Generalizirani kalkulator

u Haskellu

Mauro Raguzin

PMF-MO

10. srpnja 2024.

Sadržaj

O ideji ovog kalkulatora

U ovom smo se projektu uglavnom bavili zadnjim pitanjem, s tim da smo jasno fiksirali mjeru preciznosti i uvijek je stavili pod kontrolom korisnika našeg „proširenog” kalkulatora.

O ideji ovog kalkulatora

U ovom smo se projektu uglavnom bavili zadnjim pitanjem, s tim da smo jasno fiksirali mjeru preciznosti i uvijek je stavili pod kontrolom korisnika našeg „proširenog” kalkulatora.
Neki od ciljeva projekta su bili:

- primijeniti razne rezultate iz kolegija Izračunljiva analiza,
- konstruirati kalkulator u Haskellu,
- proširiti standardne funkcionalnosti onima koje omogućuju računanje s proizvoljnim Haskellovim funkcijama nad brojevima.

O ideji ovog kalkulatora

U ovom smo se projektu uglavnom bavili zadnjim pitanjem, s tim da smo jasno fiksirali mjeru preciznosti i uvijek je stavili pod kontrolom korisnika našeg „proširenog” kalkulatora.
Neki od ciljeva projekta su bili:

- primijeniti razne rezultate iz kolegija Izračunljiva analiza,
- konstruirati kalkulator u Haskellu,
- proširiti standardne funkcionalnosti onima koje omogućuju računanje s proizvoljnim Haskellovim funkcijama nad brojevima.

Kako se često u uvodnim tekstovima o Haskellu zna napomenuti da se interaktivna GHC sesija može koristiti kao kalkulator, ovdje to zaista želimo ojačati i proširiti na funkcije — nije moguće računati samo s ugrađenim Haskellovim brojevnim tipovima, već i s funkcijama nad tim tipovima!

Bitne funkcionalnosti kalkulatora

Konkretno, naš kalkulator podržava:

- računanje (obavljanje svih standardnih aritmetičkih operacija $+, -, *, /, |\cdot|$, no ovisi o konkretnoj algebarskoj strukturi) s prirodnim, cijelim, racionalnim i *izračunljivim*[1] realnim brojevima;
- sve aritmetičke operacije nad funkcijama oblika $\mathbb{N}^k \to D$, gdje je $D \in \{\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathrm{C}\mathbb{R}\}$ i $k \in \mathbb{N}_+$.

U slučaju rada s (implicitno izračunljivim) realnim brojevima, korisniku dajemo kontrolu nad određivanjem preciznosti u smislu standardne euklidske metrike nad $\mathbb{R}$ — preciznost je uvijek argument koji se mora specificirati pri računu realnog broja.

[1] Skup izračunljivih realnih brojeva označavamo sa $\mathrm{C}\mathbb{R} \subseteq \mathbb{R}$.

Naputci za implementaciju

Pri implementaciji kalkulatora u Haskellu nastojali smo se pridržavati sljedećih pravila:

Naputci za implementaciju

Pri implementaciji kalkulatora u Haskellu nastojali smo se pridržavati sljedećih pravila:

- koristiti samo čisti Haskell sa standardnim Prelude-om;

Naputci za implementaciju

Pri implementaciji kalkulatora u Haskellu nastojali smo se pridržavati sljedećih pravila:

- koristiti samo čisti Haskell sa standardnim Prelude-om;
- definirati sve bitnije funkcije za osnovni račun s brojevnim funkcijama s Izračunljive analize i pokušati se što više držati definicija s tog kolegija;

Naputci za implementaciju

Pri implementaciji kalkulatora u Haskellu nastojali smo se pridržavati sljedećih pravila:

- koristiti samo čisti Haskell sa standardnim Prelude-om;
- definirati sve bitnije funkcije za osnovni račun s brojevnim funkcijama s Izračunljive analize i pokušati se što više držati definicija s tog kolegija;
- postići što veću razinu formalne sličnosti definicija relevantnih funkcija u raznim dokazima rezultata iz IA i njihovih haskellovskih implementacija;

Naputci za implementaciju

Pri implementaciji kalkulatora u Haskellu nastojali smo se pridržavati sljedećih pravila:

