DEO IV: TEORIJSKE OSNOVE FUNKCIONALNOG PROGRAMIRANJA

Sadržaj

Poglavlje 11: Lambda račun

Funkcionalno programiranje temelji se na dve oblasti matematike: lambda računu i teoriji kategorija. U nastavku ovog poglavlja izložićemo osnovne pojmove svake od njih i primerima pokazati kako se one koriste u funkcionalnom programiranju.

11.1. O Lambda računu

Američki matematičar Alonzo Čerč (Alonzo Church)[1] razvio je tridesetih godina teoriju Lambda računa (λ-račun) izučavajući matematička svojstva efektivno sračunljivih funkcija (sračunljive funkcija su one funkcije čije vrednosti mogu da se sračunaju pomoću nekog algoritma). Pokazalo se da je taj rezultat mnogo vredniji – ono što je Alonzo Čerč uradio postalo je fundament funkcionalnog programiranja, odnosno funkcionalnih programskih jezika.

11.1.1. Lambda račun

U svom postu **A Brief, Incomplete, and Mostly Wrong History of Programming Languages**, Džejms Iri (James Iry) je napisao i dve izjave koje su tačne:

"1936. - Alan Tjuring je izmislio svaki programski jezik koji će ikada postojati, ali britanska obaveštajna služba ga je kidnapovala da bude 007 pre nego što ga je patentirao."

"1936. - Alonzo Čerč je takođe izmislio svaki jezik koji će ikada biti, ali je to učinio bolje. Njegov lambda račun se ignoriše jer je nedovoljno sličan C-u. Ovaj kriticizam se javlja uprkos činjenici da C još nije izmišljen. "

Ovim svojim izjavama Iri je komentarisao ironično (iz aspekta svesti programerske populacije o značaju, a moglo bi se reći i postojanju) dva teorijska rezultata fundamentalna za razvoj programskih jezika: Tjuringovu mašinu i λ-račun. Oba predstavljaju univerzalni model sračunljivosti i međusobno su ekvivalentni. Ovi rezultati utrli su put razvoju programskih jezika koje zovemo imperativnim (Tjuringova mašina) i programskih jezika koje zovemo funkcionalnim (λ-račun). Pošto se ovde bavimo funkcionalnim programiranjem u nastavku ćemo da objasnimo šta je i kako se u programiranju koristi λ-račun.

11.1.1.1. Čisti netipizirani **λ**-račun

Najjednostavniji, najmanji tip λ-računa je ***čisti netipizirani λ-račun***. Ne primenjuje se direktno u programiranju, ali je jednostavan i sadrži osnovne koncepte koji su dovoljni da se objasni suština korišćenja λ-računa u funkcionalnom programiranja pa ćemo da ga iskoristimo da objasnimo vezu između λ-računa i funkcionalnog programiranja.

Kao i svaki netipizirani λ-račun, on je pre svega konzistentan sistem za pisanje izraza u drugom obliku. Naziva se ***čistim*** jer ima smo ono što je strogo neophodno – sadrži samo ***funkcije*** i ***varijable***.

Čisti netipizirani λ-račun može se formalno definisati putem ***apstrakcija*** i ***aplikacija***.

[1] https://en.wikipedia.org/wiki/Alonzo_Church

11.1.1.1.1. Apstrakcija

Apstrakcija, ili funkcionalna apstrakcija, je parametarski izraz, odnosno funkcija. λ- račun definiše samo funkcije jedne promenljive, ali se lako može proširiti i na funkcije više promenljivih postupkom ugneždavanja funkcija, odnosno definisanjem funkcije koja prima jedan argument i vraća funkciju.

Nova apstrakcija se uvodi korišćenjem simbola **λ**. Apstrakcija se sastoji od ***zaglavlja*** i ***tela*** razdvojenih tačkom (**.**). Zaglavlje sadrži ***simbol*** **λ** i ***ime parametra***. Telo je ***proizvoljan izraz***.

Primer apstrakcije je funkcija identiteta $\lambda x.\, x$ gde je λx zaglavlje a x je telo apstrakcije.

Varijable u telu apstrakcije koje su parametri te apstrakcije (pojavljuju se u zaglavlju apstrakcije) nazivaju se ***vezane varijable***. Ostale varijable se nazivaju ***slobodne varijable***. Sledeći primer ilustruje vezane i slobodne varijable.

Primer 1: Neka je dat izraz

$$\lambda y.\, (\lambda z.\, xyz) z.$$

Varijable y, z u telu druge apstrakcija $(\lambda z.\, xyz)$ u posmatranom izrazu su vezane. Varijabla x je svuda slobodna. Varijabla z u spoljašnjoj apstrakciji $\lambda y. \ldots z$ je takođe slobodna, jer je ni jedna apstrakcija ne vezuje (ne uvodi) pre no što se varijabla koristi. Sva vezivanja su lokalna za apstrakciju, ne za izraz.

Izraz bez slobodnih varijabli zove se ***zatvoreni term*** ili ***kombinator***.

Sledeći izraz je kombinator zato što nema slobodnih varijabli:

$$\lambda x.\, \lambda y.\, \lambda z.\, xyz.$$

U njemu su tri apstrakcije ($\lambda x, \lambda y$ i λz) tri varijable (x, y, z). Pri tome je varijabla x vezana u apstrakciji λx , varijabla y vezana je u apstrakciji λy, i varijabla z je vezana u apstrakciji λz.

Imena vezanih varijabli mogu se proizvoljno birati. Izrazi koji se razlikuju samo po imenima vezanih varijabli zovu se *alfa-ekvivalenti*. Dakle, izrazi $\lambda x.\, x$ i $\lambda y.\, y$ su alfa-ekvivalenti, dok izrazi $\lambda x.\, y$ i $\lambda y.\, z$ nisu alfa-ekvivalenti.

Izrazu se može dodeliti ime. Za to se obično koristi znak jednakosti (=) kao u sledećem primeru u kome se apstrakciji $\lambda x.\, x$ dodeljuje ime $identitet$:

$$identitet = \lambda x.\, x.$$

11.1.1.1.2. Aplikacija

Aplikacija je **operacija kojom se parametri u apstrakciji zamenjuju određenom vrednošću**. To je jedina operacija definisana u čistom netipiziranom lambda računu.

Jedan analogija u programiranju na koju smo mi navikli je supstitucija parametara u pozivu funkcije argumenatima.

Prva rečenica prethodnog paragrafa iskazuje suštinu aplikacije. Ipak, samo još nekoliko napomena u vezi sa notacijom.

U standardnoj notaciji aplikacija se predstavlja ***praznim mestom***. Na primer, $\boldsymbol{f}\ x$ znači da se $\boldsymbol{f}$ primenjuje na x. Problem je što ti blanko znaci mogu da dovedu do dvosmislenosti. Da bi se izbegle te dvosmislenosti, koriste se male zagrade po potrebi pa ćemo i mi to da radimo. Dodatno, pretpostavićemo da je aplikacija levo asocijativna i da vezuje "čvršće" nego apstrakcija. Uz te pretpostavke, $\boldsymbol{f}\ a\ b\ c$ je isto što i $\big((\boldsymbol{f}\ a)\ b\big)\ c$, odnosno znači da se $\boldsymbol{f}$ prvo primenjuje na a, zatim se dobijeni rezultat primene primenjuje na b, i konačno se poslednji rezultat primene primenjuje na c. Zbog pretpostavke da aplikacija vezuje "čvršće" nego apstrakcija, nisu potrebne zagrade oko tela apstrakcije. Međutim, može se pojaviti potreba za zagradama cele apstrakcije. Kako su sve apstrakcije

univarijetetne (imaju jednu vezanu varijablu) po definiciji, ne postoji posebna sintaksa za multivarijetetnu aplikaciju. Zagrade se koriste i za ugneždene apstrakcije, na primer $\lambda x.\, \lambda y.\, xy$ se piše kao $\lambda x.\, (\lambda y.\, (xy))$.

11.1.1.2. Računanje u λ-računu

Korak računanja u λ-računu zove se ***beta-redukcija***. Predstavlja ***pojedinačnu aplikaciju jednog izraza na drugi izraz***.

Beta-redukcija koristi sledeće pravilo:

Neka je zadat izraz oblika $(\lambda x.\, t_1)t_2$ - aplikacija neke apstrakcije λx na izraz t_2:

1. Preimenovati vezane varijable u t_1 i t_2 da se izbegne dvoznačnost. Novi izrazi, t_1' i t_2' koji se dobiju ovim preimenovanjem moraju da budu alfa-ekvivalentni sa izrazima t_1 i t_2 respektivno.
2. Supstituisati sve instance (pojave) vezane varijable x u t_1 sa t_2. Ono što se dobije zamenom je rezultat koraka računanja.

Svaki *izraz na koji se može primeniti beta-redukcija* naziva se ***redeks*** (skraćenica za termin *reducible expression* – svodljiv izraz). Sledi primer redeks izraza:

$$(\lambda x.\, \lambda y.\, xy)(\lambda x.\, xy)$$

Međutim, u njemu je identifikator $\boldsymbol{y}$ dvoznačan. U prvom termu $(\lambda x.\, \lambda y.\, xy)$ on identifikuje vezanu varijablu, jer se isto ime $\boldsymbol{y}$ pojavljuje u zaglavlju apstrakcije $\boldsymbol{\lambda y}$. U drugom termu $(\lambda x.\, xy)$identifikuje slobodnu varijablu, jer se $\boldsymbol{y}$ ne pojavljuje u zaglavlju apstrakcije (argument druge apstrakcije je identifikovan identifikatorom $\boldsymbol{x}$). Da bi se redukcija ispravno uradila, varijabla $\boldsymbol{y}$ se preimenuje u $\boldsymbol{z}$ i na taj način se dobije izraz u kome nema dvoznačnih identifikatora:

$$(\lambda x.\, \lambda z.\, x\ z)(\lambda x.\, x\ y)$$

Nakon ovog preimenovanja redukcija (supstitucija vezane varijable x u telu apstrakcije) se može bezbedno nastaviti i doći do ispravnog rezultata:

$$(\lambda z.\, \lambda x.\, x\ y)\ z$$

Pitanje koje se ovim primerom nameće je da li bi trebalo beta-redukciju primenjivati i u telu apstrakcije što nas dovodi do ***evalucione strategije***.

11.1.1.2.1. Strategije evaluacije

Strategija evaluacije je pravilo koje definiše ***koji se redeksi redukuju*** i ***kojim redosledom se redukuju***.

Najjednostavniji pristup je redukcija redeksa u bilo kom proizvoljnom redosledu sve dok ništa ne ostane za redukciju. Ovaj pristup se naziva ***potpuna beta redukcija***. Ali tu ima nekih problema. Na primer, mogu se dobiti različiti među rezultati u zavisnosti od redosleda redukcije. Zato je bolje redukciju vršiti određenim redosledom. Postoji više načina da se taj redosled definiše.

- Da redukcija počne sa leve ili desne strane izraza;
- Da se počne sa najspoljašnjijim ili najunutrašnjijim redeksom;
- da se redeksi unutar tela apstrakcije redukuju ili ne redukuju.

11.1.1.2.1.1. Pravac redukcije

Izraz se može redukovati sa leva na desno ili zdesna nalevo. Ova odluka je laka: kretanje sa leva na desno je bolje jer se proces završava na svim termovima za koje terminira kada se ide sa desna na levo, a zatim i nekim za koje pravac sa desna na levo ne terminira .

Redukcija od spolja ka unutra ili od unutra ka spolja

Drugo, možemo birati da li da prvo redukujemo najspoljašnjiji ili najdublji redeks. Najspoljašnjiji redeks je redeks koji se ne nalazi ni u jednom drugom redeksu, a najdublji redeks je redeks koji ne sadrži nikakve druge redekse. Da bismo bolje videli na šta se misli, razmotrićemo sledeći izraz:

$$(\lambda x.\, x)((\lambda y.\, y)\ z).$$

Ovaj izraz sadrži dva redeksa, samog sebe i $(\lambda y.\, y)\ z$

Prvi nije sadržan ni u jednom drugom redeksu, tako da je to najspoljašnjiji redeks. Podizraz ne sadrži nijedan drugi redeks, tako da je on najdublji (najunutrašnjiji).

