravno posplošimo tudi na končno mnogo objektov. Z 1 bomo označevali množico z enim elementom, $1 = \{()\}$. Funkcije bomo pogosto pisali brez dodatnih oklepajev, torej bomo namesto $f(x)$ pisali kar $f\,x$. Sintaksa λ-računa nam bo pogosto omogočila enostaven in kompakten zapis anonimnih funkcij. Funkcijo, ki število preslika v njegov dvakratnik, bomo zapisali z $\lambda x.2x$. Pri tem λ označuje začetek anonimne funkcije, kjer naštejemo vse argumente do pike, po piki pa sledi funkcijski predpis. Funkcijo, ki vrne vsoto kvadratov dveh števil, bi napisali kot $\lambda(x, y).x^2 + y^2$, funkcijo, ki vrne funkcijo, ki svoj argument poveča za ena, pa z $\lambda.(\lambda x.x + 1)$.

3. F-algebre in F-koalgebre

V tem razdelku bomo obravnavali matematični model induktivnih in koinduktivnih podatkovnih tipov kot začetnih algeber in končnih koalgeber za funktor.

3.1. Algebre, koalgebre in homomorfizmi.

Pri motivaciji smo videli, da lahko konstruktorje za sezname zapišemo z eno preslikavo $\zeta\colon 1 + I \times L \to L$. Kodomena te preslikave je množica vrednosti našega tipa, domena pa je sestavljena iz vsot in produktov množice L z nekimi drugimi množicami. Domeno lahko zapišemo kot $F(L)$, za funktor $F(X) = 1 + I \times X$, in s to oznako preslikava ζ postane:

$$\zeta\colon F(L) \to L.$$

Situacijo lahko posplošimo na poljuben endofunktor F na poljubni kategoriji, kar nas vodi v definicijo F-algeber.

Definicija 3.1. Naj bo $\mathscr{C}$ kategorija in $F\colon \mathscr{C} \to \mathscr{C}$ endofunktor na $\mathscr{C}$. *F-algebra* je par (C, φ), kjer je C objekt kategorije $\mathscr{C}$ in $\varphi\colon F(C) \to C$ morfizem v kategoriji $\mathscr{C}$. Funktorju F pravimo *signatura* F-algebre, objektu C pa *nosilec*.

Na funktor F gledamo kot na predpis oblike operacij, ki jih mora F-algebra imeti. V izbrani F-algebri (C, φ) morfizem φ predstavlja te konkretne operacije, ki jih lahko izvedemo na elementih množice C. Zato ni presenetljivo, da lahko večino algebraičnih struktur predstavimo kot F-algebre z dodatnimi pogoji, saj so algebraične strukture podane ravno z operacijami na množicah. V tem smislu F-algebre

posplošujejo in zajemajo veliko večino algebraičnih struktur, kot na primer grupe, kolobarje, module, algebre in monoide.

Primer 3.2. Oglejmo si F-algebre za funktor $F(X) = 1 + X + X \times X$ na kategoriji $\mathcal{S}et$. Funktorje bomo ponavadi eksplicitno definirali samo na objektih, definicijo na morfizmih pa bomo zamolčali, saj definicija na objektih naravno podaja tudi definicijo na preslikavah. Zgornji funktor bi preslikavo f preslikal v $F(f) = \mathrm{id}_1 + f + f \times f$, torej f izvedemo na vsaki komponenti v produktih in koproduktih posebej, na morebitnih konstantah pa vzamemo kar identično preslikavo.

Iskane F-algebre za zgornji funktor so pari (C, φ), kjer je C poljubna neprazna množica, φ pa preslikava $\varphi\colon F(C) \to C$. Ker je vsaka preslikava iz disjunktne unije sestavljena iz preslikav, ki povedo, kako se obnaša na vsakem sumandu posebej, lahko pišemo $\varphi = [e, i, m]$, kjer so

$$\begin{aligned} e &\colon 1 \to C, \\ i &\colon C \to C, \\ m &\colon C \times C \to C \end{aligned}$$

poljubne preslikave. Množico C si tako lahko predstavljamo kot množico, na kateri imamo definirane tri operacije: dvojiško operacijo m, eniško operacijo i in nič-člено operacijo e. Na nič-člено operacijo lahko gledamo tudi kot na konstanto, saj vedno slika edini element iz svoje domene v točno določen element. Množica C je torej množica z binarno in unarno operacijo ter odlikovano konstanto. Vse te podatke smo zvedeli samo iz predpisa funktorja F in v tem smislu signatura določa obliko operacij. Če bi med temi operacijami veljale še kakšne zveze, ki bi zagotavljale asociativnost, lastnosti enote in inverza, bi C lahko postala grupa.

Pogledamo pa lahko tudi obratno. Imejmo na primer kolobar K, ki ima dve binarni operaciji (množenje in seštevanje), eno unarno operacijo (nasprotni element) in dve konstanti (enoti za množenje in seštevanje). Na konstanti lahko zopet gledamo kot na nič-člени operaciji. Dane operacije so:

$$\begin{aligned} + &\colon K \times K \to K, \\ \cdot &\colon K \times K \to K, \\ - &\colon K \to K, \\ 0 &\colon 1 \to K, \\ 1 &\colon 1 \to K. \end{aligned}$$

Vse si delijo kodomeno K, zato jih lahko s pomočjo koprodukta združimo skupaj v eno preslikavo $\kappa = [0, 1, -, +, \cdot]$, ki slika iz disjunktne unije njihovih domen v K:

$$\kappa\colon 1 + 1 + K + K \times K + K \times K \to K.$$

Od tod vidimo, da vsak kolobar K, skupaj s svojimi operacijami κ, tvori F-algebro za funktor $F(X) = 1 + 1 + X + X + X \times X + X \times X$.

V splošni F-algebri operacije φ slikajo v nosilec C, kar pomeni, da nam operacija pove, kako iz elementov množice sestaviti nove elemente; s pomočjo φ imamo podane načine, kako sestavimo nove vrednosti iz starih. S programerskega vidika to pomeni, da smo dobili predpise za konstruiranje novih objektov – konstruktorje za naš podatkovni tip.

Dualen pojem k F-algebri je F-koalgebra, ki nam podaja načine, kako dano vrednost razstavimo na eno ali več vrednosti.

Definicija 3.3. Naj bo $\mathscr{C}$ kategorija in $F\colon \mathscr{C} \to \mathscr{C}$ endofunktor na $\mathscr{C}$. *F-koalgebra* je par (P, π), kjer je P objekt kategorije $\mathscr{C}$ in $\pi\colon P \to F(P)$ morfizem v kategoriji $\mathscr{C}$. Funktorju F kot pri F-algebrah pravimo *signatura* F-koalgebre, objektu P pa *nosilec*.

Primer 3.4. Oglejmo si koalgebre za funktor $F(X) = \{0,1\} \times X$. F-koalgebre so pari oblike (S, σ), kjer je σ preslikava iz S v $\{0,1\} \times S$. Za razliko od F-algeber nam operacije tokrat ne podajajo načinov, kako narediti elemente S, temveč, kako jih razstaviti in o njih kaj izvedeti. Za vsak $s \in S$ je

$$\sigma(s) = (b, s'),$$

za enolično določena $b \in \{0,1\}$ in $s' \in S$. Preslikava σ nam torej za vsak element s da neko število b, na katero bomo gledali kot na nekaj, kar smo o s izvedeli ali kot na del s, ki smo ga opazili. Poleg tega nam σ element s preslika tudi v neki nov element s', na katerega bomo gledali kot na naslednje stanje s. Element s' lahko spet preslikamo s σ in pridobimo novo opažanje b' in element s''. S ponavljanjem za vsak element $s \in S$ dobimo neko neskončno zaporedje opažanj iz $\{0,1\}^{\mathbb{N}}$. Zgornja struktura je primer enostavnega sistema, kamor spadajo med drugim tudi deterministični avtomati. Mnogo več o teoriji sistemov in njeni povezavi s koalgebrami si lahko bralec prebere v [5].

Ugotovili smo, da koncept F-algebre posplošuje pojem algebraične strukture, vemo pa tudi, da obstajajo med praktično vsemi algebraičnimi strukturami definirane preslikave, ki spoštujejo to strukturo – homomorfizmi. Med F-algebrami definiramo homomorfizme na naraven način, ki se, kot bomo videli, na znanih konkretnih primerih ujemajo z obstoječimi definicijami.

Definicija 3.5. Naj bosta (C, φ) in (D, ψ) F-algebri za endofunktor F na kategoriji $\mathscr{C}$. *Homomorfizem* f med (C, φ) in (D, ψ) je tak morfizem $f\colon C \to D$ iz $\mathscr{C}$, da velja

$$f \circ \varphi = \psi \circ F(f),$$

oziroma, da komutira naslednji diagram:

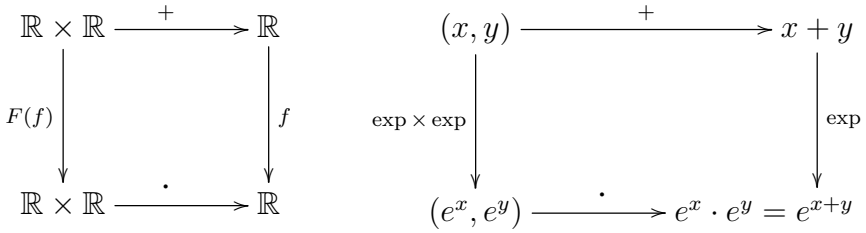
$$\begin{array}{ccc} F(C) & \xrightarrow{\varphi} & C \\ {\scriptstyle F(f)}\downarrow & & \downarrow{\scriptstyle f} \\ F(D) & \xrightarrow{\psi} & D \end{array}$$

Zgornjo enakost razumemo kot običajno definicijo homomorfizma, torej: preslikava je homomorfizem, če je vseeno, ali najprej izvedemo operacijo in potem preslikamo rezultat ali pa najprej preslikamo elemente in potem izvedemo ustrezno operacijo na njihovih slikah.

Opomba 3.6. V definiciji vidimo, da morata imeti oba objekta C in D enako signaturo. To je smiselno, saj tudi običajno gledamo homomorfizme le med objekti istega tipa, torej iz grupe v grupo, ali iz modula v modul, ne pa npr. iz vektorskega prostora v kolobar.

Primer 3.7. Poglejmo si v jeziku F-algeber primer znanega homomorfizma med realnimi števili za seštevanje $(\mathbb{R}, +)$ in realnimi števili za množenje $(\mathbb{R}, \cdot)$. Obe strukturi sta F-algebri za funktor $F(X) = X \times X$. Za homomorfizem f vzemimo

$f(x) = e^x$ in z diagramom preverimo, da ustreza tudi definiciji homomorfizma med F-algebrama.



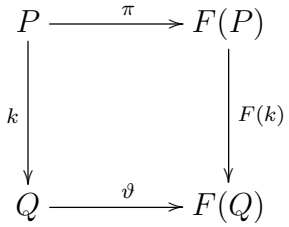
Desni diagram komutira, torej je f res homomorfizem med F-algebrama $(\mathbb{R}, +)$ in $(\mathbb{R}, \cdot)$.

Dualno definiramo homomorfizme med F-koalgebrami.

Definicija 3.8. Naj bosta (P, π) in (Q, ϑ) F-koalgebri. *Homomorfizem* k med (P, π) in (Q, ϑ) je tak morfizem $k\colon P \to Q$ iz $\mathscr{C}$, da velja

$$\vartheta \circ k = F(k) \circ \pi,$$

oziroma, da komutira naslednji diagram:



Primer 3.9. Poglejmo si primer koalgebre, ki se navezuje na analizo. Definirajmo homomorfizem med realno analitičnimi funkcijami in realnimi zaporedji. Obe koalgebraični strukturi bomo definirali za funktor $F(X) = \mathbb{R} \times X$. Množico realno analitičnih funkcij v okolici 0 označimo s $C^\omega(\mathbb{R})$. Opremimo jo z operacijo ρ, ki funkcijo preslika v njeno vrednost v točki 0 in njen odvod.

$$\begin{aligned} &\rho\colon C^\omega(\mathbb{R}) \to \mathbb{R} \times C^\omega(\mathbb{R}) \\ &\rho(f) = (f(0), f') \end{aligned}$$

Podobno na množici $\mathbb{R}^\mathbb{N}$ definiramo operacijo τ, ki zaporedje razstavi na prvi element in z indeksom pomnožen preostanek.

$$\begin{aligned} &\tau\colon \mathbb{R}^\mathbb{N} \to \mathbb{R} \times \mathbb{R}^\mathbb{N} \\ &\tau((a_i)_{i=0}^\infty) = (a_0, (i \cdot a_i)_{i=1}^\infty) \end{aligned}$$

S tema operacijama obe množici postaneta koalgebri za F.