- koristiti samo čisti Haskell sa standardnim Prelude-om;
- definirati sve bitnije funkcije za osnovni račun s brojevnim funkcijama s Izračunljive analize i pokušati se što više držati definicija s tog kolegija;
- postići što veću razinu formalne sličnosti definicija relevantnih funkcija u raznim dokazima rezultata iz IA i njihovih haskellovskih implementacija;
- uklopiti definicije funkcija u odgovarajuće algebarske strukture;

Naputci za implementaciju

Pri implementaciji kalkulatora u Haskellu nastojali smo se pridržavati sljedećih pravila:

- koristiti samo čisti Haskell sa standardnim Prelude-om;
- definirati sve bitnije funkcije za osnovni račun s brojevnim funkcijama s Izračunljive analize i pokušati se što više držati definicija s tog kolegija;
- postići što veću razinu formalne sličnosti definicija relevantnih funkcija u raznim dokazima rezultata iz IA i njihovih haskellovskih implementacija;
- uklopiti definicije funkcija u odgovarajuće algebarske strukture;
- reprezentirati algebarske strukture smislenim type-klasama, pri čemu treba težiti instanciranju već postojećih Prelude klasa, čime također dobivamo mogućnost korištenja standardnih infiksnih operatora u radu s našim brojevnim funkcijama (à la stolni kalkulatori)

Čime se *nismo* bavili

Važno je napomenuti i što ovaj kalkulator definitivno nije. Naime, on nije kalkulator koji

Čime se *nismo* bavili

Važno je napomenuti i što ovaj kalkulator definitivno nije. Naime, on nije kalkulator koji

- može izračunati sve realne brojeve; smatra se da takvo što nije fizički ni teorijski moguće sa sadašnjim razumijevanjem izračunljivosti [2];

Čime se *nismo* bavili

Važno je napomenuti i što ovaj kalkulator definitivno nije. Naime, on nije kalkulator koji

- može izračunati sve realne brojeve; smatra se da takvo što nije fizički ni teorijski moguće sa sadašnjim razumijevanjem izračunljivosti [2];
- koristi posebno odabrane, efikasne algoritme za svoje računske operacije;

Čime se *nismo* bavili

Važno je napomenuti i što ovaj kalkulator definitivno nije. Naime, on nije kalkulator koji

- može izračunati sve realne brojeve; smatra se da takvo što nije fizički ni teorijski moguće sa sadašnjim razumijevanjem izračunljivosti [2];
- koristi posebno odabrane, efikasne algoritme za svoje računske operacije;
- može računati korijene polinoma, rješavati sustave jednadžbi. . .

Čime se *nismo* bavili

Važno je napomenuti i što ovaj kalkulator definitivno nije. Naime, on nije kalkulator koji

- može izračunati sve realne brojeve; smatra se da takvo što nije fizički ni teorijski moguće sa sadašnjim razumijevanjem izračunljivosti [2];
- koristi posebno odabrane, efikasne algoritme za svoje računske operacije;
- može računati korijene polinoma, rješavati sustave jednadžbi. . .
- garantira terminaciju svih svojih izračunavanja.

Čime se *nismo* bavili

Važno je napomenuti i što ovaj kalkulator definitivno nije. Naime, on nije kalkulator koji

- može izračunati sve realne brojeve; smatra se da takvo što nije fizički ni teorijski moguće sa sadašnjim razumijevanjem izračunljivosti [2];
- koristi posebno odabrane, efikasne algoritme za svoje računske operacije;
- može računati korijene polinoma, rješavati sustave jednadžbi. . .
- garantira terminaciju svih svojih izračunavanja.

U vezi s posljednjom točkom, napomenimo da bi za stolni tip kalkulatora bilo idealno osigurati terminaciju svih operacija; one koje su ilegalne bi se trebale detektirati te bi se korisniku ispisala greška.

Čime se *nismo* bavili

Važno je napomenuti i što ovaj kalkulator definitivno nije. Naime, on nije kalkulator koji

- može izračunati sve realne brojeve; smatra se da takvo što nije fizički ni teorijski moguće sa sadašnjim razumijevanjem izračunljivosti [2];
- koristi posebno odabrane, efikasne algoritme za svoje računske operacije;
- može računati korijene polinoma, rješavati sustave jednadžbi...
- garantira terminaciju svih svojih izračunavanja.