Redukcija od koja počinje od najlevljeg najspoljašnijeg redeksa naziva se strategijom normalnog poretka, a redukcija koja počinje od najlevljeg najunutrašnijeg redeksa naziva se strategijom aplikativnog poretka.

Strategija aplikativnog poretka prvo redukuje sve argumente (sa leva na desno), a zatim primenjuje apstrakciju, dok strategija normalnog poretka radi obrnuto prvo primenjuje apstrakciju a zatim redukuje argumente.

Za izraze koji se ne mogu redukovati korišćenjem izabrane strategije evaluacije kaže se da su *u normalnom obliku*.

Ako dati term ima normalan oblik u odnosu na obe pomenute strategije, on je isti za obe. Međutim, strategija aplikativnog poretka je slabija od strategije normalnog poretka: neki termovi koji se normalizuju primenom strategije normalnog poretka ne normalizuju se primenom strategije aplikativnog poretka.

Redukcija u telu apstrakcije ili ne

Dve gore navedene strategije mogu da uđu u telo apstrakcije kako bi pronašli redekse. Iz tog razloga ih nazivaju *jakim*.

Postoje i strategije koje ne mogu da uđu u tela apstrakcije, koja se nazivaju *slabim*. Kada se govori o programskim jezicima, ne moraju se nužno pregledati tela funkcija, tako da se slabe strategije u slučaju programskih jezika smatraju praktičnijim.

Slaba strategija koja, u slučaju programskih jezika, odgovara normalnom redosledu naziva se *poziv po imenu*, a ona koja odgovara aplikativnom redosledu zove se *poziv po vrednosti*.

Poziv po imenu odgovara lenjoj evaluaciji. U lenjoj evaluaciji, argumenti se ne evaluiraju do trenutka kada su potrebni što znači da se neki argumenti uopšte ne evaluiraju, ali se neki koraci redukcije mogu duplirati. Veoma mali broj programskih jezika implementira lenju evaluaciju na ovaj način (jedini pravi primer je Algol 60). Praktičnija varijacija poziva po imenu je *poziv po potrebi*, koja se, kažu, koristi u jeziku Haskell. Poziv po potrebi se ponaša kao poziv po imenu, osim što se ne dupliraju koraci redukcije. Mehanizam radi tako što proračun koji odgovara nekom argumentu „dostavlja“ svuda gde se argument pojavljuje, tako da se svako zajedničko izračunavanje izvodi najviše jednom.

Poziv po vrednosti, s druge strane, odgovara pohlepnoj evaluaciji. U pohlepnoj evaluaciji, svi argumenti se uvek evaluiraju pre poziva funkcije. Većina programskih jezika podrazumevano primenjuje ovu strategiju.

Kao što je aplikativna strategija poretka inferiorna u odnosu na normalnu strategiju poretka, poziv po vrednosti je inferioran u odnosu na poziv po imenu.

11.1.1.2.2. Eta-redukcija

Spomenućemo ovde još i eta-redukciju zato što je ona tesno povezana sa stilom programiranja koji se u funkcionalnoj paradigmi javlja pod nazivom *point-free style*. Point-free style je stil programiranja koji dozvoljava da se izostavi navođenje argumenata pri deklaraciji funkcije.

Baziran je na eta-redukciji koja kaže: Ako se x ne pojavljuje kao slobodna vafijabla u $\boldsymbol{f}$, važi sledeće

$$\lambda x.\, \boldsymbol{f}\ x = \boldsymbol{f}$$

Dakle, eta-redukcija nam omogućuje da definišemo funkcije bez eksplicitnog navođenja parametara, odnosno garantuje nam ispravnost skraćivanja poput sledećeg: Umesto $\boldsymbol{g}\ x\ y\ z = \boldsymbol{f}\ (\boldsymbol{h}\ x)\ y\ z$ možemo pisati samo $\boldsymbol{g} = \boldsymbol{f} \cdot \boldsymbol{h}$.

11.1.1.3. Kako λ-račun može da se iskoristi za programiranje na koje smo navikli

I ovo što smo do sada pokazali je programiranje. Ali nije programiranje na koje smo mi navikli, koje koristimo svakodnevno u radu. Pa zašto smo onda uopšte pričali o tom λ-računu?

Odgovor sledi u ovom poslednjem delu priče o λ-računu u kome ćemo pokazati kako pomoću jezika λ-računa mogu da se naprave nama bliske stvari koje koristimo kada programiramo. To znači da svi konstrukti jezika na koje smo navikli poput logičkih izraza, aritmetičkih operacija, petlji, itd. mogu da se predstave kao programi pisani u jeziku koji ima samo varijable i funkcije, odnosno da svi pomenuti konstrukti (u stvari, svi konstrukti svih jezika koji su ikada postojali i koji će ikada postojati) mogu da se isprogramiraju kao funkcije. U nastavku ćemo pokazati primere kojima ćemo demonstrirati kako se u jeziku λ-računa mogu isprogramirati funkcije više promenljivih, logički tipovi, aritmetika prirodnih brojeva, i rekurzija.

11.1.1.3.1. Funkcije više promenljivih

Svaka funkcija više promenljivih je isto što i više ugneždenih funkcija jedne promenljive koje vraćaju druge funkcije. Iskazano notacijom elementarne matematike koju svi znamo to znači sledeće. Ako nam je zadata funkcija tri promenljive $f(x, y, z)$ možemo da je predstavimo kao tri ugneždene funkcije jedne promenljive $f\left(g\big(h(x)\big)\right)$.

U funkcionalnom programiranju takve se funkcije zovu *kurirane funkcije.* U čistom λ-računu sve funkcije su kurirane.

Zarad pojednostavljenja notacije uvešćemo skraćenu sintaksu na sledeći način. Multivarijetetna apstrakcija od n parametara, deklarisana kao

$$\lambda x_1 x_2 \ldots x_n.\, t$$

je isto što i n ugneždenih jednovarijetetnih apstrakcija

$$\lambda x_1.\, \lambda x_2.\, \ldots \lambda x_n.\, t$$

Ili malo čitljivije:

$$\lambda x_1.\, \big(\lambda x_2.\, (\ldots \lambda x_n)\big).\, t$$

Apstrakcije funkcije više promenljivih se mogu parcijalno aplicirati (parcijalna aplikacija je postupak u kome se vrši supstitucija dela parametra i vraća se apstrakcija koja ima preostale, nesupstituisane parametre) što znači da se izraz

$$(\lambda xyz.\, (xy)(xz))ab$$

redukuje na

$$\lambda z.\, (ab)(az).$$

11.1.1.3.2. Logičke vrednosti I logičke funkcije

Čerč je u svom originalnom radu[2] predstavio logičke vrednosti tačno i netačno na sledeći način.

Deklarišu se dva kombinatora:

$$tru = \lambda\, x\, y.x, \text{ i}$$

$$fls = \lambda\, x\, y.y.$$

Data su im imena tru i fls da bi se razlikovali od stvarnih logičkih vrednosti `true` i `false` koje u stvari nisu apstrakcije i kao takve nisu deo čistog λ-računa, ali se mogu uvesti kao termovi u nečistom λ-računu.

Struktura ovih definicija je prilično neobična. tru je funkcija sa dva argumenta koja vraća svoj prvi argument i ignoriše drugi, dok fls vraća drugi argument i ignoriše prvi.

Ako se malo "izmaknete", prepoznaćete da ovo kodira računanje grananja, odnosno `if` naredbu koja je neizostavni deo svakog programskog jezika.

Kombinator $ifThenElse = \lambda\, c\, t\, f.ctf$ je, u stvari, funkcija koja kodira naredbu `if ... then ... else`.

Sa ovako definisanim logičkim vrednostima se mogu napraviti logičke funkcije. Pokazaćemo kako se može predstaviti operator konjunkcije (logičko I, AND). Neka je naša apstrakcija za operaciju konjunkcije definisana na sledeći način:

$$and = \lambda\, x\, y.x\ y\ fls$$

Po definiciji, konjunkcija je binarna funkcija (ima dva argumenta) definisana nad skupom logičkih vrednosti (njeni argumenti mogu da uzmu vrednost tru ili fls) koja vraća vrednost tru ako su vrednosti oba argumenta tru, a u svim ostalim slučajevima vraća vrednost fls.

Način da potvrdimo da naša apstrakcija and radi ono što je nameravano je da je primenimo. Pokazaćemo kako izgleda njena primena za konfiguraciju ulaznih vrednosti $tru\ tru$ i $tru\ fls$. Primena se svodi na evaluaciju odgovarajućeg λ-izraza koji je, u stvari, primena apstrakcije and na apstrakcije tru:

$$and\ tru\ tru = (\lambda\, x\, y.x\ y\ fls)(\lambda\, x\, y.x)(\lambda\, x\, y.x).$$

Sada se primenjuje beta-redukcija da bismo evaluirali naš izraz. Očekujemo da kao rezultat dobijemo apstrakciju tru.

Prvo se preimenuju vezane varijable da bismo izbegli dvoznačnost:

$$(\lambda\, x\, y.x\ y\ fls)(\lambda\, a\, b.a)(\lambda\, c\, d.c).$$

Sada se redukuje ovaj izraz (recimo primenom strategije poziv-po-vrednosti). Strelica u sledećoj formuli predstavlja jedan korak redukcije.

$$(\lambda\, x\, y.x\ y\ fls)(\lambda\, a\, b.a)(\lambda\, c\, d.c) \rightarrow (\lambda y.(\lambda\, a\, b.a)\ y\ fls)(\lambda\, c\, d.c) \rightarrow$$
$$(\lambda\, a\, b.a)(\lambda\, c\, d.c)\ fls \rightarrow \big(\lambda b.(\lambda\, c\, d.c)\big)\ fls \rightarrow (\lambda c\, d.c) = tru.$$

Rezultat redukcije je apstrakcija $\lambda c\ d.c$ što je alfa-ekvivalentno sa apstrakcijom $tru = \lambda\, x\, y.x$.

Na potpuno analogan način pokazuje se da apstrakcija and vraća apstrakciju fls za ostale kombinacije parametara (fls, tru) i (fls, fls). Na primer, za kombinaciju (fls, tru) evaluacija izgleda ovako. Redukuje se izraz

$$and\ fls\ tru = (\lambda\, x\, y.x\ y\ fls)(\lambda\, x\, y.y)(\lambda\, x\, y.x)$$

[2] Ovo napominjemo zbog toga što važi da se vrednost može predstaviti funkcijom na beskonačan broj načina.

na sledeći način:

$(\lambda\, x\, y.x\, y\, fls)(\lambda\, a\, b.b)(\lambda\, c\, d.c) \rightarrow (\lambda y.(\lambda\, a\, b.b)\, y\, fls)(\lambda\, c\, d.c) \rightarrow$
$(\lambda\, a\, b.b)(\lambda\, c\, d.c)\, fls \rightarrow (\lambda a\, b.b)\, fls = fls.$

Na analogan način mogu se funkcijama predstaviti i drugi logički operatori (na primer, logičko ILI i negacija).

11.1.1.3.3. Prirodni brojevi i aritmetika prirodnih brojeva

Brojevi i računanja sa brojevima (aritmetika) su delovi svakog računarskog programskog jezika. Čerč je konstruisao i postupak koji omogućuju predstavljanje prirodnih brojeva pomoću λ-računa. Postupak definiše prirodne brojeve na način sličan Peanovoj konstrukciji[3] koja se svodi na kodiranje nule i primenu takozvane *naslednička funkcije* polazeći od nule.