Preslikava T, ki preslika funkcijo f v zaporedje koeficientov v njenem Taylorjevem razvoju v okolici 0:

$$T(f) = \left(\frac{f^{(i)}(0)}{i!}\right)_{i=0}^\infty,$$

je homomorfizem med danima koalgebrama. Narišimo si diagram, ki mu mora ustrezati T:

$$\begin{array}{ccc} f & \xrightarrow{\quad\rho\quad} & (f(0), f') \\ \Big\downarrow{\scriptstyle T} & & \Big\downarrow{\scriptstyle \mathrm{id}_{\mathbb{R}}\times T} \\ \left(\frac{f^{(i)}(0)}{i!}\right)_{i=0}^{\infty} & \xrightarrow{\quad\tau\quad} & \left(f(0), \left(i\frac{f^{(i)}(0)}{i!}\right)_{i=1}^{\infty}\right) \end{array}$$

in preverimo, da komutira. Da želena enakost drži na prvi komponenti, je razvidno neposredno iz definicij, za drugo komponento pa izračunajmo:

$$T(f') = \left(\frac{(f')^{(i)}(0)}{i!}\right)_{i=0}^{\infty} = \left(\frac{f^{(i+1)}(0)}{i!}\right)_{i=0}^{\infty} = \left(\frac{f^{(i)}(0)}{(i-1)!}\right)_{i=1}^{\infty} = \left(i\frac{f^{(i)}(0)}{i!}\right)_{i=1}^{\infty}$$

kar dokončno pokaže, da diagram res komutira in da je T homomorfizem.

3.2. **Kategoriji F-algeber in F-koalgeber.** V prejšnjem razdelku smo definirali F-algebre in homomorfizme med njimi. Pokazali bomo, da F-algebre skupaj s homomorfizmi med njimi tvorijo kategorijo, ki nam bo prišla prav v kasnejših definicijah.

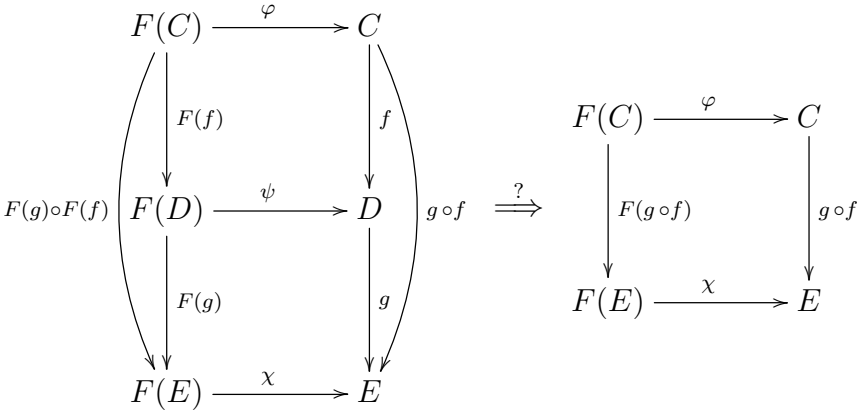
Definicija 3.10. Naj bo dan endofunktor F na kategoriji $\mathscr{C}$. Definiramo *kategorijo F-algeber* $\mathcal{A}lg(F)$ nad $\mathscr{C}$:

objekti: F-algebre, to so pari (C, φ), pri čemer je $\varphi\colon F(C) \to C$,
morfizmi: homomorfizmi med F-algebrami,
kompozitum: enak kot v kategoriji $\mathscr{C}$,
enota: enotski morfizem na (C, φ) je id_C, enotski morfizem na C iz kategorije $\mathscr{C}$.

Trditev 3.11. *Za $\mathcal{A}lg(F)$ veljajo aksiomi kategorije.*

Dokaz. Pokazati je potrebno, da je kompozitum dobro definiran in da obstaja enotski morfizem za vsak objekt (C, φ).

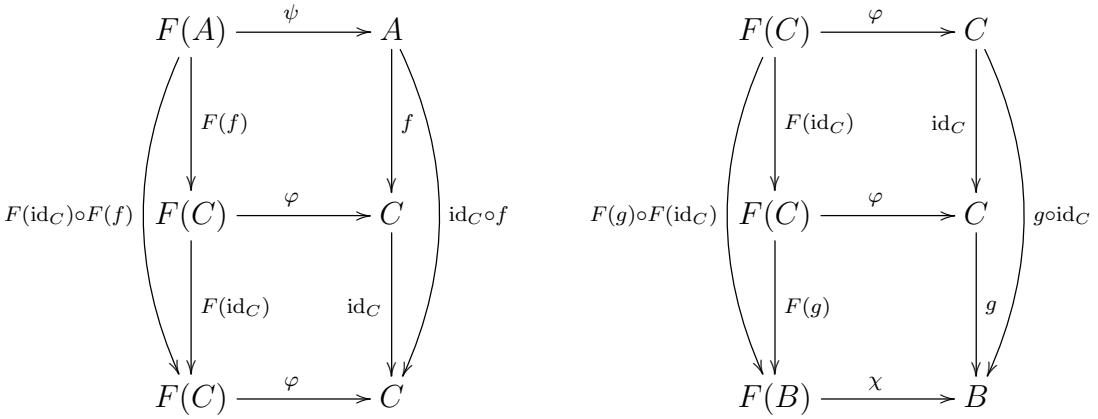
Najprej pokažimo, da je kompozitum dveh morfizmov spet morfizem. Vzemimo tri F-algebre (C, φ), (D, ψ) in (E, χ) ter homomorfizma f in g, kot kaže spodnji levi diagram. Želimo pokazati, da za $g \circ f$ komutira spodnji desni diagram, ki po definiciji pomeni, da je $g \circ f$ homomorfizem med (C, φ) in (E, χ).



Kompozitum $g \circ f$ je res morfizem med objektoma C in E, saj je kompozitum $\circ$ enak kot v $\mathscr{C}$. Da bo to tudi morfizem med F-algebrama (C, φ) in (E, χ) moramo

preveriti, da komutira največji kvadrat levega diagrama. To drži, saj komutirata oba manjša kvadrata. Ker je F funktor, velja še $F(g) \circ F(f) = F(g \circ f)$, torej je $g \circ f$ res homomorfizem med danima F-algebrama in zapisan desni diagram je smiseln in komutira.

Pokažimo še, da je id_C identični morfizem za objekt (C, φ). Poglejmo si diagrama za komponiranje z leve in z desne s poljubnim morfizmom, s katerim je mogoče komponirati.



Po zgoraj pokazanem vemo, da sta $\mathrm{id}_C \circ f$ in $g \circ \mathrm{id}_C$ morfizma. V levem diagramu velja $\mathrm{id}_C \circ f = f$, saj je id_C identični morfizem v $\mathscr{C}$. Z istim argumentom v desnem diagramu velja $g \circ \mathrm{id}_C = g$. Torej je id_C enotski morfizem in $\mathcal{A}lg(f)$ je kategorija. □

Na dualen način definiramo kategorijo F-koalgeber.

Definicija 3.12. Naj bo dan endofunktor F na kategoriji $\mathscr{C}$. Definiramo *kategorijo F-koalgeber* $\mathcal{C}o\mathcal{A}lg(F)$ nad $\mathscr{C}$:

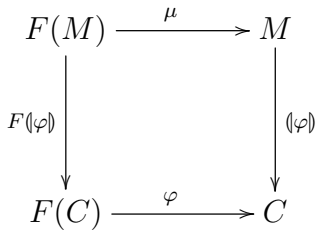
objekti: F-koalgebre, to so pari (P, π), pri čemer je $\pi \colon P \to F(P)$,
morfizmi: homomorfizmi med F-koalgebrami,
kompozitum: enak kot v kategoriji $\mathscr{C}$,
enota: enotski morfizem na (P, π) je id_P, enotski morfizem na P iz kategorije $\mathscr{C}$.

Trditev 3.13. *Za $\mathcal{C}o\mathcal{A}lg(F)$ veljajo aksiomi kategorije.*

Dokaz. Analogno kot dokaz trditve 3.11 (aksiomi za $\mathcal{A}lg(F)$). □

3.3. **Začetne algebre in končne koalgebre.** Začetne algebre in končne koalgebre so pomembne, saj so ravno to objekti, s katerimi bomo modelirali podatkovne tipe.

Definicija 3.14. F-algebra (M, μ) je *začetna F-algebra*, če je začetni objekt v kategoriji $\mathcal{A}lg(F)$. To pomeni, da za vsako F-algebro (C, φ) obstaja natanko en morfizem $(\!|\varphi|\!) \colon M \to C$, tako da komutira naslednji diagram:



oziroma ekvivalentno, da zadošča lastnosti

$$f \circ \mu = \varphi \circ F(f) \iff f = (\!|\varphi|\!).$$

Enoličnemu homomorfizmu $(\!|\varphi|\!)\colon (M, \mu) \to (C, \varphi)$ pravimo *katamorfizem*.

Začetni objekt ne obstaja nujno (glej primer 3.18). Obstoj je zagotovljen, če je funktor F polinomski, glej izrek 5.1.

Dualno k začetnim algebram definiramo končne koalgebre.

Definicija 3.15. F-koalgebra (N, ν) je *končna F-koalgebra*, če je končni objekt v kategoriji $\mathcal{C}o\mathcal{A}lg(F)$. To pomeni, da za vsako F-koalgebro (P, π) obstaja natanko en morfizem $[\![\pi]\!]\colon P \to N$, tako da komutira naslednji diagram:

$$\begin{array}{ccc} P & \xrightarrow{\pi} & F(P) \\ {\scriptstyle [\![\pi]\!]}\downarrow & & \downarrow{\scriptstyle F[\![\pi]\!]} \\ N & \xrightarrow{\nu} & F(N) \end{array}$$

oziroma ekvivalentno, da zadošča lastnosti

$$\nu \circ k = F(k) \circ \pi \iff k = [\![\pi]\!].$$

Enolični homomorfizem $[\![\pi]\!]\colon (P, \pi) \to (N, \nu)$ imenujemo *anamorfizem*.

Kako z začetnimi algebrami in končnimi koalgebrami modeliramo podatkovne tipe, bomo obravnavali kasneje, v razdelku 4. Zaenkrat samo povejmo izrek, ki ga bomo kasneje uporabili pri utemeljevanju našega modela.

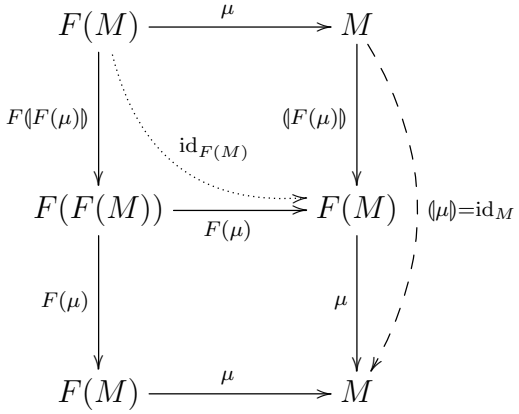
Izrek 3.16 (Lambekova lema)**.** *Naj bo (M, μ) začetna algebra za funktor F. Potem je μ izomorfizem v $\mathcal{S}et$ z inverzom*

$$\mu^{-1} = (\!|F(\mu)|\!).$$

Opomba 3.17. Izrek pove, da je $F(M)$ izomorfen M, torej da je M negibna točka funktorja F, tj. rešitev enačbe $F(X) \cong X$.

Dokaz. Pokazati želimo, da je $\mu \circ (\!|F(\mu)|\!) = \mathrm{id}_M$ in $(\!|F(\mu)|\!) \circ \mu = \mathrm{id}_{F(M)}$.

Oglejmo si diagram, v katerem nastopa $(\!|F(\mu)|\!)$. Ker želimo opazovati kompozitum $(\!|F(\mu)|\!) \circ \mu$, dopišimo še eno vrstico, ki omogoči njegovo obravnavo.