U vezi s posljednjom točkom, napomenimo da bi za stolni tip kalkulatora bilo idealno osigurati terminaciju svih operacija; one koje su ilegalne bi se trebale detektirati te bi se korisniku ispisala greška. No, kako je naš implementacijski jezik Haskell, te želimo nešto općenitiji kalkulator koji radi s korisnički isprogramiranim funkcijama te je Haskell Turing-potpun jezik, smatramo da nema smisla korisniku davati neke posebne konstrukte koje ga ograničavaju u izražajnosti.

Ograničenja

Napomena

Terminacija izračunavanja funkcije u Haskellu, za dani ulaz, dakako nije odlučivo svojstvo i zato bi jedini način forsiranja terminacije bio uvođenje posebnog pod-jezika koji ne bi mogao biti Turing-potpun, što bi nas vratilo na razinu primitivnih kalkulatora koju želimo nadići.

Ograničenja

Napomena

Terminacija izračunavanja funkcije u Haskellu, za dani ulaz, dakako nije odlučivo svojstvo i zato bi jedini način forsiranja terminacije bio uvođenje posebnog pod-jezika koji ne bi mogao biti Turing-potpun, što bi nas vratilo na razinu primitivnih kalkulatora koju želimo nadići.

Dakle, dozvoljavamo da korisnik završi s parcijalnom funkcijom prilikom čijeg izračunavanja na našem kalkulatoru može doći do divergencije izračunavanja, što u Haskellu znači da funkcija vraća tip $\perp$.

Ograničenja

Napomena

Terminacija izračunavanja funkcije u Haskellu, za dani ulaz, dakako nije odlučivo svojstvo i zato bi jedini način forsiranja terminacije bio uvođenje posebnog pod-jezika koji ne bi mogao biti Turing-potpun, što bi nas vratilo na razinu primitivnih kalkulatora koju želimo nadići.

Dakle, dozvoljavamo da korisnik završi s parcijalnom funkcijom prilikom čijeg izračunavanja na našem kalkulatoru može doći do divergencije izračunavanja, što u Haskellu znači da funkcija vraća tip $\perp$.

Pažnja

Kako se na IA gotovo uvijek pretpostavlja rad s rekurzivnim funkcijama, što bi u Haskellu odgovaralo totalnim funkcijama — onima koje ne mogu divergirati — tvrdnje s tog kolegija u našem kalkulatoru vrijede samo ako korisnik zna da operira nad funkcijama koje su same totalne.

Ograničenja (nastavak)

Kako totalnost funkcije nije općenito moguće odlučiti u bilo kojem programskom jeziku, moramo se pouzdati u korisnika ili u njegovo korištenje posebnih konstrukcija u svom kodu kojima garantira da sve tako dobivene funkcije jesu totalne.

Osnovno o izračunljivosti brojevnih funkcija

Na kolegiju Izračunljivost smo primarno promatrali izračunljivost u okviru modela parcijalno rekurzivnih funkcija oblika

$$\mathbb{N}^k \to \mathbb{N},$$

za pozitivnu mjesnost k. Na to smo se oslonili i na Izračunljivoj analizi, s tim da smo tamo dali formalna proširenja tog modela izračunljivosti na funkcije oblika

$$\mathbb{N}^k \to \mathbb{Z}, \qquad \mathbb{N}^k \to \mathbb{Q}, \qquad \mathbb{N}^k \to \mathbb{R},$$

u pravilu pretpostavljajući u rezultatima da radimo s takvim funkcijama koje su ujedno i rekurzivne tj. totalne i izračunljive.

U čemu je problem?

- U Haskellu, kao ni u praktički bilo kojem programskom jeziku, nije pretjerano teško implementirati računanje s funkcijama svih spomenutih oblika, $\mathbb{N}^k \to \mathbb{R}$
- Taj zadnji oblik zahtijeva malo drukčiji pristup, budući da on, za razliku od svih ostalih navedenih, obuhvaća funkcije koje preslikavaju prirodne brojeve — koji se mogu konačno reprezentirati na računalu — u realne brojeve, koji se tako ne mogu reprezentirati
- Ipak, to nas ne spriječava da računamo proizvoljno točne aproksimacije odgovarajućih realnih brojeva — izračunljive Cauchyjeve nizove (racionalnih brojeva) koji jako brzo konvergiraju ka realnim brojevima.