Čerč je koristio sledeće definicije:

$c0 = \lambda\, s\, z.z$,
$c1 = \lambda\, s\, z.sz$,
$c2 = \lambda\, s\, z.s\ (s\ z)$,
itd.

Ovo kodiranje prirodno kodira koncept brojanja. Svaki Čerčov prirodan broj je apstrakcija sa dva parametra: kombinatora "sledeći" označenog sa s i "početne vrednosti" označene sa z. Apstrakcija prima ta dva parametra i zatim "broji" odgovarajuće "sledeće" počinjući od z.

Sada se može definisati naslednička funkcija (zvaćemo je $nasl$) koja prima prirodan broj i vraća sledeći prirodan broj. Telo apstrakcije naslednička funkcije treba još jednom na zadati broj da primeni argument "sledeći".

Kako su prirodni brojevi definisani kao apstrakcija sa dva parametra, naslednička funkcija biće apstrakcija sa tri parametra (tri ugnježdene apstrakcije). Telo apstrakcije naslednička funkcije još jednom na broj (n) primenjuje argument "sledeći" (s):

$$nasl = \lambda nsz.\, s\ (n\ s\ z).$$

Izraz se može napisati i na sledeći način iz koga se, možda, bolje vidi šta se dešava:

$$nasl = \lambda n.\, \lambda s\ z.\, s\ (nsz).$$

U istom maniru mogu se definisati sabiranje i množenje:

$$plus = \lambda m\ n.\, \lambda s\ z.\ m\ s(nsz),$$
$$puta = \lambda m\ n.\, \lambda s\ z.m\ (ns)\ z.$$

Provera da li je u pitanju nula može se uraditi apstrakcijom $jeNula = \lambda n.n\ (\lambda x.\ fls)\ tru$.

Međutim, pri definisanju oduzimanja, odnosno predačke funkcije, nailazi se na problem. Naravno, moguće je definisati oduzimanje, ali ni blizu elegantno i efikasno kao kod sabiranja i množenja. Rešenje koje ćemo prikazati koristi apstrakciju *para* vrednosti dva susedna prirodna broja. Na taj način može se napraviti brojač koji "zaostaje za jedan". Zbog toga treba prvo da definišemo apstrakciju *para*.

Apstrakcija para

Apstrakcija para (može se proširiti i na torku) kodira se korišćenjem apstrakcija logičkih vrednosti tru i fls kao apstrakcija koja vraća jednu vrednost za ulaz tru a drugu vrednost za ulaz fls.

Prvo se definiše kombinator konstruktora para:

[3] Primer Peanove konstrukcije u jeziku Haskell može se videti na linku https://wiki.haskell.org/Peano_numbers

$$par = \lambda f\ s\ b.bfs$$

Ova apstrakcija ozbiljno "liči" na konstruktor If ... Then ... Else ($ifThenElse = \lambda c\ t\ f.ctf$). Razlika je samo u tome što je ovde uslov poslednji parametar.

Sada se mogu definisati apstrakcije destruktora:

$$prvi = \lambda p.p\ tru$$
$$drugi = \lambda p.p\ fls$$

Zarad jednostavnije notacije uvode se dve pomoćne apstrakcije:

$$zz = par\ c_0 c_0$$
$$ss = \lambda nn.par(drugi\ n\ n)(nasl(drugi\ n\ n))$$

Ovde je prvi element para prethodnik a drugi je naslednik. zz kodira početnu tačku a ss je zaostajući naslednik.

Sada se može definisati prethodnička funkcija:

$$pred = \lambda n.fst(n\ ss\ zz)$$

Sa ovakvom definicijom je $pred\ c_0 = c_0$ što nije idealno. Međutim, kako nema kodiranja negativnih brojeva, i nema baš puno izbora. Ako se pojavi potreba, može se koristiti bilo koja druga vrednost kao prvi element u zz da bi se signalizirala greška.

11.1.1.4. Rekurzija

Pre no što se pozabavimo načinom opisivanja rekurzije pomoću λ-računa, kazaćemo nešto o izrazima za koje se beta-redukcija nikada ne završava. Takvi izrazi nemaju normalnu formu i kaže se da *divergiraju*.

Najjednostavniji slučaj divergentnog izraza je primena takozvanog ***omega kombinatora*** (Ω) :

$$\Omega = \lambda x.x\ x$$

na samog sebe. Korišćenje bilo koje strategije evaluacije za evaluaciju izraza Ω Ω kao rezultat daje ponovo Ω Ω .

Omega kombinator sam za sebe nema praktičnu primenu. Međutim, može se proširiti tako da se pomoću njega kodira apstrakcija rekurzije koja je od izuzetnog značaja u funkcionalnom programiranju. Podsetimo se: rekurzija je proces u kome funkcija poziva samu sebe .

Za opisivanje apstrakcije rekurzija može se koristiti ***kombinator fiksne tačke*** koji se uobičajeno označava sa Y[4]:

$$Y = \lambda f.(\lambda x.f\ (x\ x))(\lambda x.f\ (x\ x)).$$

Ekvivalentan kraći zapis je:

$$X = \lambda f.\Omega(\lambda x.f\ (x\ x)).$$

Ovde je ideja da se da mogućnost da se izrazi rekurzija tako što će se proslediti funkcija (f) kao prvi argument apstrakcije. Omega kombinator beskonačno replicira sam sebe. Y kombinator koristi istu potencijalno beskonačno replicirajuću strukturu za kodiranje rekurzije.

[4] Napomena: ovo nije jedini kombinator fiksne tačke. Postoji i čuveni Tjuringov kombinator koji ima sledeći oblik: $\Theta = (\lambda x\ y.y\ (x\ x\ y))(\lambda xy.y(x\ x\ y))$.

U praksi to funkcioniše na sledeći način. Posmatra se funkcija koja odbrojava unazad do nule koristeći $pred$ kombinator definisan prethodno, i vraća nulu kada je argument nula. Da bismo je definisali, dodaćemo rekurzivni poziv kao njen prvi argument:

$$cdown = \lambda f.\lambda n.ifThenElse\ (jeNula\ n)c_0(f(pred\ n)).$$

$Y\ cdown$ funkcioniše na sledeći način:

$Y\ cdown\ c_2 = \Big(\lambda f.\big(\lambda x.f\ (x\ x)\big)\big(\lambda x.f\ (x\ x)\big)\Big)\ cdown\ c_2 \rightarrow$

$\big(\lambda x.cdown\ (x\ x)\big)\big(\lambda x.cdown\ (x\ x)\big)\ c_2 \rightarrow$

$cdown((\lambda x.cdown(x\ x))(\lambda x.cdown(x\ x)))c_2 \Rightarrow$

$\big(\lambda x.cdown(x\ x)\big)\ \big(\lambda x.cdown(x\ x)\big)\ (pred\ c_2) \Rightarrow \big(\lambda x.cdown(x\ x)\big)\ (\lambda x.cdown(x\ x))c_1$

$\rightarrow cdown\ \Big(\big(\lambda x.cdown(x\ x)\big)\ \big(\lambda x.cdown(x\ x)\big)\Big)\ c_1 \Rightarrow$

$\big(\lambda x.cdown(x\ x)\big)\ (\lambda x.cdown(x\ x))(pred\ c_1) \Rightarrow (\lambda x.cdown(x\ x))(\lambda x.cdown(x\ x))c_0$

$\rightarrow cdown(\big(\lambda x.cdown(x\ x)\big)\ (\lambda x.cdown(x\ x)))c_0 \Rightarrow c_0.$

Na kraju moramo da kažemo da je ovaj primer samo ilustrativan i da u njemu nisu eksplicitno iskazana ograničenja koja se odnose na strategiju redukcije i koja ovde nisu ni malo naivna. U primeru je korišćena potpuna beta-redukcija sa pažljivo izabranim termovima koji će se prvi redukovati. Ako bi se primenila redukcija po vrednosti, isti primer bi vrlo brzo „zaglavio“ pokušavajući da evaluira izraz $((\lambda x.cdown(x\ x))(\lambda x.cdown(x\ x)))$ koji divergira pri redukciji po vrednosti. Naravno, postoje i kombinatori fiksne tačke koji konvergiraju pri redukciji po vrednosti. Jedan primer naveden je u izvoru dostupnom na lokaciji https://serokell.io/blog/untyped-lambda-calculus koji je nama bio osnova za izlaganje o λ-računu.

Poglavlje 12: O teoriji kategorija

Cilj ovoga odeljka je da čitaoca upozna sa osnovnim idejama jedne oblasti matematike koja je posejana u tlo programiranja donela koristan plod – funkcionalno programiranje. Teorija kategorija je matematički aparat koji se odlikuje najvišim nivoom apstrakcije u matematici koja je, sama po sebi, iznad svega apstrakcija. Zajedno (i u tesnoj vezi) sa λ-računom, ona je teorijska osnova koja funkcionalnim programskim jezicima daje moć koju programski jezici bez rigoroznog matematičkog fundamenta nemaju.

Na početku ovoga odeljka biće prikazani osnovni koncepti i terminologija teorije kategorija. Način izlaganja biće prilagođen cilju da čitalac stekne neku vrstu intuitivnog razumevanja osnovne ideje teorije kategorija i da, kroz primere, stekne uvid u primenu teorije kategorija u računarskom programiranju. Ideja o načinu izlaganja i najveći deo sadržaja ovog odeljka potiču iz izvora [1], odlične polazne tačka za čitaoca koji želi da stekne dublje razumevanje jezika funkcionalnog programiranja.

12.1.1. Kategorije

Definicija 17.2.1. Kategorija C obuhvata:

1. Kolekciju objekata;
2. Kolekciju morfizama (strelica);
3. Operacije koje svakom morfizmu f dodeljuju objekat $dom\, f$ (domen) i objekat $cod\, f$ (kodomen). Oznaka $f : A \to B$[5] koriste se da se označi da je $dom\, f = A$ i $cod\, f = B$, a oznaka $C(A, B)$ koristi se da se označi kolekcija svih morfizama sa domenom A i kodomenom B.
4. Operator kompozicije (∘) koji svakom paru morfizama f i g , za koje je $cod\, f = dom\, g$, dodeljuje kompozitni morfizam $g \circ f : dom\, f \to cod\, g$ tako da je zadovoljen *zakon asocijativnosti* koji kaže:

   Za proizvoljne morfizme $f : A \to B$, $g : B \to C$, i $h : C \to D$ (gde A, B, C, i D nisu nužno različiti), važi $h \circ (g \circ f) = (h \circ g) \circ f$;

5. Za svaki objekat A, identitetski morfizam $id_A : A \to A$ koji zadovoljava *zakon identiteta* koji kaže: za svaki morfizam $f : A \to B$, $id_B \circ f = f$ i $f \circ id_A = f$.

Definicija koju smo naveli omogućuje da se konstruišu različite kategorije. Konstruisanje se sada svodi na smišljanje objekata i morfizama takvih da budu zadovoljeni zakon asocijativnosti morfizama i zakon identiteta koji se nazivaju *zakoni kategorije*.

12.1.1.1. Kategorija skupova (Set) i konkretne kategorije

U našoj definiciji kategorija je definisana putem pojma skupa (kolekcija) gde objekti mogu da budu bilo šta, operacije su funkcije iz teorije skupova, a jednakost je jednakost iz teorije skupova.

I sam skup, kategorijski gledano, može se opisati kao diskretna kategorija. U toj kategoriji, objekti su elementi skupa i u njoj su jedini morfizmi identiteti.

Naša definicija kategorije dozvoljava da usvojimo da naš **objekat** bude ***skup*** a da **morfizam** bude ***funkcija*** bez opasnosti da napravimo cirkularnu definiciju[6] jer se na taj način **ne definiše pojam skupa**

[5] U literaturi se ravnopravno koristi i oznaka $A \xrightarrow{f} B$.