Črtkana puščica je po definiciji enaka $(\!|\mu|\!)$, enoličnemu homomorfizmu iz začetnega objekta (M, μ) v F-algebro (M, μ). Toda morfizem iz M v M je tudi id_M in zaradi

enoličnosti morata biti ta dva morfizma enaka. Po drugi strani je zaradi komutativnosti diagrama ta puščica enaka $\mu \circ (\!|F(\mu)|\!)$ in zopet zaradi enoličnosti morfizma iz začetne algebre velja

$$\mu \circ (\!|F(\mu)|\!) = \mathrm{id}_M,$$

s čimer smo pokazali polovico našega izreka.

Za drugo polovico si oglejmo zgornji kvadrat. Želimo izračunati kompozitum $(\!|F(\mu)|\!) \circ \mu$. Tega direktno ne znamo, zato se odpravimo po drugi strani diagrama, po pikčasti puščici. Izračunamo

$$\begin{aligned}
(\!|F(\mu)|\!) \circ \mu &= F(\mu) \circ F(\!|F(\mu)|\!) && - \text{diagram komutira} \\
&= F(\mu \circ (\!|F(\mu)|\!)) && - F \text{ je funktor} \\
&= F(\mathrm{id}_M) && - \text{po prej dokazanem} \\
&= \mathrm{id}_{F(M)}. && - F \text{ je funktor}
\end{aligned}$$

□

Primer 3.18. Funktor potenčne množice $\mathcal{P}\colon \mathcal{S}et \to \mathcal{S}et$ nima začetne algebre. Obstoj začetne algebre (M, μ) za $\mathcal{P}$ bi po Lambekovi lemi namreč pomenil, da obstaja bijekcija $\mu : \mathcal{P}(M) \to M$ za neko množico M, toda po Cantorjevem izreku vemo, da taka bijekcija ne more obstajati.

Za koalgebre velja dualna verzija Lambekove leme, ki podobno trdi, da je nosilec končne koalgebre negibna točka funktorja.

Izrek 3.19. *Naj bo (N, ν) končna koalgebra za funktor F. Potem je ν izomorfizem v $\mathcal{S}et$ z inverzom*

$$\nu^{-1} = [\![F(\nu)]\!].$$

Dokaz. Dokaz poteka podobno kot dokaz Lambekove leme 3.16, zato ne bomo izdelali vseh podrobnosti. Dokazati želimo, da je $[\![F(\nu)]\!]$ levi in desni inverz ν. Morfizem

$$[\![F(\nu)]\!]\colon F(N) \to N$$

ima pravilno domeno in kodomeno. Za levi inverz izračunamo:

$$[\![F(\nu)]\!] \circ \nu = [\![\nu]\!] = \mathrm{id}_N,$$

za desni inverz pa:

$$\begin{aligned}
\nu \circ [\![F(\nu)]\!] &= F([\![F(\nu)]\!]) \circ F(\nu) && - \text{diagram komutira, } F \text{ je funktor} \\
&= F([\![F(\nu)]\!] \circ \nu) && - \text{po prej dokazanem} \\
&= F(\mathrm{id}_N) && - F \text{ je funktor} \\
&= \mathrm{id}_{F(N)}.
\end{aligned}$$

□

4. Induktivni in koinduktivni tipi

Podatkovne tipe bomo modelirali s pomočjo F-algeber in F-koalgeber. Najprej si poglejmo bolj znane induktivne tipe. Pri takih imamo podane konstruktorje, načine, kako tip zgraditi, zato jih bomo modelirali s F-algebrami.

4.1. **Induktivni tipi.** Pri motivaciji smo videli, kako definiramo tip z množico vrednosti T tako, da podamo vse možne načine, kako konstruirati vrednost tipa. Tipom, za katere je to mogoče, bomo rekli *induktivni podatkovni tipi*. Vsi konstruktorji vsakega induktivnega podatkovnega tipa slikajo v množico T, zato jih lahko združimo v eno funkcijo φ, ki slika iz $F(T)$ v T, za ustrezno izbrani funktor F. Par (T, φ) tako postane F-algebra. Če želimo, da množica T predstavlja množico vrednosti tipa, mora biti zaprta za konstruktorje, vsak njen element pa je možno dobiti z uporabo konstruktorjev na nič ali več drugih elementih. Poleg tega je smiselno predpostaviti, da so konstruktorji injektivni: ob njihovi uporabi z različnimi argumenti namreč želimo dobiti različne rezultate, ne želimo, da med vrednostmi veljajo kakšne dodatne vezi ali enačbe.

Sledi, da za F-algebro (T, φ), ki je model podatkovnega tipa, želimo, da je funkcija φ bijektivna in posledično, da iskana množica T reši enačbo $F(T) \cong T$. Ta enačba ima v splošnem veliko rešitev, toda, ker želimo, da tip zavzame le vrednosti, ki jih lahko dobimo z uporabo konstruktorjev, iščemo najmanjšo množico T, ki reši to enačbo. Po Lambekovi lemi 3.16 vemo, da so začetne algebre negibne točke funktorja, pokazali pa bomo tudi, da so minimalne (trditev 6.8), torej so res primeren model za podatkovne tipe.

4.1.1. *Naravna števila.* Naravna števila[1] $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ definiramo s pomočjo opazke, da je naravno število bodisi enako 0, bodisi je naslednik nekega naravnega števila. S pomočjo tega lahko definiramo dva konstruktorja; enega, ki vrne konstanto 0, in drugega, ki sprejme naravno število in vrne njegovega naslednika.

$$\begin{aligned} &\mathsf{zero}\colon 1 \to \mathbb{N} && \mathsf{succ}\colon \mathbb{N} \to \mathbb{N} \\ &\mathsf{zero}() = 0 && \mathsf{succ}(n) = n + 1 \end{aligned}$$

Združimo funkciji v eno funkcijo $\gamma = [\mathsf{zero}, \mathsf{succ}]\colon 1 + \mathbb{N} \to \mathbb{N}$. Par $(\mathbb{N}, \gamma)$ je F-algebra za funktor $F(X) = 1 + X$. Res, γ namreč slika iz $F(\mathbb{N})$ v $\mathbb{N}$, kar ustreza definiciji F-algebre.

Še več, $(\mathbb{N}, \gamma)$ je začetna F-algebra, kot tudi pričakujemo, saj modelira induktivni podatkovni tip.

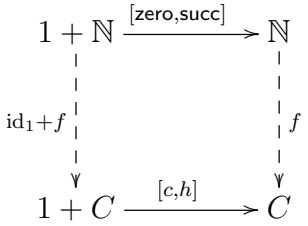
Trditev 4.1. *F-algebra $(\mathbb{N}, \gamma)$ je začetni objekt v kategoriji $\mathcal{A}lg(F)$.*

Dokaz. Pokažimo začetnost po definiciji. Vzemimo poljubno drugo F-algebro (C, φ) in pokažimo, da obstaja natanko en homomorfizem $f\colon (\mathbb{N}, \gamma) \to (C, \varphi)$. Pokažimo najprej enoličnost. Vsaka preslikava iz koprodukta je nastala iz dveh preslikav, torej lahko φ razbijemo na $\varphi = [c, h]$, kjer je c konstanta, h pa neka funkcija na C.

$$\begin{aligned} &c\colon 1 \to C && h\colon C \to C \\ &c() = c_0 \end{aligned}$$

[1] V množico naravnih števil bomo vključili tudi 0, kot je v navadi pri teoriji množic in računalništvu.

Če želimo, da je f homomorfizem, mora diagram



komutirati. Na diagramu zgoraj smo z $\mathrm{id}_1 + f$ označili funkcijo $F(f)$, ki je definirana na vsaki komponenti disjunktne unije posebej. Za vsako množico v disjunktni uniji zgoraj levo v diagramu posebej napišimo pogoj za komutativnost:

$$\begin{aligned} f(\mathsf{zero}()) &= c(\mathrm{id}_1()), \\ \forall n \in \mathbb{N}: \quad f(\mathsf{succ}(n)) &= h(f(n)), \end{aligned}$$

ki ga lahko delno poenostavimo v

$$\begin{aligned} f(0) &= c_0, \\ \forall n \in \mathbb{N}: \quad f(n+1) &= h(f(n)). \end{aligned}$$

Z zaporednim vstavljanjem $n = 0, 1, \ldots$ v drugo enačbo dobimo:

$$\begin{aligned} n = 0: &\qquad f(1) = h(f(0)) = h(c_0) \\ n = 1: &\qquad f(2) = h(f(1)) = h(h(c_0)) \\ &\vdots \\ n = n-1: &\qquad f(n) = h(f(n-1)) = h(h^{n-1}(c_0)) = h^n(c_0). \end{aligned}$$

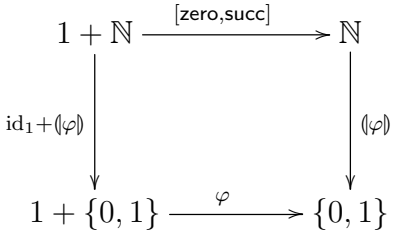
Izračunali smo torej, da če tak f obstaja, mora veljati $f(n) = h^n(c())$. Ker takšna preslikava tudi obstaja in ustreza enačbam za komutativnost, je to naš iskani homomorfizem f. □

Intuitivno bi rešitev našli veliko enostavneje. Poglejmo si na primer število $4 = \mathsf{succ}(\mathsf{succ}(\mathsf{succ}(\mathsf{succ}(\mathsf{zero}()))))$. Homomorfizem bo, podobno kot pri seznamih, le zamenjal konstruktorje z drugimi funkcijami, torej lahko uganemo, da se 4 s f preslika v $f(4) = h(h(h(h(c()))))$. S tem vidimo, da se preslikava, ki f priredi $(\!|f|\!)$, obnaša pri naravnih številih podobno, kot se fold obnaša pri seznamih. Vidimo, da na primeru naravnih števil katamorfizmi res posplošujejo fold na drug podatkovni tip.

Ugotovili smo, da če želimo definirati funkcijo na naravnih številih, moramo samo povedati, kam naj se preslika 0 in kako neko vrednost prevedemo na naslednjo. S tem je funkcija f enolično določena za vsa naravna števila. Rekli bomo, da smo funkcijo f definirali *rekurzivno*. Drugače povedano, za definicijo funkcije na $\mathbb{N}$ je dovolj predpisati, s čim zamenjamo vsakega izmed konstruktorjev.

Primer 4.2. Poglejmo si homomorfizme iz $(\mathbb{N}, \gamma)$ v $(\{0,1\}, \varphi)$ za različne izbire funkcije φ. Vsaka funkcija porodi homomorfizem $(\!|\varphi|\!)$, za katerega spodnji diagram

komutira.



Funkcija $\varphi = [c, h]$ je sestavljena iz konstante c in neke funkcije h na množici $\{0, 1\}$. Za konstanto c vzemimo 0 in obravnavajmo vse štiri možne funkcije h_1, h_2, h_3, h_4 in njihove pripadajoče funkcije φ_1, φ_2, φ_3, φ_4.

(1) $h_1(x) = 1$
Vemo, da $(\!|\varphi_1|\!)$ ravno zamenja konstruktorje, iz katerih je nastalo število, s primernimi komponentami funkcije φ_1, kar s pridom uporabimo v računu.

$$(\!|\varphi_1|\!)(n) = (\!|\varphi_1|\!)(\mathsf{succ}^n(\mathsf{zero}())) = h_1^n(c()) = h_1^n(0) = \begin{cases} 0; & n = 0 \\ 1; & n \geq 1 \end{cases}$$

Dobili smo funkcijo, ki preveri, ali je dano naravno število neničelno.

(2) $h_2(x) = 1 - x$
Računajmo podobno kot pri prejšnjem primeru:

$$(\!|\varphi_2|\!)(n) = (\!|\varphi_2|\!)(\mathsf{succ}^n(\mathsf{zero}())) = h_2^n(c()) = h_2^n(0) = \begin{cases} 0; & n \text{ sod} \\ 1; & n \text{ lih} \end{cases}$$

Zadnja enakost drži, saj h_2^n zaporedoma spreminja 0 v 1 in 1 v 0. Zamenjava se zgodi natanko n-krat, pri čemer smo začeli z 0. Dobili smo funkcijo, ki preveri, ali je dano število liho.

(3) $h_3(x) = 0$, $h_4(x) = x$
Za obe funkciji lahko naredimo enak račun.

$$(\!|\varphi_3|\!)(n) = (\!|\varphi_3|\!)(\mathsf{succ}^n(\mathsf{zero}())) = h_3^n(c()) = h_3^n(0) = 0$$

$$(\!|\varphi_4|\!)(n) = (\!|\varphi_4|\!)(\mathsf{succ}^n(\mathsf{zero}())) = h_4^n(c()) = h_4^n(0) = 0$$

Obakrat dobimo ničelni homomorfizem.

Če bi konstanto c iz 0 spremenili v 1, bi po vrsti dobili: funkcijo, ki vse slika v 1, funkcijo, ki preveri, ali je število sodo, funkcijo, ki preveri, ali je število enako 0, in še enkrat funkcijo, ki vse slika v 1.

Poskusimo sedaj konstruirati operaciji seštevanja in množenja na naravnih številih. Razmišljamo podobno kot pri operaciji fold na seznamih in funkciji definiramo rekurzivno. Predpisati želimo, kaj se naj zgodi z 0 (s čim zamenjamo konstruktor zero) in kako neko vrednost prevesti na naslednjo (s čim zamenjamo konstruktor succ). Inducirani homomorfizem bo tako postal ravno operacija, ki jo želimo. Pri seštevanju števil m in n želimo začeti z n in potem m-krat prišteti 1, pri množenju pa želimo začeti z 0 in m-krat prišteti n.

$$\mathsf{add}(m, n) = (\!|[\lambda x.n, \mathsf{succ}]|\!)(m)$$
$$\mathsf{mul}(m, n) = (\!|[\lambda x.0, \lambda x.\mathsf{add}(x, n)]|\!)(m)$$

Definirajmo še funkcijo predhodnik, ki bo konstruktorju inverzna operacija oziroma *destruktor*. Operacijo definiramo na standarden način, predhodnik števila n

je $n-1$, število 0, ki nima predhodnika, pa preslikamo v $()$. Z lahkoto preverimo, da je ta funkcija res inverzna funkcija funkciji γ.

$$\mathsf{pred}\colon \mathbb{N} \to 1+\mathbb{N}$$
$$\mathsf{pred}(n) = \begin{cases} \iota_1(); & n = 0 \\ \iota_2(n-1); & n \geq 1 \end{cases}$$

Z ι_1 in ι_2 smo označili injekciji v koprodukt, v katerega slika funkcija pred. Sedaj želimo pred definirati rekurzivno. Po Lambekovi lemi vemo, da je to možno in kakšen je predpis zanjo. Vendar pa jo poskusimo skonstruirati, kot da tega ne bi vedeli. Najdimo tak homomorfizem $\varphi = [c, h]$, ki slika v $1+\mathbb{N}$, da bo $(\!|\varphi|\!)$ enak funkciji pred. Funkcija φ opremi $1+\mathbb{N}$ s strukturo F-algebre:

$$\varphi\colon 1+1+\mathbb{N} \to 1+\mathbb{N}$$
$$c\colon 1 \to 1+\mathbb{N}$$
$$h\colon 1+\mathbb{N} \to 1+\mathbb{N}.$$

Iz dokaza trditve 4.1, da je $(\mathbb{N}, \gamma)$ začetna algebra, vemo, da lahko $(\!|\varphi|\!)$ zapišemo kot $(\!|\varphi|\!)(n) = h^n(c())$. Izračunajmo $(\!|\varphi|\!)$ na nekaj prvih naravnih številih.

$$\iota_1() = \mathsf{pred}(0) = (\!|\varphi|\!)(0) = c(),$$

torej mora biti c enak ι_1. Predhodnik 2 je 1 in predhodnik 1 je 0, zato nastavimo

$$\iota_2(0) = \mathsf{pred}(1) = (\!|\varphi|\!)(1) = h(c()) = h(\iota_1()),$$
$$\iota_2(1) = \mathsf{pred}(2) = (\!|\varphi|\!)(2) = h^2(c()) = h(\iota_2(0)).$$

Podobno bi izračunali za ostala števila in dobili enačbe za h:

$$h(\iota_1()) = \iota_2(0)$$
$$h(\iota_2(0)) = \iota_2(1)$$
$$h(\iota_2(1)) = \iota_2(2)$$
$$\vdots$$

Funkcija s takim predpisom kot h zgoraj je natanko konstruktor za naravna števila, komponiran s kanonično injekcijo. Izračunali smo $c = \iota_1$ in $h = \iota_2 \circ \gamma$, torej lahko zapišemo $\varphi = [\iota_1, \iota_2 \circ \gamma] = \mathrm{id}_1 + \gamma$ in velja

$$\mathsf{pred} = (\!|\mathrm{id}_1 + \gamma|\!) = (\!|F(\gamma)|\!),$$

kar je natanko enako predpisu iz Lambekove leme.

4.1.2. *Seznami.* Na sezname rekurzivno pogledamo tako: *seznam* je bodisi prazen bodisi sestavljen iz prvega elementa in *seznama*. To nam podaja dva že znana načina za ustvarjanje seznamov: prvi naredi prazen seznam, drugi pa sprejme seznam in element neke množice A ter vrne seznam s tem elementom na začetku. Označimo z L_A množico vseh seznamov z elementi iz A in zapišimo oba konstruktorja kot preslikavi:

$$\mathsf{nil}\colon 1 \to L_A, \qquad\qquad \mathsf{cons}\colon A \times L_A \to L_A.$$

Podatkovni tip seznamov na neki množici A lahko tako modeliramo kot F_A-algebro za funktor $F_A(X) = 1 + A \times X$. Res, ko konstruktorja s pomočjo disjunktne unije združimo v eno preslikavo $\zeta = [\mathsf{nil}, \mathsf{cons}]$, postane par (L_A, ζ) F_A-algebra. Pri tem

je A prost parameter, ki je lahko poljubna neprazna množica. Matematično korektno definicijo tipov s parametri lahko naredimo s pomočjo bifunktorjev, kot je to pokazano v [6, str. 21].

Trditev 4.3. *Par (L_A, ζ) je začetna F_A-algebra za $F_A(X) = 1 + A \times X$.*

Dokaz. Vemo že, da je to F_A-algebra. Za dokaz začetnosti bomo postopali podobno kot pri naravnih številih. Vzemimo poljubno F_A-algebro (C, φ). Pišimo $\varphi = [c, h]$. Iz uvoda (glej sliko 2) in analogije z naravnimi števili lahko uganemo, da bo homomorfizem dan s preslikavo $s \mapsto \mathsf{fold}(s, h, c)$. Pokažimo, da je to tudi edini možen homomorfizem. Če želimo, da je f homomorfizem, mora komutirati naslednji diagram:

$$\begin{array}{ccc} 1 + A \times L_A & \xrightarrow{[\mathsf{nil},\mathsf{cons}]} & L_A \\ \Big\downarrow{\scriptstyle \mathrm{id}_1 + (\mathrm{id}_A \times f)} & & \Big\downarrow{\scriptstyle f} \\ 1 + A \times C & \xrightarrow{[c,h]} & C \end{array}$$

Na diagramu smo s funktorjem F kot običajno f preslikali v $\mathrm{id}_1 + (\mathrm{id}_A \times f)$. Zapišimo enačbe za komutativnost diagrama za vsako množico v disjunktni uniji.

$$f(\mathsf{nil}()) = c(\mathrm{id}_1()) \tag{list-1}$$

$$f(\mathsf{cons}(a, s)) = h(\mathrm{id}_A(a), f(s)) \tag{list-2}$$

Funkcija f je z zgornjima enačbama enolično določena. Dokažimo to z indukcijo po dolžini seznama. Za prazne sezname to drži neposredno iz enačbe (list-1). Predpostavimo, da to velja za sezname dolžine n in dokazujmo za sezname dolžine $n + 1$. Vzemimo poljuben seznam s dolžine $n + 1$. Gotovo ima vsaj en element in ga lahko zapišemo v obliki $s = \mathsf{cons}(a, s')$ za neka $a \in A$ in $s' \in L_A$. Po enačbi (list-2) velja:

$$f(s) = f(\mathsf{cons}(a, s')) = h(a, f(s')).$$

Desna stran enakosti je po indukcijski predpostavki enolično določena. Torej mora biti f enak $\mathsf{fold}(\,\cdot\,, h, c)$. □

Primer 4.4. Napišimo tri rekurzivno definirane funkcije za delo s seznami. Najprej napišimo funkcijo, ki izračuna dolžino seznama.

$$\mathsf{length}\colon L_A \to \mathbb{N}$$

Razmislimo induktivno: dolžina praznega seznama je enaka 0, dolžina nepraznega pa je za ena večja kot dolžina tega seznama brez prvega elementa. Definiramo torej:

$$\mathsf{length}(s) = (\!|[\mathsf{zero}, \lambda(a, n).\mathsf{succ}(n)]|\!)(s).$$

Zgornjo definicijo lahko ekvivalentno napišemo z uporabo funkcije fold:

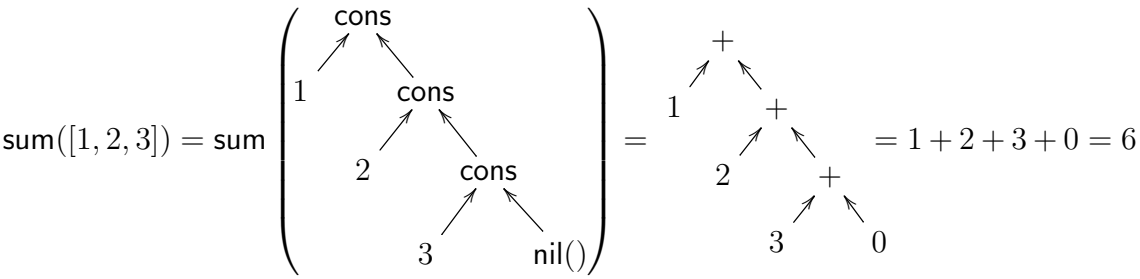
$$\mathsf{length}(s) = \mathsf{fold}(s, \lambda(a, n).\mathsf{succ}(n), 0).$$

Vemo, da fold zamenja konstruktorje s svojimi argumenti; nil smo zamenjali z 0, cons pa s funkcijo "+1". Jasno je, da bo rezultat ravno dolžina seznama. Poglejmo si še primer uporabe te funkcije na seznamu $[a, b, c]$ po pravilih za homomorfizme,

kjer za c in h vstavimo zero in $\lambda(a,n).\mathsf{succ}(n)$.

$$\begin{aligned}
\mathsf{length}([a,b,c]) &= \mathsf{length}(\mathsf{cons}(a,\mathsf{cons}(b,\mathsf{cons}(c,\mathsf{nil}())))) && -\text{ definicija } [a,b,c]\\
&= \mathsf{succ}(\mathsf{length}(\mathsf{cons}(b,\mathsf{cons}(c,\mathsf{nil}())))) && -\text{ uporabimo (list-2)}\\
&= \mathsf{succ}(\mathsf{succ}(\mathsf{length}(\mathsf{cons}(c,\mathsf{nil}())))) && -\text{ uporabimo (list-2)}\\
&= \mathsf{succ}(\mathsf{succ}(\mathsf{succ}(\mathsf{length}(\mathsf{nil}())))) && -\text{ uporabimo (list-2)}\\
&= \mathsf{succ}(\mathsf{succ}(\mathsf{succ}(0))) && -\text{ uporabimo (list-1)}\\
&= 3 && -\text{ konstruktorji za } \mathbb{N}
\end{aligned}$$

Podobno definiramo funkcijo $\mathsf{sum}\colon L_{\mathbb{N}} \to \mathbb{N}$, $\mathsf{sum} = (\!|[\mathsf{zero},\mathsf{add}]|\!)$. Intuitivno je njena pravilnost jasna: k vsoti dodajamo element po element, pri čemer smo začeli z 0. Pravilnost hitro opazimo tudi na primeru iz slike 3, kjer smo za boljšo intuicijo zero nadomestili z 0 in add s $+$.



Slika 3. Diagram uporabe funkcije sum na seznamu $[1,2,3]$.

Napišimo še funkcijo $\mathsf{map}(f)\colon L_A \to L_B$, ki sprejme seznam iz L_A ter uporabi funkcijo $f\colon A \to B$ na vsakem elementu seznama, kar nam da seznam iz L_B.

$$\mathsf{map}(f)([a,b,c]) = [f(a), f(b), f(c)]$$

Funkcijo $\mathsf{map}(f)$ lahko na seznamih definiramo z naslednjim predpisom:

$$\begin{aligned}
\mathsf{map}(f)(s) &= (\!|[\mathsf{nil}, \lambda(a,l).\mathsf{cons}(f(a),l)]|\!)(s)\\
&= (\!|[\mathsf{nil}, \mathsf{cons} \circ (f \times \mathrm{id}_{L_A})]|\!)(s).
\end{aligned}$$

Če bi pri definiciji seznama vzeli enojec 1 za množico A, bi F_A-algebra (L_A, ζ) postala izomorfna naravnim številom. Izomorfizem bi bil dan s funkcijo length iz primera 4.4.