[6] Cirkularna definicija je definicija u kojoj se dva pojma definišu uzajamno jedan putem drugog – pojam A putem pojma B i pojam B putem pojma A.

putem pojma kategorije, već se **predstavlja skup putem kategorije**. Rezultat je **kategorija** zvana ***kategorija skupova*** ili **Set kategorija** koja je vrlo zgodna za sticanje intuicije o samoj ideji kategorije.

Set kategorija je prototip[7] za kategorije koje se zovu ***konkretne kategorije*** o kojima ćemo govoriti u ovom odeljku.

U kategoriji **Set** objekti su skupovi a morfizmi su totalne[8] funkcije među skupovima. Kompozicija morfizama je kompozicija funkcija, a identitetski morfizam je identitetska funkcija. Dakle, konstruisali smo Set kategorija koja izgleda ovako:

1. Objekat u Set kategoriji je skup.
2. Morfizam $f : A \to B$ u Set kategoriji je totalna funkcija[9] iz skupa A u skup B.
3. Za svaku totalnu funkciju f sa domenom A i kodomenom B, važi $dom\ f = A$, $cod\ f = B$, i $f \in \boldsymbol{Set}(A, B)$.
4. Kompozicija totalne funkcije $f : A \to B$ sa drugom totalnom funkcijom $g : B \to C$ je totalna funkcija iz A na C koja mapira svaki element $a \in A$ na $g(f(a)) \in C$. Kompozicija totalnih funkcija na skupove zadovoljava *zakon asocijativnosti*: za svaku funkciju $f : A \to B$, $g : B \to C$ i $h : C \to D$, važi $h \circ (g \circ f) = (h \circ g) \circ f$.
5. Za svaki skup A, identitetska funkcija $id_A : A \to A$ je totalna funkcija sa domenom i kodomenom A . Za svaku funkciju $f : A \to B$, identitetske funkcije nad A i B zadovoljavaju jednačine *zakona identiteta* $id_B \circ f = f$ i $f \circ id_A = f$.

Naravno, potrebno je dokazati da za ovako konstruisanu kategoriju važe *asocijativnost kompozicije* i *zakon identiteta*. U slučaju Set kategorije to sledi očigledno iz definicije. Međutim, u nekim drugim situacijama izvođenje ovih dokaza nije baš trivijalno.

U nastavku ćemo da prikažemo još nekoliko kategorija iz klase ***konkretnih kategorija*** sa ciljem da čitaoca upoznamo sa značajnim konkretnim kategorijama, ali i sa ciljem da pokažemo kako ograničenja koje se nameću na objekte i morfizme Set kategorije vode ka objektima/skupovima sa strukturom i morfizma koji tu strukturu očuvavaju.

12.1.1.2. Kategorija Poset

Kategorija **Poset** je slična **Set** kategoriji, samo su objekti i morfizmi malo drugačiji. Kada kažemo „malo drugačiji“, to znači da su na objekte (skupovi) i na morfizme nametnuta neka ograničenja.

Za razliku od Set kategorije u kojoj su objekti **proizvoljni skupovi** a morfizmi **proizvoljne totalne funkcije**, kategorija Poset ima za **objekte parcijalno uređene skupove** a za **morfizme totalne funkcije** koje **očuvavaju uređenje**.

Podsećanje:
Parcijalno uređenje ($\leq_P$) na skupu P je refleksivna, tranzitivna i antisimetrična relacija nad elementima skupa P što znači da za svako $p, p', p" \in P$ važi:
(1) $p \leq p$;
(2) $p \leq p' \leq p" \Rightarrow p \leq p"$, i
(3) $p \leq p'$ i $p' \leq p \Rightarrow p = p'$.

Funkcija koja očuvava uređenje (monotona funkcija) sa $(P, \leq_P)$ na $(Q, \leq_Q)$ je funkcija $f : P \to Q$ takva da važi:
$p \leq_P p' \Rightarrow f(p) \leq_Q f(p')$.

---

[7] Ovde termin prototip znači da je u pitanju kategorija nad kojom se na neki dobro-definisan način mogu izgraditi druge kategorije.

[8] Totalna funkcija je funkcija čija je vrednost definisana za sve vrednosti iz domena (skupa vrednosti).

[9] Ovde postoji jedan suptilan detalj: jednoj funkciji nad skupovima odgovara više morfizama u Set kategoriji pa je korektnije da se morfizam posmatra kao torka (f, B), gde je f totalna funkcija sa domenom A , a B je skup koji sadrži vrednosti funkcije f.

Sledi prepis u "definicioni format" kategorije **Poset** dopunjen verifikacijom asocijativnosti kompozicije i zakona identiteta:

1. Objekat u kategoriji **Poset** je skup P nad čijim elementima važi refleksivna, tranzitivna i antisimetrična relacija $\leq_P$.
2. Morfizam $f\colon (P, \leq_P) \to (Q, \leq_Q)$ u kategoriji **Poset** je totalna funkcija iz P na Q koja očuvava uređenje nad P, odnosno zadovoljava uslov: ako je $p \leq_P p'$, tada je $f(p) \leq_Q f(p')$.
3. Za svaku totalnu funkciju f sa domenom P i kodomenom Q koja očuvava poredak, je $\boldsymbol{dom}\, \boldsymbol{f} = (P, \leq_P)$, $\boldsymbol{cod}\, \boldsymbol{f} = (Q, \leq_Q)$, i $f \in \boldsymbol{Poset}((P, \leq_P), (Q, \leq_P))$.
4. Kompozicija dve totalne funkcije koje očuvavaju poredak $f : P \to Q$ i $g : Q \to R$ je totalna funkcija $g \circ f$ iz P na R. Dalje, ako $je\ p \leq_P p'$ tada važi $f(p) \leq_Q f(p')$ zato što f očuvava poredak skupa P, a zato što g očuvava poredak skupa Q, važi $g(f(p)) \leq_R g(f(p'))$ odakle sledi da je kompozicija $g \circ f$ funkcija koja očuvava poredak. Kompozicija funkcija koje očuvavaju poredak je asocijativna zato što je svaka funkcija koja očuvava poredak nad parcijalno uređenim skupom samo funkcija nad skupovima, a kompozicija funkcija nad skupovima je asocijativna.
5. Za svako parcijalno uređenje $(P, \leq_P)$, funkcija identiteta id_P očuvava poredak nad P i zadovoljava jednačine zakona identiteta.

12.1.1.3. Kategorija Mon

Objekti kategorije mogu da budu bilo šta pa i algebarske strukture. Jedna takva struktura je algebarska struktura monoid.

Definicija: ***Monoid*** je Skup M sa binarnom operacijom $M \times M \to M$ koja se označava sa • ako su zadovoljena sledeća dva aksioma:

- **Asocijativnost**: Za svako a, b i c u M, važi jednačina $(a \bullet b) \bullet c = a \bullet (b \bullet c)$.
- **Identitetski element**: Postoji element e u M takav za svaki element a u M važe jednačine $e \bullet a = a$ i $a \bullet e = a$.

Monoidski homomorfizam iz $(M, \bullet, e)$ na $(M', \bullet', e')$ je funkcija $f\colon M \to M'$ takva da važi $f(e) = e'$ i $f(x \bullet y) = f(x) \bullet' f(y)$. Kompozicija dva monoidska homomorfizma je isto što i njihova kompozicija kao funkcija nad skupovima.

Kategorija Mon ima za **objekte *monoide***, a za **morfizam *monoidski homomorfizam***. Da bi se verifikovalo da je Mon stvarno kategorija, treba pokazati da je kompozicija dve funkcije koje su homomorfizmi funkcija koja je takođe homomorfizam.

Još opštije, algebre (jednostavno rečeno, algebra je skup i kolekcija operacija nad tim skupom) sa zadatom signaturom formiraju objekte kategorije. Da, ipak, prevedemo ovo na jezik razumljiv bar malom delu smrtnika.

Neka je Ω skup operatorskih simbola sa mapiranjem elemenata skupa Ω na prirodne brojeve; za svako $\omega \in \Omega$, $ar(\omega)$ je **arnost** operatora ω.
Ω -algebra A je skup $|A|$ (*nosač* od A) i, za svaki operator ω arnosti $ar(\omega)$, funkcija $a_\omega\colon |A|^{ar(\omega)} \to |A|$, koja se zove *interpretacija od* ω i koja mapira $ar(\omega)$-torke elemenata nosača natrag na nosač.
Ω -homomorfizam iz Ω -algebre A na Ω -algebru B je funkcija $h : |A| \to |B|$ takva da za svaki operator $\omega \in \Omega$ i torku $x_1, x_2, \ldots, x_{ar(\omega)}$ elemenata skupa $|A|$ važi sledeća jednačina:

$$h(a_\omega(x_1, x_2, \ldots, x_{ar(\omega)})) = b_\omega(h(x_1), h(x_2), \ldots, h(x_{ar(\omega)})).$$

Kategorija **Ω -Alg** ima **Ω -algebre** za **objekte** i **Ω -homomorfizme** za **morfizme**.

Ova konstrukcija se može rafinisati dodavanjem signaturi Ω skupa E jednačina između izraza izgrađenih od elemenata iz Ω (operatori) i skupa simbola varijabli $\{x, y, z, \ldots\}$. U tom slučaju Ω -

algebre A za koje su jednačine iz E zadovoljene za sve dodele elemenata iz $|A|$ simbolima varijabli čine objekte kategorije $(\boldsymbol{\Omega}, \boldsymbol{E})$**-Alg**.

U tom slučaju, kategorija **Mon** je $(\boldsymbol{\Omega}, \boldsymbol{E})$**-Alg** gde je $\Omega = \{\bullet, e\}$; $ar(\bullet) = 2$; $ar(e) = 0$; $E = \{(x \bullet y) \bullet z = x \bullet (y \bullet z), e \bullet x = x,\ x \bullet e = x\}$.

Pored konkretnih kategorija postoje i druge klase među kojima i **konačne** kategorije (kategorije čiji su objekti konačni skupovi a morfizmi sve funkcije među njima); primeri su kategorije poput $\mathbf{0}$, $\mathbf{1}$, $\mathbf{2}$, i $\mathbf{3}$.

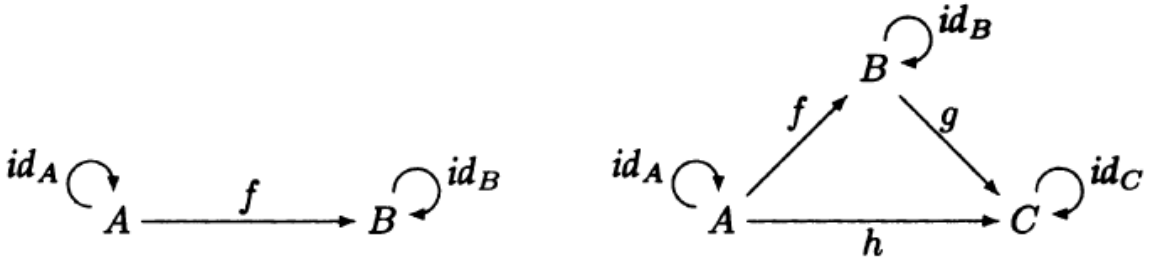
Kategorija $\mathbf{0}$ je kategorija bez objekata i bez morfizama. Asocijativnost i identitet su ovde bespredmetni.

Kategorija $\mathbf{1}$ ima jedan objekat i jedan morfizam. Po zakonu identiteta, taj morfizam mora biti identitetski za objekat. Kompozicija ovog morfizma sa samim sobom može biti samo ona sama, što zadovoljava zakone identiteta i asocijativnosti. Zapazite da se nismo potrudili da navedemo koje matematičke ili fizičke entitete treba da predstavljaju objekat i morfizam. Ovde su bitna samo njihova algebarska svojstva, a ona su u potpunosti određena zakonima kategorija.