S tem smo obravnavo induktivnih tipov zaključili, čeprav jih obstaja še mnogo, na primer različna drevesa, besede nad abecedo, sintaktični izrazi. Za več primerov glej na primer [6, str. 18]. Dualen koncept k induktivnim tipom so koinduktivni tipi, ki jim bomo posvetili naslednji razdelek.

4.2. **Koinduktivni tipi.** Dostikrat se v funkcijskem programiranju srečamo tudi z neskončnimi podatkovnimi tipi, neskončnimi seznami, drevesi. Takih podatkovnih tipov z začetnimi algebrami ne moremo modelirati, saj ob izvedbi končno mnogo konstruktorjev ne moremo dobiti neskončnega tipa. Zato ni naravno podati načinov, kako jih zgradimo, ampak kako jih razstavimo. Pravimo, da podamo *destruktorje*, tj. načine, kako vrednost našega tipa razstavimo na več vrednosti. Tipom, ki jih definiramo na tak način, pravimo *koinduktivni tipi*.

4.2.1. *Tokovi.* Poglejmo si zgornje ugotovitve na primeru tokov, tj. neskončnih zaporedij elementov iz neke množice A. Podatkovni tip, ki predstavlja tokove, označimo s `stream`. Vsak tok je sestavljen iz prvega elementa, recimo mu glava, in preostanka, recimo mu rep. Preostanek je zopet objekt tipa `stream`. Glavo lahko toku odstranimo poljubno mnogokrat, pa bo rep še vedno ostal tok (ne bo ga zmanjkalo, kot bi na primer seznama), saj je neskončen. Množica vrednosti tipa `stream` je torej tista množica zaporedij, ki se ne spremeni, če vsakemu zaporedju iz te množice z začetka odstranimo en element. Če množico vrednosti tipa `stream` nad množico A označimo s S_A, potem S_A ustreza množično-teoretični enačbi

$$S_A \cong A \times S_A,$$

ki je zgolj matematični zapis zgornjega razmisleka o odstranjevanju elementov. Rešitev te enačbe je tudi $S_A = \emptyset$, vendar to ni rešitev, ki jo iščemo. Množica, ki predstavlja vrednosti tipa `stream`, je največja rešitev take enačbe.

Postavimo zgornja opažanja v korekten matematični okvir. Destruktorja, ki iz toka izluščita glavo ali rep, definiramo kot preslikavi iz množice S_A:

$$\mathsf{head}\colon S_A \to A,$$
$$\mathsf{tail}\colon S_A \to S_A.$$

Funkciji imata enako domeno, zato ju lahko združimo v eno preslikavo, ki slika v kartezični produkt in dani tok razstavi na glavo in rep.

$$\langle \mathsf{head}, \mathsf{tail} \rangle\colon S_A \to A \times S_A$$

Preslikava $\sigma = \langle \mathsf{head}, \mathsf{tail} \rangle$ z množico S_A tvori F_A-koalgebro za funktor $F_A(X) = A \times X$, saj slika iz S_A v $F_A(S_A)$.

S pomočjo teh dveh destruktorjev lahko hitro dobimo posamezne elemente toka. Ničti element dobimo kar s funkcijo head, če pa želimo splošen, n-ti element, moramo samo prej odstraniti prvih n elementov in potem pogledati glavo. Funkcija, ki stori ravno to, je $\mathsf{head} \circ \mathsf{tail}^n$.

Zgornja F_A-koalgebra modelira koinduktivni podatkovni tip `stream`, tako da ni presenečenje, da je to končna F_A-koalgebra. Množica A je podobno kot pri seznamih parameter in je lahko kakršnakoli neprazna množica.

Trditev 4.5. *F_A-koalgebra (S_A, σ) je končni objekt v kategoriji $\mathcal{C}o\mathcal{A}lg(F_A)$.*

Dokaz. Pokažimo končnost po definiciji. Vzemimo poljubno F_A-koalgebro (P, π) in pokažimo, da obstaja natanko en homomorfizem $f\colon (P, \pi) \to (S_A, \sigma)$, predstavljen s črtkano pušico na spodnjem diagramu.

$$\begin{array}{ccc} P & \xrightarrow{\langle c,h \rangle} & A \times P \\ \Big\downarrow{\scriptstyle f} & & \Big\downarrow{\scriptstyle \mathrm{id}_A \times f} \\ S_A & \xrightarrow{\langle \mathsf{head},\mathsf{tail} \rangle} & A \times S_A \end{array}$$

Vsaka preslikava v produkt je podana kot produkt dveh preslikav, zato lahko zapišemo $\pi = \langle c, h \rangle$, kjer je $c\colon P \to A$ in $h\colon P \to P$. Če želimo, da je f homomorfizem, mora zgornji diagram komutirati, torej mora veljati

(stream-1) $$\mathsf{head} \circ f = \mathrm{id}_A \circ c,$$

(stream-2) $$\mathsf{tail} \circ f = f \circ h.$$

Poglejmo, kako izgleda n-ti element slike f. Izračunamo

$$\begin{aligned}
\mathsf{head}(\mathsf{tail}^n(f(p))) &= \mathsf{head}(\mathsf{tail}^{n-1}(f(h(p)))) = \cdots && -\text{ po (stream-2)}\\
&= \mathsf{head}(f(h^n(p))) && -\text{ po (stream-1)}\\
&= \mathrm{id}_A(c(h^n(p)))\\
&= c(h^n(p))
\end{aligned}$$

Celoten tok $f(p)$ je torej enak:

$$f(p) = (c(p), c(h(p)), c(h(h(p))), \ldots).$$

Z zgornjo enakostjo je vsak element slike f določen, torej tak f obstaja. Če želimo, da diagram komutira, smo prisiljeni izbrati izračunano funkcijo, torej je f enolično določena in lahko zapišemo $f = [\![\langle c, h\rangle]\!]$. □

Iz zgornjega dokaza vidimo, da homomorfizmi, podobno kot pri F-algebrah, samo zamenjajo destruktorje dane koalgebre z destruktorji za `stream`. Zaradi tega funkcije, ki slikajo v S_A, lažje definiramo tako, da povemo, kakšni naj bodo destruktorji objektov od koder slikamo, in nato funkcijo definiramo posredno, s pomočjo dejstva, da je (S_A, σ) končna F_A-koalgebra.

Pokažimo to na primeru. Napišimo funkcijo $\mathsf{iterate}$, ki sprejme funkcijo $f\colon A \to A$ in vrne funkcijo, ki za dano začetno vrednost naredi tok zaporedne uporabe f na začetni vrednosti.

$$\begin{aligned}
&\mathsf{iterate}(f)\colon A \to S_A\\
&\mathsf{iterate}(f)(a) = (a, f(a), f(f(a)), \ldots)
\end{aligned}$$

Vidimo, da je glava rezultata enaka a, rep pa je enak, kot če bi poklicali funkcijo $\mathsf{iterate}(f)$ z argumentom $f(a)$. Funkcija $\mathsf{iterate}(f)$ torej ustreza enačbama

$$\begin{aligned}
\mathsf{head} \circ \mathsf{iterate}(f) &= \mathrm{id}_A,\\
\mathsf{tail} \circ \mathsf{iterate}(f) &= \mathsf{iterate}(f) \circ f,
\end{aligned}$$

kar sta ravno enačbi za homomorfizem med koalgebrama $A \xrightarrow{\langle \mathrm{id}_A, f\rangle} A \times A$ in $S_A \xrightarrow{\langle \mathsf{head}, \mathsf{tail}\rangle} A \times S_A$. Zaradi enoličnosti homomorfizma v končno algebro (S_A, σ) je $\mathsf{iterate}(f)$ torej enak $[\![\langle \mathrm{id}_A, f\rangle]\!]$. Če za množico A vzamemo kar naravna števila, potem lahko s pomočjo $\mathsf{iterate}$ ustvarimo tok naravnih števil.

$$[\![\langle \mathrm{id}_{\mathbb{N}}, \mathsf{succ}\rangle]\!](0) = \mathsf{iterate}(\mathsf{succ})(0) = (0, 1, 2, 3, \ldots)$$

4.2.2. *Konaravna števila.* Oglejmo si še končno koalgebro za funktor $F(X) = 1+X$. Vemo, da so začetna algebra za F naravna števila s konstruktorjema zero in succ. V razdelku 4.1.1 smo definirali funkcijo $\mathsf{pred}\colon \mathbb{N} \to 1+\mathbb{N}$, ki je bila inverz konstruktorjev za $\mathbb{N}$. Par $(\mathbb{N}, \mathsf{pred})$ je koalgebra za funktor F, toda ta koalgebra ni končna.

Trditev 4.6. *F-koalgebra $(\mathbb{N}, \mathsf{pred})$ ni končna koalgebra za funktor $F(X) = 1 + X$.*

Dokaz. Zadošča najti neko F-koalgebro (C, φ), da ne obstaja natanko en homomorfizem med (C, φ) in $(\mathbb{N}, \mathsf{pred})$. Našli bomo primer, ko homomorfizma sploh ni, od koder sledi, da $(\mathbb{N}, \mathsf{pred})$ ni končna.

Vzemimo za $C = \mathbb{N}$ in $\varphi = \iota_2$, torej kar vložitev naravnih števil v $F(\mathbb{N})$ in poskusimo najti želeni homomorfizem f. Zanj bi moral komutirati diagram

$$\begin{array}{ccc} \mathbb{N} & \xrightarrow{\iota_2} & 1+\mathbb{N} \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle 1+f} \\ \mathbb{N} & \xrightarrow{\mathsf{pred}} & 1+\mathbb{N} \end{array}$$

Z diagrama preberemo, da mora za f veljati $F(f)(\iota_2(n)) = \mathsf{pred}(f(n))$ za vsak naraven n. Enakost malo poenostavimo, saj je $F(f)$ na drugi komponenti kar f, in dobimo $f(n) = \mathsf{pred}(f(n))$. Slika vsakega naravnega števila s f mora biti tako število, ki je predhodnik samemu sebi. Toda takega naravnega števila ni in $(\mathbb{N}, \mathsf{pred})$ ni končna F-koalgebra. □

Problem iz prejšnje trditve rešimo tako, da dodamo naravnim številom še en element, recimo mu ∞, ki je predhodnik samega sebe. Tako dobimo *konaravna števila* $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$ opremljena s funkcijo predhodnika

$$\begin{aligned} &\mathsf{pred}\colon \mathbb{N}_\infty \to 1 + \mathbb{N}_\infty, \\ &\mathsf{pred}(0) = \iota_1(), \\ &\mathsf{pred}(n) = \iota_2(n-1), \quad n \in \mathbb{N} \setminus \{0\}, \\ &\mathsf{pred}(\infty) = \iota_2(\infty), \end{aligned}$$

ki pa so končna koalgebra. Podajmo samo neformalno utemeljitev tega dejstva, kjer si pomagamo z intuicijo, da homomorfizem samo zamenja destruktorje ene koalgebre z destruktorji druge.

Naj bo (P, π) neka koalgebra za F. Iščemo homomorfizem f med (P, π) in $(\mathbb{N}_\infty, \mathsf{pred})$. Obravnavajmo dve možnosti. Za $p \in P$ se lahko zgodi, da obstaja i, tako da je $\pi^i(p) = \iota_1()$. Tak element p moramo s f preslikati v tisto število $n \in \mathbb{N}_\infty$, da velja $\mathsf{pred}^i(n) = \iota_1()$, torej $n = i - 1$. Lahko pa tak i ne obstaja; v tem primeru moramo postaviti $f(p) = \infty$. S tem je preslikava f določena in če želimo, da bo homomorfizem, smo prisiljeni izbrati to definicijo.

Iskani f iz trditve 4.6, kjer bi namesto naravnih vzeli konaravna števila, bi bil kar konstantna funkcija $f(n) = \infty$, za vsak $n \in \mathbb{N}_\infty$.

5. Eksistenčni izrek

Kot vemo, v splošni kategoriji ni nujno, da začetni objekt obstaja. V kategoriji F-algeber velja enako, kot smo pokazali v primeru 3.18 za funktor potenčne množice. Popolnoma enak argument lahko naredimo tudi za kategorijo F-koalgeber. A kljub temu začetne algebre in končne koalgebre obstajajo za nekatere enostavne funktorje; v razdelku 4 smo nekatere celo eksplicitno konstruirali. Poleg tega iz prakse vemo, da programski jeziki podpirajo kar nekaj induktivnih in koinduktivnih tipov, tako da lahko predvidevamo, da za enostavne funktorje začetne algebre in končne koalgebre obstajajo. To dejstvo potrjuje naslednji izrek, ki ga bomo zgolj navedli kot delni odgovor na vprašanje eksistence, s katero se ne bomo več ukvarjali.

Izrek 5.1. *Naj bo $F\colon \mathcal{S}et \to \mathcal{S}et$ polinomski funktor, tj. oblike*

$$F(X) = \coprod_{i \in I} A_i \times X^i,$$

za neko družino množic $(A_i)_{i\in I}$, indeksirano s končno podmnožico naravnih števil. Potem obstajata začetna algebra v kategoriji F-algeber in končna koalgebra v kategoriji F-koalgeber.

Navedeni izrek je (v večji splošnosti) dokazan v [1, izreka 2.1.9 in 2.3.3]. V dokazu avtorja iskano začetno F-algebro konstruirata s pomočjo limit v okviru teorije kategorij, končno F-koalgebro pa dualno s pomočjo kolimit.

6. Indukcija in koindukcija

Tako pri začetnih kot tudi pri končnih objektih imamo zagotovljen obstoj in enoličnost določenih morfizmov. Videli bomo, da obstoj omogoča rekurzivne in korekurzivne definicije funkcij, medtem ko enoličnost porodi principa indukcije in koindukcije. Vsebino tega razdelka bomo črpali iz [3], pri koindukciji in korekurziji pa so ideje vzete tudi iz [5], kjer se bralec med drugim lahko pouči tudi o širši uporabi koindukcije v teoriji sistemov.

6.1. **Rekurzija.** Tradicionalni koncept rekurzije podaja način, kako definirati funkcije na naravnih številih. Funkcijo lahko rekurzivno definiramo tako, da povemo, kam se slika 0 (začetna vrednost), ter podamo predpis, kako iz prejšnjih vrednosti dobimo naslednje. Pogosto takemu predpisu rečemo tudi induktivna definicija. V splošnem pa je rekurzivna definicija vsaka definicija, ki se sklicuje na objekt sam. Mi bomo obravnavali samo poseben primer rekurzije, ki jo lahko posplošimo na poljuben induktivni podatkovni tip, in sicer *strukturno rekurzijo*. Primer je že videna funkcija length, ki izračuna dolžino seznama:

$$\begin{aligned}\mathsf{length}(\mathsf{nil}()) &= 0\\ \mathsf{length}(\mathsf{cons}(a,s)) &= 1+\mathsf{length}(s).\end{aligned}$$

Kasneje bomo potrebovali še funkcijo, ki seznam podvoji:

$$\begin{aligned}\mathsf{dbl}(\mathsf{nil}()) &= \mathsf{nil}(),\\ \mathsf{dbl}(\mathsf{cons}(a,s)) &= \mathsf{cons}(a,\mathsf{cons}(a,\mathsf{dbl}(s))).\end{aligned}$$

Funkcijo definiramo rekurzivno tako, da predpišemo njene vrednosti na vseh konstruktorjih. S tem smo posredno funkcijo predpisali tudi na vseh vrednostih našega tipa. Natančnejša definicija rekurzivne definicije bi se glasila tako:

Definicija 6.1. Naj bo F endofunktor na $\mathcal{S}et$ in (M,μ) začetna F-algebra. *Princip rekurzije* pomeni, da za vsako F-algebro (C,φ) obstaja enoličen morfizem $(\!|\varphi|\!)\colon M\to C$. Za funkcijo $(\!|\varphi|\!)$, ki slika iz nosilca M začetne algebre, pravimo, da je *definirana rekurzivno*.

Pri rekurzivnih definicijah konstruktorji na levi strani nastopajo znotraj funkcije, ki jo definiramo. Definicija funkcije length je podana z dvema enačbama, ena direktno poda vrednost funkcije, druga pa vsebuje rekurzivni sklic, ki se po končno korakih prevede na osnovni primer. Podobno obnašanje zaznamo pri vseh rekurzivno definiranih funkcijah: imajo enega ali več osnovnih nerekurzivnih primerov, ostali rekurzivni primeri pa se po končno korakih prevedejo na osnovne.

Pri obravnavi naravnih števil kot induktivnega podatkovnega tipa smo že podali enostavni rekurzivni definiciji funkcij add in mult ter funkcij, ki preverita, ali je število neničelno ter ali je liho. Vse te funkcije smo lahko definirali kot enolične

morfizme iz F-algebre. Poskusimo to narediti še na enem standardnem primeru rekurzivne definicije: definicije fakultete naravnega števila.

$$\mathsf{factorial}(n) = \begin{cases} 1; & n = 0 \\ n \cdot \mathsf{factorial}(n-1); & \text{sicer} \end{cases}$$

Definicijo lahko prepišemo z uporabo konstruktorjev zero in succ.

$$\mathsf{factorial}(\mathsf{zero}()) = 1$$
$$\mathsf{factorial}(\mathsf{succ}(n)) = \mathsf{succ}(n) \cdot \mathsf{factorial}(n)$$

Oblika definicije je podobna kot za funkcijo length, z manjšo razliko: tokrat funkcija potrebuje poleg prejšnje funkcijske vrednost tudi trenutno vrednost parametra n.

Primer 6.2. Definicije fakultete se ne da tako enostavno prepisati v jezik F-algeber ravno zaradi tega, ker poleg prejšnje funkcijske vrednosti potrebuje tudi vrednost argumenta. Ta problem rešimo tako, da poleg računanja fakultete sproti računamo še argument in na koncu vrnemo samo prvi del kot rezultat. Takemu postopku pravimo *parčkanje* (*angl.* tupling).

Namesto da rekurzivno definiramo preslikavo $\mathsf{factorial}$, raje definirajmo preslikavo $H\colon \mathbb{N} \to \mathbb{N} \times \mathbb{N}$, tako, da velja $H(n) = (\mathsf{factorial}(n), n)$. Sedaj imamo preko prejšnje funkcijske vrednosti H dostop tako do vrednosti funkcije $\mathsf{factorial}$ kot tudi do njenega argumenta. Preslikavo H lahko definiramo kot en sam katamorfizem. Velja namreč

$$H(0) = (1, 0) \text{ in}$$
$$H(n+1) = ((p_2(H(n)) + 1) \cdot p_1(H(n)), p_2(H(n)) + 1),$$

kar pomeni, da lahko H definiramo rekurzivno:

$$H = (\!|[\lambda.(1,0), \lambda(f,n).(\mathsf{mult}(\mathsf{succ}(n), f), \mathsf{succ}(n))]|\!).$$

Želeni rezultat je prva komponenta H, zato definiramo:

$$\mathsf{factorial} = p_1 \circ H.$$

Na podoben način lahko definiramo veliko rekurzivnih funkcij, ki potrebujejo prejšnje funkcijske vrednosti ali vrednosti argumentov.

Pogost je tudi primer, ko želimo, da je naša funkcija odvisna še od nekega zunanjega parametra $a \in A$. Tudi to je mogoče modelirati v okviru F-algeber. Poglejmo si splošen primer take funkcije $f\colon \mathbb{N} \times A \to B$,

$$f(0, a) = c(a),$$
$$f(n+1, a) = h(f(n, a), a),$$

za neki funkciji $c\colon A \to B$ in $h\colon B \times A \to B$. Problem rešimo s postopkom, imenovanim curryiranje (*angl.* currying), po matematiku Haskellu Curryju. Definiramo $\tilde{f}$, ki za vsako naravno število n vrne *funkcijo*, ki parametru a priredi njegovo vrednost. Funkcije f ne definiramo več kot zaporedja vrednosti, temveč definiramo $\tilde{f}$ kot zaporedje funkcij:

$$\tilde{f}\colon \mathbb{N} \to B^A,$$
$$\tilde{f}(0) = c,$$
$$\tilde{f}(n+1) = \lambda a.h(\tilde{f}(n)(a), a).$$

Preslikavo $\tilde{f}$ definiramo rekurzivno kot preslikavo iz začetne F-algebre $(\mathbb{N}, \gamma)$ v F-algebro z nosilcem B^A, za znani funktor $F(X) = X + 1$:

$$\tilde{f} = (\!|[c, \lambda g.(\lambda a.h(g(a), a))]|\!).$$

S pomočjo funkcije $\tilde{f}$, ki smo jo dobili s curryingom, lahko definiramo funkcijo f z enostavno evalvacijo:

$$f(n, a) = \tilde{f}(n)(a).$$

Zgornja enačba povzame tudi bistvo curryiranja da zapišemo funkcijo več argumentov kot zaporedno evalvacijo funkcij z enim samim argumentom.

6.2. **Korekurzija.** Pri rekurzivnih definicijah smo funkcijo definirali na vseh konstruktorjih danega tipa in jo s tem natanko določili. Pri korekurzivnih definicijah imamo na voljo destruktorje, ki naš objekt razstavijo in nam povedo določeno informacijo o njem. To postane posebej prikladno pri delu z neskončnimi objekti. Funkcijo korekurzivno definiramo s tem, da predpišemo njeno opazljivo obnašanje. To storimo tako, da predpišemo vrednosti vseh destruktorjev na vrednostih funkcije. Kot uvodni primer ponovimo definicijo funkcije $\mathsf{iterate}(\mathsf{f})$ iz razdelka 4.2.1.

$$\mathsf{head}(\mathsf{iterate}(f)(a)) = a$$
$$\mathsf{tail}(\mathsf{iterate}(f)(a)) = \mathsf{iterate}(f)(f(a))$$

Za občutek definirajmo še funkcijo even, ki iz danega toka izlušči elemente s sodim indeksom:

$$\mathsf{even}((s(0), s(1), s(2), \ldots)) = (s(0), s(2), s(4), \ldots).$$

Veljati mora, da je glava toka $\mathsf{even}(s)$ enaka glavi s, rep pa mora biti enak toku, ki ga dobimo, če even uporabimo na repu repa s. To lahko napišemo z enačbami

$$\mathsf{head}(\mathsf{even}(s)) = \mathsf{head}(s),$$
$$\mathsf{tail}(\mathsf{even}(s)) = \mathsf{even}(\mathsf{tail}(\mathsf{tail}(s))).$$

Vidimo, da so destruktorji, za razliko od konstruktorjev pri rekurzivnih definicijah, na levi strani enačb zunaj funkcije, ki jo definiramo. Prav tako nimamo nekakšnega osnovnega primera, na katerega bi se prevedli vsi ostali in morda je kdo izmed bralcev celo podvomil v dobro definiranost gornjih funkcij. A ni razloga za skrb, kot bomo videli takoj, ko formalno definiramo princip korekurzije.

Definicija 6.3. Naj bo F endofunktor na $\mathcal{S}et$ in (N, ν) končna F-koalgebra. *Princip korekurzije* pomeni, da za vsako F-koalgebro (P, π) obstaja enoličen morfizem $[\![\pi]\!] \colon P \to N$. Za funkcijo $[\![\pi]\!]$, ki slika v nosilec N končne koalgebre, pravimo, da je *definirana korekurzivno*.

Zgoraj definirani funkciji even in $\mathsf{iterate}(f)$ lahko definiramo kot anamorfizma iz primernih koalgeber, kot smo za slednjo že naredili. Zaradi končnosti (S_A, σ) sta obe dobro definirani. Kasneje bomo potrebovali še funkcijo $\mathsf{merge} \colon S_A \times S_A \to S_A$, ki združi dva tokova v enega samega z zaporednim prepletanjem elementov. Ustrezati mora enačbam

$$\mathsf{head}(\mathsf{merge}(s, t)) = \mathsf{head}(s),$$
$$\mathsf{tail}(\mathsf{merge}(s, t)) = \mathsf{merge}(t, \mathsf{tail}(s)),$$

kar ustreza korekurzivni definiciji

$$\mathsf{merge} = [\![\langle \mathsf{head}, \lambda(s, t).(t, \mathsf{tail}(s))\rangle]\!].$$

6.3. **Indukcija.** Princip indukcije je v matematiki dobro znan način dokazovanja in se zelo pogosto uporablja za dokazovanje trditev, v katerih nastopajo naravna števila. V tem poglavju bomo princip indukcije postavili v formalen okvir s pomočjo F-algeber in ga posplošili na poljuben induktivni podatkovni tip.