Kategorija $\mathbf{2}$ ima dva objekta, dva identitetska morfizma i morfizam od jednog objekta na drugi. Opet, nije bitno šta objekti i morfizmi predstavljaju, ali da bismo lakše razgovarali o njima, mogli bismo objekte označiti A i B i neidentitetsku strelicu sa f. Postoji samo jedan način da se definiše kompozicija pa je lako proveriti da li su zakoni identiteta i asocijativnosti zadovoljeni.

Kategorija $\mathbf{3}$ ima tri objekta (A, B, i C), tri identitetska morfizma, i tri neidentitetska morfizma: $f: A \to B$, $g: B \to C$ i $h: A \to C$. I ovde se kompozicija može definisati samo na jedan način i oba zakona kategorije su zadovoljena. Kako su f, g, i h jedini neidentitetski morfizmi, mora da važi $g \circ f = h$.

Grafički prikaz kategorije $\mathbf{2}$ i kategorije $\mathbf{3}$ dat je na slici 17.2.1:



Slika 17.2.1 Grafički prikaz kategorije $\mathbf{2}$ i kategorije $\mathbf{3}$

Na kraju još da damo primer koji kategoriju približava funkcionalnom programskom jeziku. Prikazaćemo primer jednostavnog jezika u kome su, zarad lakšeg razumevanja, izostale mnoge suštinske stvari (mogućnost prosleđivanja funkcija kao parametara i vraćanja funkcija kao rezultata kao i parametrizovani tipovi podataka, polimorfične funkcije, itd.) koje se takođe mogu modelovati jezikom teorije kategorija. Posmatramo jednostavan funkcionalni programski jezik koji ima:

- Primitivne tipove:

`Int`	celobrojni
`Real`	realni
`Bool`	logički
`Unit`	one-element tip

- Ugrađene opercije

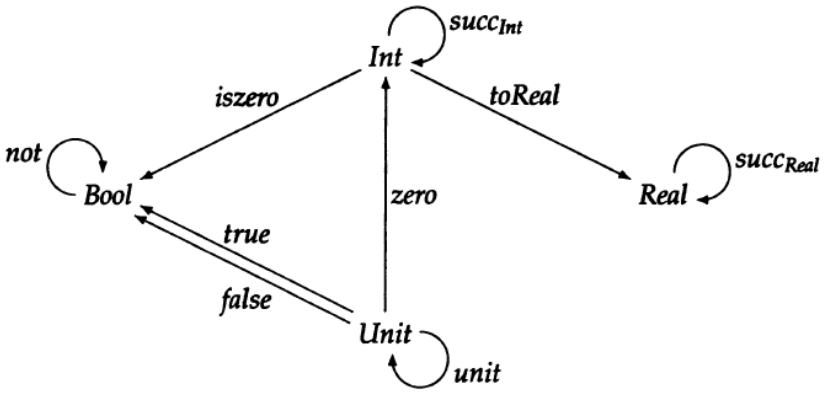
`iszero : Int → Bool`	provera na nulu
`not : Bool → Bool`	negacija
$succ_{Int}$ `: Int → Int`	celobrojni sukcesor
$succ_{Real}$ `: Real → Real`	realni sukcesor
`toReal : Int → Real`	konverzija celobrojni u realni

- Konstante
  `zero: Int`
  `true: Bool`
  `false: Bool`
  `unit: Unit.`

Kategorija koja reprezentuje ovaj jezik (zvaćemo je SFPL) pravi se na sledeći način.

1. Objekti su Int, $Real$, $Bool$, i $Unit$;
2. Morfizmi su $iszero$, not, $succ_{Int}$, $succ_{Real}$, i $toReal$;
3. Konstante $zero$, $true$, $false$, i $unit$ su morfizmi od objekta $Unit$ na objekte Int, $Bool$, $Bool$, i $Unit$ respektivno, koji mapiraju pojedinačni element od $Unit$ na odgovarajuće elemente tih tipova;
4. Dodaju se identitetski morfizmi svakom tipu;
5. Za svaki kompozibilan par morfizama dodaje se morfizam koji se dobija njihovom kompozicijom;
6. Izjednačuju se određeni morfizmi koji predstavljaju iste funkcije prema semantici jezika, poput $false = not \circ true$ i $iszero \circ zero = true$.

Rezultat je ilustrovan dijagramom na slici 17.2.2.



Slika 17.2.2 Kategorija SFPL

12.1.2. Pravljenje kategorija od kategorija

Pored kategorija koje smo do sada videli, a koje reprezentuju matematičke objekte iz drugih domena matematike, postoje vrlo korisne kategorije koje se mogu dobiti izgradnjom od drugih kategorija. U nastavku ćemo prikazati najvažnije.

12.1.2.1. Dualna kategorija

Britansko-libanski matematičar Majkl Atija (Michael Francis Atiyah) rekao je: "Dualitet u matematici nije teorema, nego princip." U matematici, dualitet prevodi koncepte, teoreme ili matematičke strukture u druge koncepte, teoreme ili strukture u maniru jedan na jedan, često (ali ne uvek) pomoću operacije involucije[10]: ako je B dual od A, onda je A dual od B. Dualna kategorija je jedan od tih koncepata.

Kategorija C^{OP} je dualna kategoriji C ako su im objekti isti a morfizmi suprotni što znači da važi: ako je $f\colon \rightarrow B$ morfizam u C, tada je $f : B \rightarrow A$ morfizam u C^{OP} i obrnuto: ako je $f\colon \rightarrow B$ morfizam u

[10] U matematici, involucija/ involutivna funkcija je operacija/funkcija koja je samo-inverzna (involutivna funkcija je inverzija same sebe).

C^{OP}, tada je $f : B \to A$ morfizam u C. Kompozitni i identitetski morfizmi definišu se na očigledan način, samo se "obrnu" strelice morfizama.

Princip dualnosti ovde omogućuje da se definicije preformulišu u dualne definicije (u stvari, većina definicija se i javlja u "co-x" parovima) a i tvrđenja o kategorijama mogu se transformisati u dualna tvrđenja prostom zamenom reči „domen" i „kodomen" i zamenom kompozicije morfizama $g \circ f$ u $f \circ g$. Princip dualnosti u teoriji kategorija je osnova za "besplatne teoreme" o kategorijama: ako se teorema jednom dokaže, njen dual sledi direktno iz principa dualnosti.

12.1.2.2. Produkt

Još jedan način za pravljenje korisnih novih kategorija iz postojećih je produkt kategorija.

Za bilo koji par kategorija C i D, **produkt kategorija** $C \times D$ je kategorija koja za objekte ima parove (A, B) gde je A objekt kategorije C, B je objekat kategorije D a morfizmi su parovi (f, g) gde je f morfizam kategorije C a g morfizam kategorije D. Kompozicija i identitetski morfizam takođe se definišu kao parovi: $(f, g) \circ (h, i) = (f \circ h,\ g \circ i)$ i $id_{(A,B)} = (id_A,\ id_B)$ respektivno.

12.1.2.3. Kategorija morfizama

Ako sledimo definiciju kategorije, ništa nas ne sprečava da za objekte izaberemo morfizme. Dakle, mogu se konstruisati kategorije iz postojećih kategorija tako što će se morfizmi postojeće kategorije proglasiti objektima nove kategorije i definisati novi morfizmi tako da zadovolje zakone kategorije. Pokazaćemo kako to izgleda na kategoriji Set a kategorija morfizama za proizvoljnu kategoriju C dobija tako što se prosto zameni Set sa C.

Definisaćemo sada kategoriju $\boldsymbol{Set}^{\rightarrow}$ koja je kategorija morfizama kategorije $\boldsymbol{Set}$.

Objekti kategorije $Set^{\rightarrow}$ su, dakle, baš morfizmi u Set, odnosno svaki morfizam $f : A \to B$ iz kategorije Set je objekat u kategoriji $Set^{\rightarrow}$.

To znači da morfizam u $Set^{\rightarrow}$ ima morfizme iz Set za svoj domen i kodomen.

Morfizam u kategoriji $Set^{\rightarrow}$ koji preslikava iz $f : A \to B$ na $f' : A' \to B'$ je definisan kao par (a, b) morfizama iz Set gde je
$a : A \to A'$ i $b : B \to B'$ tako da je $f' \circ a = b \circ f$.

Kompozicija morfizama (a, b) i (a', b') iz $Set^{\rightarrow}$ kategorije gde je

$(a, b) : (f : A \to B) \to (f' : A' \to B')$ i
$(a', b') : (f' : A' \to B') \to (f'' : A'' \to B'')$

definiše se kao
$(a', b') \circ (a, b) = (a' \circ a,\ b' \circ b)$.

12.1.2.4. Podkategorija

Konačno, mogu se definisati i podkategorije kategorije C.

Definicija. Kategorija $\boldsymbol{B}$ je podktegorija kategorije $\boldsymbol{C}$ ako važi:

- Svaki objekat kategorije $\boldsymbol{B}$ je i objekat kategorije $\boldsymbol{C}$, i
- za sve objekte B i B' iz $\boldsymbol{B}$, važi $\boldsymbol{B}(B, B') \subseteq \boldsymbol{C}(B, B')$, i
- identitetski i kompozitni morfizmi su isti u $\boldsymbol{B}$ kao u **C**.

12.1.3. Dijagrami

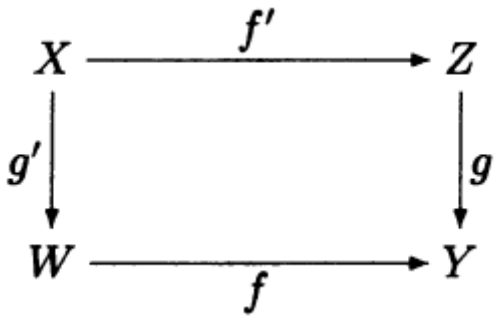
Da bi se čitljivost učinila boljom i u teoriji kategorija koriste se grafički simboli za reprezentaciju. U pitanju je jednostavan standardizovan grafički jezik koji omogućuje predstavljanje **u kategoriji**.

Definicija. Dijagram u kategoriji $\boldsymbol{C}$ je kolekcija temena i usmerenih ivica konzistentno označenih objektima i morfizmima iz $\boldsymbol{C}$, gde „konzistentno“ znači da ako je ivica u dijagramu označena morfizmom f i f ima domen A i kodomen B, onda krajnje tačke ove ivice moraju biti označene sa A i B.

Dijagrami se često koriste za izražavanje i dokazivanje svojstava kategorijskih konstrukcija. Takva svojstva se često mogu izraziti iskazom da je određeni dijagram komutativan (da komutira).

Definicija. Za dijagram u kategoriji $\boldsymbol{C}$ se kaže da je komutativan (komutira) ako su za svaki par temena X i Y sve putanje u dijagramu od X do Y jednake gde jednakost znači da svaka putanja u dijagramu određuje morfizam i da su ovi morfizmi jednaki u $\boldsymbol{C}$.

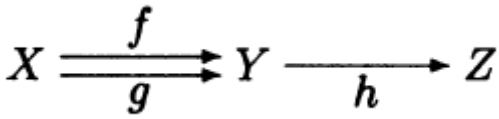
Na primer, iskaz da je dijagram na slici 17.2.3 komutativan je isto što i iskaz da važi $f \circ g' = g \circ f'$.



Slika 17.2.3 Komutativnost dijagrama je ekvivalentna iskazu $f \circ g' = g \circ f'$

Ako se malo udubite u sliku i iskaz $f \circ g' = g \circ f'$, videćete da to u stvari znači sledeće: i morfizam $f \circ g'$ i morfizam $g \circ f'$ preslikavaju objekat X u objekat Y “prolazeći” kroz dve različite “usputne stanice”: objekte W i Z respektivno. Ako biste na dijagram dodali dijagonalu od X do Y, ta dijagonala je kompozicija koja direktno preslikava objekat X u objekat Y. A to je bilo koji od morfizama $f \circ g'$, odnosno $g \circ f'$.

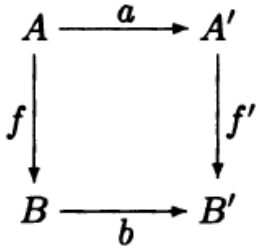
Jedna korisna rafinacija ove konvencije je da se za jednakost dve putanje zahteva da bar jedna od njih sadrži više od jednog morfizma. Primer je dijagram sa slike 17.2.4 koji kaže da važi $h \circ f = h \circ g$, ali ne kaže da važi $f = g$.



Slika 17.2.4 Ovde važi $h \circ f = h \circ g$, ali ne važi $f = g$

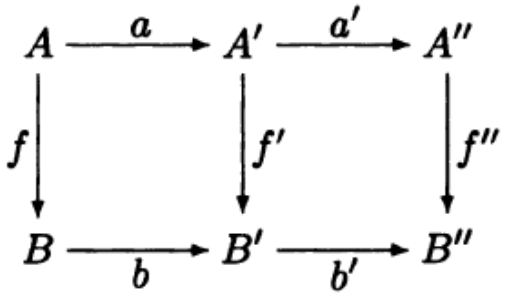
Primer kategorije $\boldsymbol{Set}^{\rightarrow}$ izražen putem dijagrama treba da ilustruje prednost korišćenja dijagrama.

Svaki morfizam $f : A \rightarrow B$ u kategoriji $\boldsymbol{Set}$ je objekat u kategoriji $\boldsymbol{Set}^{\rightarrow}$. Morfizam u kategoriji $\boldsymbol{Set}^{\rightarrow}$ iz $f : A \rightarrow B$ u $f' : A' \rightarrow B'$ je par (a, b) morfizama iz $\boldsymbol{Set}$ takav da je sledeći dijagram (slika 17.2.5) komutativan u $\boldsymbol{Set}$:



Slika 17.2.5 Morfizam u kategoriji $\boldsymbol{Set}^{\rightarrow}$ je komutativan dijagram

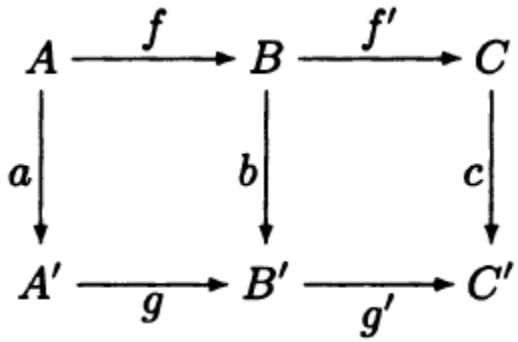
Kompozicija morfizama (a, b) i (a', b') iz $Set^{\rightarrow}$ kategorije gde je $(a, b) : (f: A \rightarrow B) \rightarrow (f': A' \rightarrow B')$ i $(a', b') : (f' : A' \rightarrow B') \rightarrow (f'' : A'' \rightarrow B'')$ je $(a', b') \circ (a, b) = (a' \circ a,\ b' \circ b)$ što odgovara "slepljivanju" dva dijagrama (slika 17.2.6).



Slika 17.2.6 Kompozicija morfizama u kategoriji $\boldsymbol{Set}^{\rightarrow}$ je komutativan dijagram

Kada se neko svojstvo izrazi putem komutativnih dijagrama, dokazi koji uključuju to svojstvo često se mogu dati „vizuelno". Sledeći jednostavan dokaz demonstrira tehniku: jednačine odgovaraju putanjama u dijagramu i te putanje se transformišu zamenom jedne putanje kroz komutativni pod-dijagram drugom putanjom.

Tvrđenje 17.2.1. Ako su oba unutrašnja kvadrata u sledećem dijagramu (Slika 17.2.7) komutativna, komutativan je i spoljašnji pravougaonik.

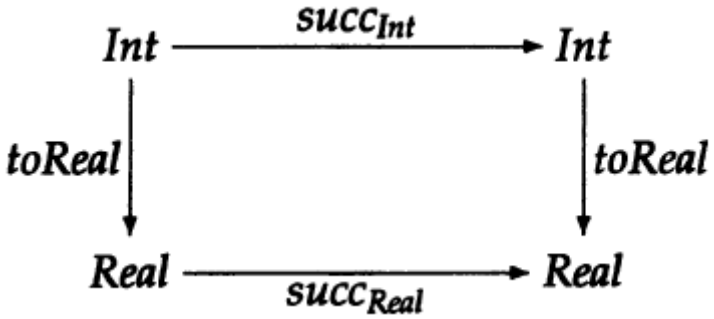


Slika 17.2.7 Tvrđenje 17.2.1 izraženo komutativnim dijagramima

Dokaz je jednostavan:

$(g' \circ g) \circ a = g' \circ (g \circ a)$	(asocijativnost)
$g' \circ (b \circ f)$	(komutativnost prvog kvadrata)
$(g' \circ b) \circ f$	(asocijativnost)
$(c \circ f') \circ f$	(komutativnost drugog kvadrata)
$= c \circ (f' \circ f)$	(asocijativnost).■

Na kraju, čisto da se stekne osećaj o korisnosti komutativnosti dijagrama u nečemu što je nalik praksi, pokazaćemo kako se komutativni dijagram može iskoristiti u jednostavnom funkcionalnom jeziku SFPL. Sledeći dijagram (slika 17.2.8) demonstrira utvrđivanje validnosti transformacija programa u kojima je redosled operacija permutovan. Komutativnost dijagrama sa slike ekvivalentna je tvrđenju da je rezultat izraza koji se dobija konverzijom celobrojne vrednosti u realnu i zatim sračunavanjem realnog sukcesora isti kao i rezultat koji se dobije ako se prvo sračuna celobrojni sukcesor i zatim se celobrojni rezultat konvertuje u realnu vrednost.



Slika 17.2.8 Transformacije sa permutovanim redosledom operacija su validne ako je ovaj dijagram u SFPL kategoriji komutativan

12.1.4. Monomorfizmi, epimorfizmi i izomorfizmi

Funkcija je vrlo tesno povezana sa morfizmom kategorije. Zbog toga nas zanimaju kategorijski analogoni funkcija sa posebnim svojstvima kao što su injektivnost (funkcija f koja preslikava različite elemente svog domena u različite elemente kodomena), surjektivnost (funkcija f takva da za svaki element y kodomena funkcije, postoji najmanje jedan element x u domenu funkcije tako da $f(x) = y$) i bijektivnost/ definisanje izomorfizma (funkcija f takva da je svaki element kodomena mapiran na tačno jedan element domena), jer ovi analogoni igraju važnu ulogu u kategorijskim rasuđivanjima.

Definicija 1. Morfizam $f : B \to C$ u kategoriji C je ***monomorfizam*** ("moničan je'') ako za svaki par morfizama u C važi sledeće:

Za $g : A \to B$ i $h: A \to B$, jednakost $f \circ g = f \circ h$ implicira da je $g = h$.

Tvrđenje 1. U kategoriji $\boldsymbol{Set}$, monomorfizmi su samo injektivne funkcije, odnosno funkcije f takve da $f(x) = f(y)$ implicira $x = y$.

Definicija 2. Morfizam $f : A \to B$ je ***epimorfizam*** ("epičan je") ako za bilo koji par morfizama $g : B \to C$ i $h : B \to C$, jednakost $g \circ f = h \circ f$ implicira da je $g = h$.

Tvrđenje 2. U kategoriji $\boldsymbol{Set}$, epimorfizmi su samo surjektivne funkcije, odnosno funkcije f takve da važi: za svako b ∈ B postoji $a \in A$ za koje je $f(a) = b$.

Iz zaista ogromnog skupa specijalnih morfizama, ovde ćemo još definisati ***izomorfizam***.

Definicija 3. Morfizam $f : A \to B$ je ***izomorfizam*** ako postoji morfizam $f^{-1} : B \to A$ koji se zove inverzija takav da je $f^{-1} \circ f = id_A$ i $f \circ f^{-1} = id_B$. Za objekte A i B kaže se da su izomorfni ako među njima postoji izomorfizam

Za dva izomorfna objekta često se kaže da su *identični do na izomorfizam*.

12.1.5. Univerzalne konstrukcije

Karakterizacije određenog tipa objekta u smislu njegovog odnosa sa ostatkom univerzuma naziva se ***univerzalna konstrukcija*** i veoma je uobičajena u teoriji kategorija. Univerzalna konstrukcija svodi se na sledeće: specificira se određeno svojstvo, a zatim se uspostavlja hijerarhiju objekata na bazi toga koliko dobro objekti modeliraju specificirano svojstvo. Zatim se bira „najbolji“ model pri čemu “najbolji“ može značiti različite stvari: najjednostavniji, najmanje ograničen, najmanji ili najveći, u zavisnosti od konteksta.

U matematici, tačnije u teoriji kategorija, univerzalno svojstvo je svojstvo koje karakteriše do izomorfizma rezultat nekih konstrukcija. Dakle, univerzalna svojstva mogu da se koriste za definisanje

nekih objekata nezavisno od izabranog metoda za njihovu konstrukciju. Na primer, definicije celih brojeva iz prirodnih brojeva, racionalnih brojeva iz celih brojeva, realnih brojeva iz racionalnih brojeva i polinomskih prstenova iz polja njihovih koeficijenata mogu se izvršiti putem univerzalnih svojstava. Koncept univerzalnog svojstva omogućava jednostavan dokaz da su sve konstrukcije izvršene putem univerzalnog svojstva ekvivalentne: dovoljno je dokazati da one zadovoljavaju isto univerzalno svojstvo.

Pojednostavljeno, moglo bi se kazati da je *univerzalna konstrukcija definicija do izomorfizma jedinstvenog objekta koji zadovoljava određeno univerzalno svojstvo*.

Dakle, univerzalna konstrukcija definiše više objekata koji su izomorfni. Na primer, ako za univerzalno svojstvo objekta stan usvojimo postojanje terase, svi stanovi sa terasom su predstavljeni jedinstvenim objektom koji zadovoljava univerzalno svojstvo da ima terasu. Naravno, pojedinačni objekti mogu da se razlikuju po drugim svojstvima: da li imaju ostavu, da li imaju kupatilo, itd. Svi stanovi koji imaju terasu su predstavljeni jedinstvenim objektom.

U teoriji kategorija inicijalni i terminalni objekti, produkti i ko-produkti, ekvilajzeri i ko-ekvilajzeri, prekompozicije i ko-prekompozicije su primeri univerzalnih konstrukcija koji su svi specifične instance opštijih koncepata ***limit*** i ***kolimit***.

U ovom izlaganju prikazaćemo primere univerzalnih konstruktora od specifičnijih ka opštijim. Izlaganje ćemo završiti predstavljanjem koncepta limit, odnosno kolimit.

12.1.5.1. Inicijalni i terminalni objekti

Hajde da pokušamo da koristimo univerzalnu konstrukciju da definišemo prazan skup a da ne pominjemo elemente skupa. Šta se može reći o funkcijama unutar i van praznog skupa?

Pre svega, ne možete imati funkciju koja mapira iz bilo kog nepraznog skupa u prazan skup. Funkcija koja bi mapirala iz nepraznog skupa u prazan skup imala bi neprazan skup za domen a prazan skup (znači skup bez i jednog elementa) za kodomen. U domenu, dakle, ima elemenata koje nekako treba da se mapiraju u elemente kodomena u kome NEMA NIČEGA NA ŠTA BI SE MOGLO MAPIRATI.

Nasuprot tome, lako je mapirati prazan skup u bilo koji drugi skup. To je jednostavno: prosto, NEĆE SE MAPIRATI NIŠTA, odnosno NEMA NIČEGA ŠTA BI SE MOGLO MAPIRATI. Dakle, prazan skup ima svojstvo da postoji jedinstveno preslikavanje iz njega u bilo koji drugi skup. Pored toga, ne postoji ni jedan drugi skup sa ovim svojstvom jer da postoji, postojalo bi mapiranje iz tog drugog skupa u prazan skup, a to je nemoguće.