Začnimo s primerom dokaza s principom indukcije, ki verjetno ne bo nič presenetljivega.

Primer 6.4. Spomnimo se, da smo v razdelku 6.1 definirali funkcijo length, ki nam vrne dolžino seznama, in funkcijo dbl, ki podvoji seznam. Pokažimo trditev $\mathsf{length}(\mathsf{dbl}(s)) = 2 \cdot \mathsf{length}(s)$ z indukcijo po dolžini seznama. Najprej preverimo, da velja za prazne sezname.

$$\mathsf{length}(\mathsf{dbl}(\mathsf{nil}())) = \mathsf{length}(\mathsf{nil}()) = 0 = 2 \cdot \mathsf{length}(\mathsf{nil}())$$

Nato predpostavimo, da trditev velja za sezname neke dolžine, in dokazujmo za sezname z enim elementom več.

$$\begin{aligned}
\mathsf{length}(\mathsf{dbl}(\mathsf{cons}(a, s))) &= \mathsf{length}(\mathsf{cons}(a, \mathsf{cons}(a, \mathsf{dbl}(s)))) \\
&= 1 + \mathsf{length}(\mathsf{cons}(a, \mathsf{dbl}(s))) \\
&= 2 + \mathsf{length}(\mathsf{dbl}(s)) \overset{\text{IP}}{=} 2 + 2 \cdot \mathsf{length}(s) \\
&= 2 \cdot (1 + \mathsf{length}(s)) = 2 \cdot \mathsf{length}(\mathsf{cons}(a, s))
\end{aligned}$$

Na koraku, označenem z IP, smo uporabili indukcijsko predpostavko. S tem je trditev dokazana za vse sezname.

Prepišimo zgornji dokaz z indukcijo s pomočjo algeber in morfizmov.

Primer 6.5. Definirajmo preslikavi, ki ustrezata levi in desni strani enakosti, ki smo jo dokazovali, in pokažimo, da sta obe homomorfizma iz začetne algebre. Od tod bo po začetnosti sledila želena enakost. Obe preslikavi slikata iz množice vseh seznamov nad neko neprazno množico A v naravna števila.

$$\begin{aligned}
f &\colon L_A \to \mathbb{N} & f(s) &= \mathsf{length}(\mathsf{dbl}(s)) \\
g &\colon L_A \to \mathbb{N} & g(s) &= 2 \cdot \mathsf{length}(s)
\end{aligned}$$

Opremimo množici L_A in $\mathbb{N}$ s strukturo algebre. Nosilec L_A opremimo z že znano preslikavo $\zeta = [\mathsf{nil}, \mathsf{cons}]$, $\mathbb{N}$ pa opremimo s preslikavo $\xi = [\mathsf{zero}, \mathsf{succ} \circ \mathsf{succ} \circ p_2]$, ki porodi strukturo sodih števil, kar je tudi smiselna izbira, saj so le soda števila slike preslikav f in g. S tem (L_A, ζ) in $(\mathbb{N}, \xi)$ postaneta algebri za funktor $F_A(X) = 1 + A \times X$. Pokažimo, da sta f in g homomorfizma. Preveriti želimo, da komutira naslednji diagram:

$$\begin{array}{ccc}
1 + A \times L_A & \xrightarrow{[\mathsf{nil},\mathsf{cons}]} & L_A \\
\Big\downarrow{\scriptstyle \mathrm{id}_1 + \mathrm{id}_A \times f,\ \mathrm{id}_1 + \mathrm{id}_A \times g} & & \Big\downarrow{\scriptstyle f,g} \\
1 + A \times \mathbb{N} & \xrightarrow{[\mathsf{zero},\mathsf{succ}^2 \circ p_2]} & \mathbb{N}
\end{array}$$

Preverimo komutativnost najprej za f. Na prvi komponenti imamo

$$f(\mathsf{nil}()) = 0 = \mathsf{zero}(),$$

na drugi pa

$$\begin{aligned} f(\mathsf{cons}(a,s)) &= \mathsf{length}(\mathsf{dbl}(\mathsf{cons}(a,s))) = \mathsf{length}(\mathsf{cons}(a,\mathsf{cons}(a,\mathsf{dbl}(s)))) \\ &= 2 + \mathsf{length}(\mathsf{dbl}(s)) = \mathsf{succ}^2(\mathsf{length}(\mathsf{dbl}(s))) \\ &= \mathsf{succ}^2(p_2((\mathrm{id}_A \times f)(s))). \end{aligned}$$

Opazimo, da so izračuni enaki izračunom pred uporabo indukcijske hipoteze v prejšnjem dokazu. Pred njeno uporabo smo pravzaprav preverili, da je f homomorfizem. Res, indukcijsko hipotezo lahko uporabimo šele, ko $\mathsf{length}(f)$ nastopa samostojno, to pa je takrat, ko smo že prišli na drugo stran diagrama, torej je moral komutirati.

Preverimo še komutativnost za g. Najprej izračunamo

$$g(\mathsf{nil}()) = 2 \cdot 0 = \mathsf{zero}(),$$

kar pokaže komutativnost na prvi komponenti. Nadaljujemo z drugo:

$$\begin{aligned} g(\mathsf{cons}(a,s)) &= 2 \cdot \mathsf{length}(\mathsf{cons}(a,s)) = 2 \cdot (1 + \mathsf{length}(s)) = 2 + 2 \cdot \mathsf{length}(s) \\ &= \mathsf{succ}^2(2 \cdot \mathsf{length}(s)) = \mathsf{succ}^2(p_2((\mathrm{id}_A \times g)(s))). \end{aligned}$$

Zopet opazimo, da smo ponavljali popolnoma enake korake kot pri prejšnjem dokazu po uporabi indukcijske hipoteze. Klasičen dokaz z indukcijo v svojem bistvu dokaže, da sta tako leva kot desna stran homomorfizma, iz lastnosti seznamov, ki so začetna algebra, pa sledi enakost.

Na poti do splošnega principa indukcije za začetne algebre si bomo pomagali z njihovo minimalnostjo.

Definicija 6.6. *Podalgebra* F-algebre (C,φ) je monomorfizem $i\colon (A,\alpha) \hookrightarrow (C,\varphi)$. Pogosto A identificiramo kar z njeno sliko v C in jo obravnavamo kot podmnožico C.

Definicija 6.7. F-algebra je *minimalna*, če nima nobene prave podalgebre. To pomeni, da je za vsako podalgebro $i\colon (A,\alpha) \hookrightarrow (M,\mu)$, algebra (A,α) izomorfna (M,μ).

V splošnem bi princip indukcije neformalno zapisali tako: "*Če iz dejstva, da trditev velja za vse argumente konstruktorjev, sledi, da velja za vrednosti, zgrajene s pomočjo teh konstruktorjev, potem velja za vse vrednosti tega tipa.*"

Drugače povedano, če imamo podmnožico nosilca začetne algebre, ki je zaprta za operacije v smislu, da je njena inkluzija homomorfizem, potem je kar enaka nosilcu začetne algebre. Natančneje točno ta princip formulira in dokaže naslednja trditev.

Trditev 6.8. *Začetne algebre so minimalne.*

Dokaz. Naj bo (M,μ) začetna algebra in $i\colon (A,\alpha) \hookrightarrow (M,\mu)$ njena podalgebra. Ker je (M,μ) začetna, obstaja enolični homomorfizem $!\colon (M,\mu) \to (A,\alpha)$.

$$(A,\alpha) \underset{!}{\overset{i}{\rightleftarrows}} (M,\mu)$$

Kompozitum $i \circ !$ je homomorfizem in slika iz (M,μ) v (M,μ), prav tako pa iz (M,μ) v (M,μ) slika tudi homomorfizem id_M. Ker je (M,μ) začetni objekt, morata biti ta dva morfizma enaka:

$$i \circ ! = \mathrm{id}_M.$$

Ker je id_M surjektiven, je tudi i surjektiven, skupaj s predpostavko je torej bijektiven in je (A,α) izomorfna (M,μ). □

Opomba 6.9. Trditev velja v večji splošnosti v vsaki kategoriji: monomorfizem, ki slika v začetni objekt, je nujno izomorfizem.

Zgornja trditev na vsakem induktivnem tipu porodi princip indukcije. Kot se spodobi, na naravnih številih princip, ki ga porodi zgornja trditev, sovpada s klasičnim principom indukcije.

Primer 6.10. Naravna števila $\mathbb{N}$ skupaj s preslikavo $\gamma = [\mathsf{zero}, \mathsf{succ}]$ so začetna F-algebra za funktor $F(X) = 1 + X$. Vzemimo sedaj neko podalgebro $i\colon (A, \alpha) \hookrightarrow (\mathbb{N}, \gamma)$ in A brez škode za splošnost glejmo kot podmnožico $\mathbb{N}$. Prav tako lahko $F(A)$ gledamo kot podmnožico $F(\mathbb{N})$ in operacije na A kot zožitve preslikave γ:

$$\alpha = \gamma|_{F(A)}\colon F(A) \to A.$$

Množica A je po definiciji podalgebre zaprta za konstruktorje: $\gamma(F(A)) \subseteq A$. Poglejmo si natančneje, kaj sledi iz tega. Zaprtost za zero, drugače zapisano, pomeni $0 \in A$. Zaprtost za succ nam pove, da iz $n \in A$ sledi $n + 1 \in A$. Minimalnost $\mathbb{N}$ nam pove, da je i bijekcija, torej je A kar enaka množici $\mathbb{N}$.

Prevedimo zgornji princip še v bolj domačo obliko. Naj bo φ predikat, ki je enak karakteristični funkciji A. Potem velja

(1) $\varphi(0)$,
(2) $\forall n\colon (\varphi(n) \implies \varphi(n+1))$.

Od tod smo sklepali, da za vsako naravno število n velja $\varphi(n)$. Zgoraj opisani sklep je natanko klasični princip indukcije na naravnih številih.

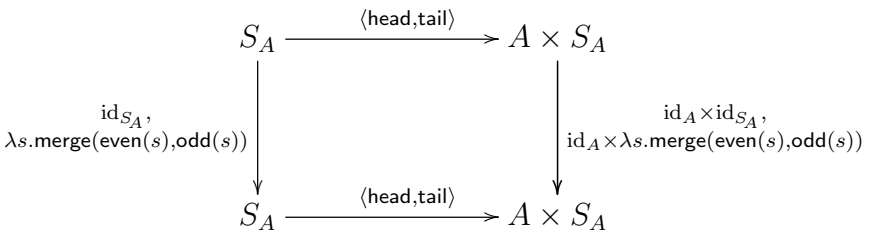
Začetno algebro funktorja $F(X) = 1+X$ bi pravzaprav lahko vzeli kar za definicijo naravnih števil, saj izpolnjuje vse Peanove aksiome. Implicitno smo tudi pokazali, da je ekvivalentno privzeti princip indukcije na naravnih številih, ali pa privzeti, da je F-algebra $(\mathbb{N}, \gamma)$ začetna. Bolj podrobno si lahko bralec ta primer ogleda v [7].

6.4. **Koindukcija.** Dualno k indukciji lahko formaliziramo tudi princip koindukcije, ki nam omogoča dokazovanje trditev v zvezi z neskončnimi objekti. Začnimo s primerom, ki bo po obliki zelo podoben indukciji.

Primer 6.11. Definirali smo že preslikavi even in merge. Sedaj definirajmo še $\mathsf{odd} = \mathsf{even} \circ \mathsf{tail}$, ki vrne tok elementov z lihimi indeksi. Dokažimo trditev:

$$\mathsf{merge}(\mathsf{even}(s), \mathsf{odd}(s)) = s.$$

Preverimo, da sta obe strani enakosti homomorfizma. Ker oba slikata v končno koalgebro, morata biti enaka.