Na taj način dobili smo u kategoriji skupova (kategorija čiji je objekat proizvoljan skup) definiciju praznog skupa u kojoj se ne pominju elementi skupa, već se pominju samo funkcije – prazan skup je skup koji ima jedinstveno preslikavanje na bilo koji drugi skup.

Definicija praznog skupa može se generalizovati na bilo koju kategoriju. Ova generalizacija uvodi koncept koji se i teoriji kategorija naziva ***inicijalni objekat***: Objekat koji ima **jedinstveno preslikavanje na bilo koji drugi objekat u kategoriji** naziva se inicijalni objekat. U kategoriji skupova, prazan objekat je jedini inicijalni objekat. U kategorijama koje nisu kategorije skupova može biti više inicijalnih objekata pri čemu su **svi inicijalni objekti izomorfni**.

Sledi formalna definicija inicijalnog objekta.

Definicija 4. Objekat $\mathbf{0}$ zove se **inicijalni objekat** ako za svaki objekat A postoji tačno jedan morfizam iz $\mathbf{0}$ na A.

Svaka konstrukcija u teoriji kategorija ima svoj dual: ono što se dobije invertovanjem smera svih morfizama (strelica). Dual inicijalnog objekta naziva se ***terminalni objekat***. To je objekat koji **ima jedinstveni morfizam koji dolazi iz bilo kog drugog objekta u kategoriji**.

U kategoriji skupova terminalni objekat je skup od jednog elementa. Jedinstveni morfizam (funkcija) iz bilo kog skupa (domena) u skup sa jednim elementom (kodomen) preslikava sve elemente domena u taj jedan element kodomena. To se zove **konstantna funkcija**. Dakle, ovde imamo univerzalnu definiciju skupa od jednog elementa kao terminalnog objekta u kategoriji skupova.

Sledi formalna definicija terminalnog objekta.

Definicija 5. Dualno, objekat $\mathbf{1}$ zove se **terminalni** (**finalni**) **objekat** ako za svaki objekat A postoji tačno jedan morfizam iz A na $\mathbf{1}$.

Koncept terminalnog objekta može se iskoristiti da se napravi kategorijski analogon elemenata skupova. Osnova za to je činjenica da su funkcije iz singltonskog skupa na skup S u kategoriji $\boldsymbol{Set}$ korespondencije jedan-na-jedan sa elementima u S. Štaviše, ako je x element iz S posmatran kao morfizam $x : 1 \to S$ iz nekog skupa 1 od jednog elementa, i f je funkcija iz S na neki drugi skup T, tada je element $f(x)$ jedinstven element iz T koji je slika kompozitne funkcije $f \circ x$.

U kategorijskoj terminologiji, morfizam od terminalnog objekta ka objektu S zove se **globalni element** ili **konstanta** od S.

12.1.5.2. Produkti

U teoriji kategorija, sve informacije o objektima su kodirane u morfizmima između objekata. Objekat u teoriji kategorija NEMA INTERNU STRUKTURU. Sva struktura objekta reflektuje se kroz odnose (morfizme) koje dati objekat ima sa drugim objektima u kategoriji, uključujući i samog sebe.

Ovo zahteva malo navikavanja, posebno kada se radi o poznatim objektima kao što su skupovi gde sve počinje i završava se sa pojmom elementa. Kako izraziti ideju da je skup prazan, ili da se sastoji od parova elemenata iz dva druga skupa, a da se ne spominju elementi?

S druge strane, ako se konstruišu određeni tipovi skupova isključivo putem njihovih odnosa, te konstrukcije se lako mogu primeniti na druge kategorije.

Uobičajena definicija kartezijanskog produkta dva skupa A i B u terminologiji teorije skupova je:

$$A \times B = \{(a, b) \mid a \in A \, i \, b \in B\}.$$

Ovde je cilj da se osmisli karakterizacija **produkata u kategoriji** (NE **produkata kategorija**) oslonjena na koncept morfizma[11]. Šta stvarno znači poslednja rečenica? Znači treba da se definiše kartezijanski skup bez pominjanja elemenata.

Prvo ćemo da razmotrimo koji su to morfizmi posebno značajni za proizvode skupova (napomena: nama su skupovi objekti u kategoriji).

Sastavni deo definicije proizvoda dva skupa A i B, definišu se i funkcije projekcije (proizvoda na pojedinačne skupove) $\mathrm{p}_1 : A \times B \to A$ i $\mathrm{p}_2 : A \times B \to B$. Dakle, možemo da pokušamo da produkt $A \times B$ definišemo kao skup zajedno sa dve funkcije, od kojih jedna mapira na A a druga na B. Ono što je tu problem je što **takvih skupova ima beskonačno mnogo**, a neki od njih ni ne liče na produkt.

Ispostavlja se da, da bi se okarakterisao skup kao proizvod $A \times B$, mora da se pogleda njegov odnos sa svakim drugim skupom koji ima preslikavanja u A i B. Kroz utvrđivanje tog odnosa treba da se vidi da li se skup ističe kao model za svoj odnos sa A i B. Pošto su projekcije ono što opisuje taj odnos, skup

[11] Koristeći zapažanje da se elementi mogu tretirati kao morfizmi od terminalnog objekta, mogla bi se definisati konstrukcija proizvoda u kategoriji putem globalnih elemenata. Međutim, to bi bilo kontradiktorno sa stilom teorije kategorija koja tretira objekte kao entitete bez unutrašnje strukture i sve što se zna o pojedinačnom objektu izražava njegovim morfizmima, odnosno vezama sa drugim objektima.

koji se ističe kao model je onaj koji ima „najbolje“ projekcije. U stvari, njegove projekcije su toliko dobre da bilo koji drugi par projekcija iz bilo kog drugog skupa mora da “prođe“ kroz njih.

Kako se može tvrditi da je jedan kandidat za proizvod bolji od drugog? Neka su data dva skupa X i Y i njihova mapiranja na A i B:

$$p_1 : X \to A$$
$$p_2 : X \to B$$
$$q_1 : Y \to A$$
$$q_2 : Y \to B$$

Želimo da ta dva skupa na neki način dovedemo u vezu pa ćemo da pretpostavimo da postoji funkcija +,+++(morfizam) h od Y na X. Kako je h u vezi sa projekcijom, na nju se postavljaju uslovi:

$$q_1 = p_1 \circ h$$
$$q_2 = p_2 \circ h$$

Iz postavljenih uslova sledi da q_1 i q_2 imaju zajednički „faktor“ – funkciju h. Kažemo da se funkcije q_1 i q_2 „faktorizuju“ putem funkcije h. Ova faktorizacija odgovara terminu “prolaženje“.

Naravno, funkcija h ne mora da postoji pa poređenje ne mora da bude moguće. Ako “ima sreće“, **funkcija h će da postoji** i to će da bude **jedinstvena funkcija** i tada kažemo da je **skup X bolji kandidat za produkt nego skup Y**.

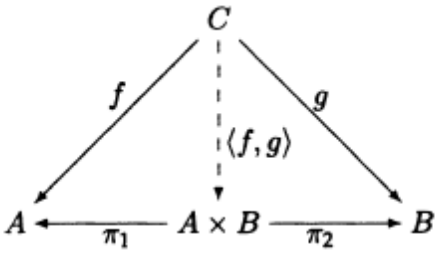
Na kraju, možemo da formulišemo univerzalnu konstrukciji za produkt u kategoriji.

X je produkt od A i B ako i samo ako:

1. Postoji par morfizama
   $p_1 : X \to A$
   $p_2 : X \to B$
2. Za svaki drugi objekat Y sa parom morfizama
   $q_1 : Y \to A$
   $q_2 : Y \to B$
3. Postoji jedinstven morfizam h (*medijatorski morfizam*) takav da je
   $q_1 = p_1 \circ h$
   $q_2 = p_2 \circ h$

Konačno, formalna definicija **produkta u kategoriji** je sledeća [1].

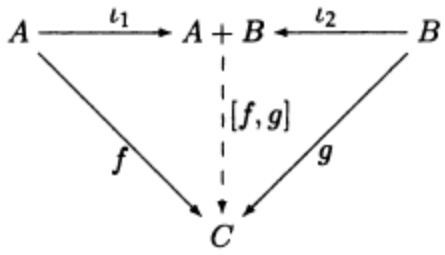
Definicija 6. Produkt dva objekta A i B je objekat $A \times B$ zajedno sa dva projekciona morfizma $\pi_1 : A \times B \to A$ i $\pi_2 : A \times B \to B$, tako da za svaki objekat C i par morfizama $f : C \to A$ i $g : C \to B$ postoji tačno jedan medijatorski morfizam $\langle f, g \rangle : C \to A \times B$ za koji je dijagram na slici 17.2.9 komutativan, što je ekvivalentno iskazu $\pi_1 \circ \langle f, g \rangle = f$ i $\pi_2 \circ \langle f, g \rangle = g$.



Slika 17.2.9 Komutativni dijagram produkta objekata A i B[12]

Dualni koncept je **koprodukt** koji se definiše na sledeći način.

Definicija 7. Koprodukt dva objekta A i B je objekat $A + B$ zajedno sa dva projekciona morfizma $\tau_1 : A \to A + B$ i $\tau_2 : B \to A + B$, tako da za svaki objekat C i par morfizama $f : A \to C$ i $g : B \to C$ postoji tačno jedan medijatorski morfizam $[f, g] : A + B \to C$ za koji je dijagram na slici 17.2.10 komutativan.

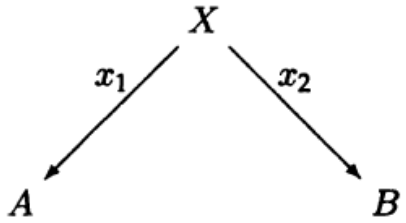


Slika 17.2.10 Komutativni dijagram koprodukta objekata A i B

12.1.5.3. Generalna karakterizacija univerzalnih konstrukcija

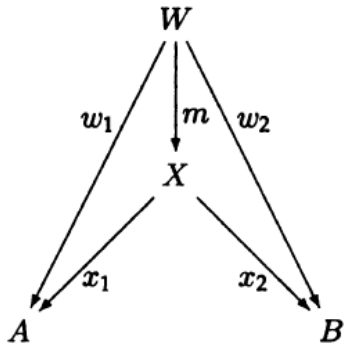
Univerzalna konstrukcija u teoriji kategorija opisuje klasu objekata i pridruženih morfizama takvih da imaju zajedničko svojstvo i izdvaja objekte koji su terminalni kada je ta klasa objekata kategorija.

Na primer, definicija produkta objekata A i B u $\boldsymbol{C}$ opisuje **klasu** torki (X, x_1, x_2), gde je $x_1 : X \to A$ i $x_2 : X \to B$ kao na dijagramu sa slike 17.2.11:



Slika 17.2.11 Univerzalna konstrukcija produkata objekata

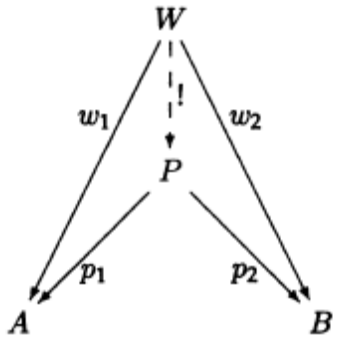
Izvor [1] naziva te torke "klinovi nad A i B," objekti "kategorije klinova nad A i B." Morfizam u toj kategoriji , recimo $m : (W, w_1, w_2) \to (X, x_1, x_2)$, je $\boldsymbol{C}$-morfizam $m : W \to X$ takav da je $x_1 = w_1 \circ m$ i $x_2 = w_2 \circ m$ (slika 17.2.12):



Slika 17.2.12 torke kategorije klinova

[12] Iscrtkane strelice (morfizmi) na komutativnom dijagramu koriste se da predstave morfizme za koje se tvrdi da postoje ako je ostatak dijagrama prikladno popunjen.

Terminalni objekat u kategoriji klinova (slika 17.2.13), recimo (P, p_1, p_2), je jedan sa jedinstvenim morfizmom koji vodi do njega iz svakog klina:



Slika 17.2.13 Terminalni objekat u kategoriji klinova

Objekat P se obično označava $A \times B$; morfizmi p_1 i p_2 se označavaju sa π_1 i π_2. Jedinstveni morfizam od klina (W, w_1, w_2) do $(A \times B, \pi_1, \pi_2)$, čija je egzistencija zagarantovana činjenicom da je $(A \times B, \pi_1, \pi_2)$ terminalni objekat, zapisuje se kao (w_1, w_2).

Za entitete definisane univerzalnom konstrukcijom kaže se da su **univerzalni među entitetima koji zadovoljavaju dato svojstvo**, ili jednostavno **da imaju univerzalno svojstvo**. Jedinstveni morfizmi na njih sa drugih objekata koji dele dato svojstv često se nazivaju **medijatorski morfizmi**.

Ko-univerzalna konstrukcija ima isti oblik kao i univerzalna konstrukcija, osim što su **morfizmi (strelice) obrnuti** i ona **izdvaja inicijalni objekat** sa datim svojstvom.

12.1.5.4. Ekvilajzeri

https://www.mql5.com/en/articles/12417 - primene u finansijskom trgovanju

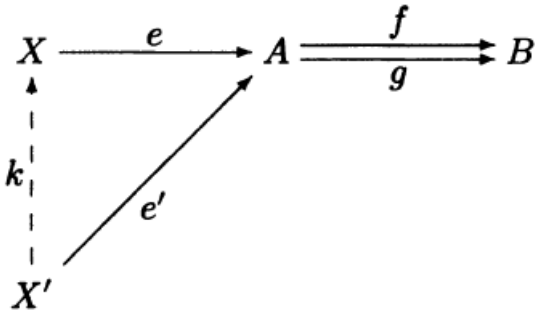
Još jedna osnovna univerzalna konstrukcija je ekvilajzer dva morfizma.

Definicija 8. Morfizam $e : X \to A$ je ekvilajzer para morfizama $f: A \to B$ i $g: A \to B$ ako važi:

1. $f \circ e = g \circ e$;

2. Ukoliko $e' : X' \to A$ zadovoljava $f \circ e' = g \circ e'$, postoji jedinstven morfizam $k: X' \to X$ takav da je $e \circ k = e'$.

Na slici 17.2.14 prikazan je dijagram ekvilajzera.



Slika 17.2.14 Dijagram ekvilajzera

Iz definicije se lako čita da je ekvilajzer univerzalni konstrukt koji se bavi morfizmima sa istim domenom i istim kodomenom. U definiciji, obe funkcije f i g imaju domen A i kodomen B.

Kategorija **Set** će da nam posluži za intuitivnu ilustraciju konstrukcije ekvilajzer. Neka su f i g dve funkcije u kategoriji Set sa istim domenom A i kodomenom B i neka je X podskup od A nad kojim su f i g jednake:

$$X = \{x | x \in A, f(x) = g(x)\}$$

Ovde je ekvilajzer morfizama f i g **funkcija inkluzije** $e : X \to A$ koja mapira svaki element $x \in X$ na isto x posmatrano kao element skupa A.

Napomena: Ako je A podskup od B, ***funkcija inkluzije*** je funkcija ι koja mapira svaki element x iz A na x posmatran kao element od B.

$\iota: \mathrm{A} \to \mathrm{B}$, $\iota(x) = x$

Dualna konstrukcija je **ko-ekvilajzer**. Ona obezbeđuje kategorijalnu analogiju skupovskom konceptu **relacije ekvivalencije**.

12.1.5.5. Povlačenje i izvlačenje

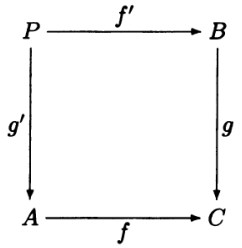
Još jedan koristan kategorijski konstrukt je ***povlačenje*** parova morfizama, sa svojim dualom zvanim ***izvlačenje***.

Definicija 9. *Povlačenje* (*pullback*) para morfizama $f : A \to C$ i $g : B \to C$ je objekat P i par morfizama $g' : P \to A$ i $f' : P \to B$ takvih da je:

(1) $f \circ g' = g \circ f'$, i
(2) ako su $i : X \to A$ i $j : X \to B$ takvi da je $f \circ i = g \circ j$, tada postoji jedinstven morfizam $k : X \to P$ takav da je $i = g' \circ k$ i $j = f' \circ k$.

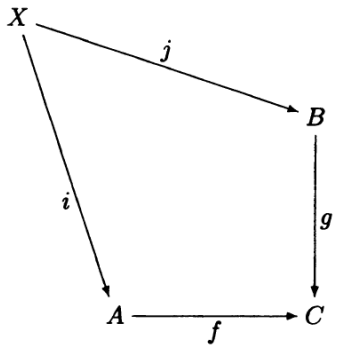
Da bismo možda i razumeli nešto od ove definicije, poslužićemo se dijagramima.

Prvi iskaz: $f \circ g' = g \circ f'$ može se dijagramom predstaviti na sledeći način (slika 7.2.15):



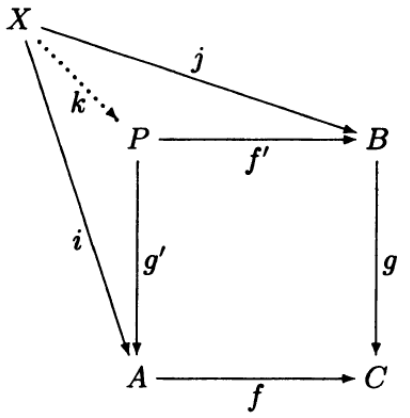
Slika 7.2.15 Pull back – objekat P

Uslov iz iskaza (2) u definiciji je sledeći dijagram (slika 7.2.16):



Slika 7.2.16 Pull back – objekat X

Kada dodamo i posledica iz iskaza (2), dobijamo definiciju povlačenja u obliku dijagrama na slici 7.2.17:



Slika 7.2.17 Pull back definicija

Uobičajeno je da se kaže da je f' povlačenje (ili inverzna slika) morfizma f duž g i da je g' povlačenje morfizma g duž f.

Primer koji obrazlaže ovu terminologiju opet dolazi iz kategorije **Set**:

12.1.5.6. Limiti

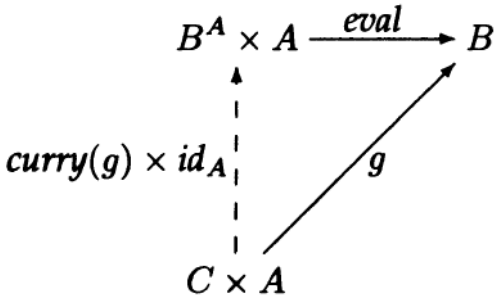
12.1.5.7. Stepenovanje

Bazni univerzalni konstrukt zvani ***stepenovanje*** omogućuje teorijski jednu od ključnih stvari u funkcionalnom programiranju – *kuriranje*.

Za početak, prikazaćemo skicu konstrukcije za kategoriju **Set** preuzetu iz [1].

Ako su A i B skupovi, kolekcija $B^A = \{f : A \to B\}$ svih funkcija sa domenom A i kodomenom B je takođe skup. Slično, u nekim drugim kategorijama $\boldsymbol{C}$ može se $\boldsymbol{C}(A, B)$ predstaviti kao objekat B^{A} koji je iz $\boldsymbol{C}$. [13]

Ovakva definicija nama ne koristi jer mi ne želimo da pojam B^{A} definišemo putem elemenata skupa – hoćemo da ga definišemo putem morfizama objekta u kategoriji. Polazna tačka je zapažanje da se entitetu B^{A} može pridružiti specijalna evaluaciona funkcija $eval: (B^A \times A) \to B$ definisana na sledeći način: $eval(f, a) = f(a)$. Dakle, sa $f : A \to B$ i $a \in A$, funkcija $eval$ za ulaz (f, a) vraća kao rezultat $f(a) \in B$. Kategorijalna definicija pojma B^A „kači se“ na zapažanje da $eval$ poseduje univerzalno svojstvo svih funkcija $g : (C \times A) \to B$ koje kaže: Za $g : (C \times A) \to B$ **postoji tačno jedna funkcija** $curry(g) : C \to B^A$ takva da je sledeći dijagram (slika 7.2.18) komutativan:



slika 7.2.18 Stepenovanje –kuriranje

Ovde $curry(g) \times id_A$ označava produkt koji za ulaz (c, a) daje $(curry(g)(c), a)$.

[13] Ovo ne važi za sve kategorije – na primer, nema analogne konstrukcije u kategoriji **Mon**.

Bilo koje određeno $c \in \boldsymbol{C}$ određuje funkciju u B^A tako što fiksira prvi argument funkcije g na c, a drugi argument ostavlja slobodnim. Neka je definisana funkcija g_c kao kurirana verzija funkcije g za određeno $c \in \boldsymbol{C}$:

$g_c(a) = g(c, a)$.

Tada je $curry(g)$ samo funkcija koja prosleđuje svako c odgovarajućoj kuriranoj verziji funkcije g:

$$curry(g)(c) = g_c$$

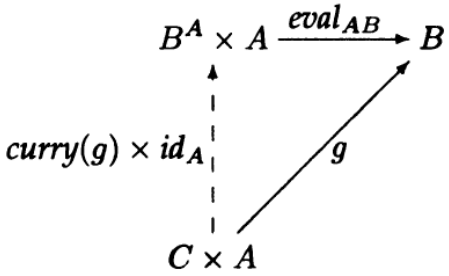
Sada se za svako $(c, a) \in \boldsymbol{C} \times A$, dobija:

$$\begin{aligned}(eval \circ (curry(g) \times id_A))(c, a) &= eval(curry(g)(c), a) \\ &= eval(g_c, a) \\ &= g_c(a) \\ &= g(c, a)\end{aligned}$$

Na taj način je pokazano da je dijagram sa funkcijom $curry(g)$ komutativan. To je i jedina funkcija koja ovaj dijagram čini komutativnim (dokaz je dat i u [1]).

Sledi formalna definicija eksponencijalnog objekta.

Definicija 11. Neka je $\boldsymbol{C}$ kategorija sa svim binarnim produktima i neka su A i B objekti u $\boldsymbol{C}$. Objekat B^A je eksponencijalni objekat ako postoji morfizam $eval_{AB} : (B^A \times A) \to B$ takav da za svaki objekat C i morfizam $g: (C \times A) \to B$ postoji jedinstven morfizam $curry(g): C - + B^A$ takav da je $evalAB \circ (curry(g) \times idA) = g$, odnosno jedinstven morfizam $curry(g): C \to B^A$ koji dijagram sa slike 7.2.19 čini komutativanim.



slike 7.2.20 eksponencijalni objekat

Ako $\boldsymbol{C}$ ima stepen B^A za svaki par objekata A i B, kaže se da $\boldsymbol{C}$ **ima stepenovanje**.

Kategorije sa stepenima i produktima za sve parove objekata su toliko važne da imaju posebno ime – **kartezijanski zatvorena kategorija** (*cartesian closed category* - $\boldsymbol{CCC}$).

$\boldsymbol{CCC}$ je kategorija sa terminalnim objektom, binarnim produktom i stepenovanjem.

12.1.6. Funktori, prirodne transformacije i adjunkti

Literatura uz Poglavlje 12

1. Benjamin C. Pierce, Basic Category Theory for Computer Scientists, The MIT Press, 1991.