Desna stran enakosti je že endomorfizem na (S_A, σ). Preverimo še za levo, na vsaki komponenti posebej, zgolj z uporabo definicij funkcij merge, even in odd:

$$\begin{aligned}
\mathsf{head}(\mathsf{merge}(\mathsf{even}(s), \mathsf{odd}(s))) &= \mathsf{head}(\mathsf{even}(s)) = \mathsf{head}(s) = \mathrm{id}_A(\mathsf{head}(s)),\\
\mathsf{tail}(\mathsf{merge}(\mathsf{even}(s), \mathsf{odd}(s))) &= \mathsf{merge}(\mathsf{odd}(s), \mathsf{tail}(\mathsf{even}(s)))\\
&= \mathsf{merge}(\mathsf{even}(\mathsf{tail}(s)), \mathsf{even}(\mathsf{tail}(\mathsf{tail}(s))))\\
&= \mathsf{merge}(\mathsf{even}(\mathsf{tail}(s)), \mathsf{odd}(\mathsf{tail}(s))).
\end{aligned}$$

Tudi leva stran enačbe je endomorfizem in iz lastnosti tokov, ki so končna koalgebra, sledi, da sta homomorfizma enaka.

V literaturi, npr. v [4], se pojavlja tudi formulacija zgornjega dokaza, ki zelo spominja na indukcijo. Poteka popolnoma enako kot dokaz zgoraj, le da po tem, ko pokažemo, da se glavi tokov na levi in na desni ujemata:

$$\mathsf{head}(\mathsf{merge}(\mathsf{even}(s), \mathsf{odd}(s))) = \mathsf{head}(\mathsf{even}(s)) = \mathsf{head}(s),$$

privzamemo koindukcijsko hipotezo. Ta pravi, da trditev velja za repe tokov, torej, da za $t = \mathsf{tail}(s)$ velja

$$\mathsf{merge}(\mathsf{even}(t), \mathsf{odd}(t)) = t.$$

Nato dokazujemo, da sta enaka tudi repa obeh strani enakosti.

$$\begin{aligned}
\mathsf{tail}(\mathsf{merge}(\mathsf{even}(s), \mathsf{odd}(s))) &= \mathsf{merge}(\mathsf{odd}(s), \mathsf{tail}(\mathsf{even}(s)))\\
&= \mathsf{merge}(\mathsf{even}(\mathsf{tail}(s)), \mathsf{even}(\mathsf{tail}(\mathsf{tail}(s))))\\
&= \mathsf{merge}(\mathsf{even}(\mathsf{tail}(s)), \mathsf{odd}(\mathsf{tail}(s)))\\
&= \mathsf{tail}(s), \text{po koindukcijski hipotezi.}
\end{aligned}$$

Podobno kot pri indukciji vidimo, da se lahko na koindukcijsko hipotezo skličemo šele, ko $\mathsf{tail}(s)$ nastopa kot argument funkcije, to pa je takrat, ko smo že prišli na drugo stran diagrama in pokazali, da je funkcija homomorfizem.

Splošneje pa se princip koindukcije na koalgebrah zapiše s pomočjo bisimulacije.

Definicija 6.12. Naj bo F endofunktor na $\mathcal{S}et$ in (P, π), (Q, ϑ) F-koalgebri. *Bisimulacija* med (P, π) in (Q, ϑ) je relacija $R \subseteq P \times Q$, za katero obstaja preslikava $\rho\colon R \to F(R)$, da sta kanonični projekciji $p_1\colon R \to P$, $p_2\colon R \to Q$ homomorfizma F-koalgeber. Drugače povedano, obstajati mora ρ, tako da spodnji diagram komutira.

$$\begin{array}{ccccc}
P & \xleftarrow{p_1} & R & \xrightarrow{p_2} & Q\\
\downarrow{\scriptstyle \pi} & & \downarrow{\scriptstyle \rho} & & \downarrow{\scriptstyle \vartheta}\\
F(P) & \xleftarrow{F(p_1)} & F(R) & \xrightarrow{F(p_2)} & F(Q)
\end{array}$$

Bisimulaciji med (P, π) in (P, π) rečemo bisimulacija na (P, π) ali na kratko kar bisimulacija na P.

Sedaj lahko natančno definiramo princip koindukcije za končne koalgebre in dokažemo njegovo pravilnost.

Trditev 6.13 (Princip koindukcije)**.** *Naj bo (N, ν) končna koalgebra. Potem za vsako bisimulacijo R na N in za vsaka $n, n' \in N$ velja: če je*

$$(n, n') \in R,$$

potem velja

$$n = n'.$$

Dokaz. Naj bo (n, n') poljuben par iz relacije R. Po definiciji bisimulacije sta kanonični projekciji $p_1, p_2 \colon R \to N$ homomorfizma. Toda, obe slikata v končno koalgebro (N, ν), torej morata biti enaki. V posebnem velja tudi $p_1(n, n') = p_2(n, n')$, kar se poenostavi v $n = n'$. □

Dokažimo še enkrat, tokrat z uporabo bisimulacije, da drži enakost $\mathsf{merge}(\mathsf{even}(s), \mathsf{odd}(s)) = s$. Pogoje za bisimulacijo na množici tokov S_A prepišimo v bolj domačo obliko.

$$\begin{array}{ccccc} S_A & \xleftarrow{p_1} & R & \xrightarrow{p_2} & S_A \\ \downarrow{\scriptstyle \langle \mathsf{head},\mathsf{tail}\rangle} & & \downarrow{\scriptstyle \rho} & & \downarrow{\scriptstyle \langle \mathsf{head},\mathsf{tail}\rangle} \\ A \times S_A & \xleftarrow{\mathrm{id}_A \times p_1} & A \times R & \xrightarrow{\mathrm{id}_A \times p_2} & A \times S_A \end{array}$$

Iz diagrama preberemo pogoje: relacija R je bisimulacija, če za vsaka $s, t \in S_A$ velja

$$(s,t) \in R \implies \begin{cases} \mathsf{head}(s) = \mathsf{head}(t), & \text{in} \\ (\mathsf{tail}(s), \mathsf{tail}(t)) \in R \end{cases}$$

Definirajmo relacijo $R \subseteq S_A \times S_A$,

$$R = \{(\mathsf{merge}(\mathsf{even}(s), \mathsf{odd}(s)), s); s \in S_A\}$$

in pokažimo, da je bisimulacija. S tem bomo po principu koindukcije pokazali želeno enakost. Naj bo $(\mathsf{merge}(\mathsf{even}(s), \mathsf{odd}(s)), s)$ poljuben element R. Preverimo oba pogoja in opazimo, da s tem dokazujemo, da sta projekciji relacije R na S_A homomorfizma.

$$\mathsf{head}(\mathsf{merge}(\mathsf{even}(s), \mathsf{odd}(s))) = \mathsf{head}(\mathsf{even}(s)) = \mathsf{head}(s)$$

Preverimo še drugi pogoj za bisimulacijo. Če izračunamo tail na obeh komponentah, vidimo, da sta repa v relaciji, saj lahko na prvi komponenti izračunamo

$$\begin{aligned} \mathsf{tail}(\mathsf{merge}(\mathsf{even}(s), \mathsf{odd}(s))) &= \mathsf{merge}(\mathsf{odd}(s), \mathsf{tail}(\mathsf{even}(s))) \\ &= \mathsf{merge}(\mathsf{even}(\mathsf{tail}(s)), \mathsf{even}(\mathsf{tail}(\mathsf{tail}(s)))) \\ &= \mathsf{merge}(\mathsf{even}(\mathsf{tail}(s)), \mathsf{odd}(\mathsf{tail}(s))). \end{aligned}$$

To nam pove, da je

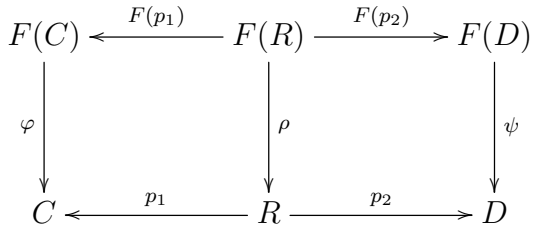
$$\begin{aligned} &(\mathsf{tail}(\mathsf{merge}(\mathsf{even}(s), \mathsf{odd}(s))), \mathsf{tail}(s)) \\ &= (\mathsf{merge}(\mathsf{even}(\mathsf{tail}(s), \mathsf{odd}(\mathsf{tail}(s)))), \mathsf{tail}(s)), \end{aligned}$$

pri čemer je desna stran enakosti po definiciji zopet element iz R. S tem smo dokazali, da je R bisimulacija in po principu koindukcije je trditev dokazana.

Podobna formulacija kot za koindukcijo obstaja tudi za princip indukcije, s pomočjo pojma kongruence.

Definicija 6.14. Naj bo F funktor in (C, φ), (D, ψ) F-algebri. *Kongruenca* med (C, φ) in (D, ψ) je relacija $R \subseteq C \times D$, za katero obstaja F-algebraična struktura $\rho \colon F(R) \to R$, da sta kanonični projekciji $p_1 \colon R \to C$, $p_2 \colon R \to D$ homomorfizma

F-algeber. Zapisano z diagramom: obstajati mora ρ, tako da naslednji diagram komutira v kategoriji $\mathcal{A}lg(F)$.



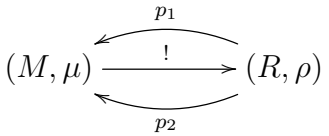
Kongruenci med (C, φ) in (C, φ) rečemo kongruenca na (C, φ) ali kar kongruenca na C.

Za kongruence velja podobna trditev kot za bisimulacije.

Trditev 6.15. *Naj bo (M, μ) začetna algebra. Potem za vsako kongruenco R na M in vsak $m \in M$ velja*

$$(m, m) \in R\,.$$

Dokaz. Ker je (M, μ) začetna, obstaja enolični morfizem $!\colon (M, \mu) \to (R, \rho)$, po definiciji kongruence pa sta homomorfizma tudi obe kanonični projekciji.



Zaradi začetnosti (M, μ) velja

$$p_1 \circ ! = p_2 \circ ! = \mathrm{id}_M.$$

Če zgornjo enakost evalviramo v m, dobimo

$$p_1(!(m)) = p_2(!(m)) = m,$$

torej je $!(m) = (m, m) \in R$. □

Zadnji dve trditvi nam omogočata zelo kratko, sklepno formulacijo indukcije in koindukcije, ki poleg tega še lepo poudari njuno dualnost.

Princip indukcije za začetne algebre (M, μ):

$$\text{za vsako kongruenco } R \subseteq M \times M \text{ velja } \Delta_M \subseteq R.$$

Princip koindukcije za končne koalgebre (N, ν):

$$\text{za vsako bisimulacijo } R \subseteq N \times N \text{ velja } R \subseteq \Delta_N,$$

kjer smo z Δ_X označili relacijo enakosti na X, tj. $\Delta_X = \{(x, x); x \in X\}$.

Literatura

[1] J. Adámek in S. Milius, *Introduction to category theory, algebras and coalgebra*, ESSLLI, 2010, dostopno na `www.tu-braunschweig.de/Medien-DB/iti/survey_full.pdf`.

[2] R. Amadio in P. Curien, *Domains and lambda-calculi*, Cambridge Tracts Theoret. Comput. Sci. **46**, Cambridge University Press, Cambridge, 1998.

[3] B. Jacobs in J. Rutten, *An introduction to (co)algebra and (co)induction*, v: Advanced topics in bisimulation and coinduction (ur. D. Sangiorgi in J. Rutten), Cambridge Tracts Theoret. Comput. Sci. **52**, Cambridge University Press, Cambridge, 2012, str. 38–99, dostopno na `homepages.cwi.nl/~janr/papers/files-of-papers/2011_Jacobs_Rutten_new.pdf`

[4] D. Kozen in A. Silva, *Practical coinduction*, Cornell University, Ithaca, NY, 2012, dostopno na `haslab.uminho.pt/xana/files/structural.pdf`.

[5] J. J. M. M. Rutten *Universal coalgebra: a theory of systems*, Theoret. Comput. Sci. **249**, (2000), 3–80, dostopno na `homepages.cwi.nl/~janr/papers/files-of-papers/universal_coalgebra.pdf`.

[6] V. Vene, *Categorical programming with inductive and coinductive types*, doktorska disertacija, Dissertationes Mathematicae Universitatis Tartuensis **23**, Univerza v Tartuju, Tartu, 2000, dostopno na `kodu.ut.ee/ varmo/papers/thesis.pdf`.

[7] *Initiality and Finality*, v: CLiki for the TUNES project [ogled 19. 2. 2015], dostopno na `tunes.org/wiki/initiality_20and_20finality.html`.

[8] *Haskell/list processing*, v: Wikibooks, open books for an open world, [ogled 19. 2. 2015], dostopno na `en.wikibooks.org/wiki/Haskell/List_processing#foldr`.