Izračunljivi realni brojevi CR

Tako dolazimo do centralne definicije iz Izračunljive analize.

Definicija

Realan broj $\alpha \in \mathbb{R}$ nazivamo izračunljivim realnim brojem ako postoji rekurzivna funkcija $F : \mathbb{N} \to \mathbb{Q}$ takva da vrijedi

$$|\alpha - F(k)| < 2^{-k},$$

za sve $k \in \mathbb{N}$.

k koristimo kao garantiranu preciznost aproksimacije koju dobivamo preko rekurzivne (u Haskellu, totalne) funkcije F. Na ovoj se definiciji, i mnogim rezultatima iz IA, temelji naša implementacija izračunavanja s (izračunljivim) realnim brojevima.

O prezentaciji kalkulatora

- U nastavku ćemo proći kroz sve rezultate iz IA koji su relevantni za ovaj projekt
- Usporednim prikazom slike originalnih materijala iz IA i slike izvornog koda kalkulatora ćemo vidjeti kako su određeni matematički konstrukti implementirani u Haskellu
- Definicije tipova $\mathbb{Z}$ i $\mathbb{Q}$ u Haskellu nisu dane, jer se koriste posebno optimizirane interne GHC-ove implementacije tih tipova (`Integer` odnosno `Rational`)
- Iako ne otkrivamo ništa novo pokazivanjem kako se računa s funkcijama oblika, $\mathbb{N}^k \to \mathbb{Z}$ i $\mathbb{N}^k \to \mathbb{Q}$, ispada da se implementacija operacija nad takvim funkcijama može definirati na sasvim uniforman način, čak i kad se uzmu u obzir i funkcije oblika $\mathbb{N}^k \to \mathrm{CR}$.

Prirodni brojevi

Prvo i osnovno, treba nam tip koji reprezentira $\mathbb{N}$. Haskell ga nema definiranog u Prelude-u, pa ćemo napraviti jednostavnu rekurzivnu definiciju i instancirati sve klase čija svojstva $\mathbb{N}$ zadovoljava. Primijetimo da ćemo pritom ipak dozvoliti da tip `Nat` ima funkciju `negate` za negaciju, iako ta operacija nema smisla, jer želimo iskoristiti Haskellovu predefiniranu `Num` klasu koja nam omogućuje korištenje standardnih infiksnih operatora za aritmetičke operacije. Jedini operator koji ovdje nije definiran je onaj za inverz tj. recipročnu vrijednost broja `recip`; on se nalazi u klasi `Fractional` koja modelira tipove brojeva koji zadovoljavaju aksiome polja. To dakle ostavljamo za $\mathbb{Q}$ i CR.

Definicija Nat i pripadnih instanci

```haskell
data Nat =
    Zero
  | Succ Nat
  deriving (Eq, Ord, Show)
```

```haskell
one = Succ Zero

instance Num Nat where
    a + Zero = a
    a + (Succ b) = Succ (a + b)

    a - Zero = a
    a - (Succ b) = pred (a - b) -- modificirano oduzimanje

    a * Zero = Zero
    a * (Succ b) = a * b + a

    abs n = n

    signum Zero = Zero
    signum _    = one

    fromInteger n | n < 0  = undefined
                  | n == 0 = Zero
                  | n > 0  = Succ $ fromInteger (pred n)

    negate n = n
```

```haskell
instance Enum Nat where
    toEnum 0 = Zero
    toEnum n | n < 0      = undefined
             | otherwise = Succ $ toEnum (pred n)

    fromEnum Zero     = 0
    fromEnum (Succ n) = succ $ fromEnum n

    succ n = Succ n

    pred Zero     = Zero
    pred (Succ n) = n

instance Integral Nat where
    toInteger Zero     = 0
    toInteger (Succ n) = succ $ toInteger n

    quotRem x Zero = (x, Zero)
    quotRem x y = sub x y Zero
      where sub x y c | y > x = (c, x)
                      | otherwise = sub (x - y) y (succ c)

    divMod = quotRem
```

$$|x-y| = (x \dot{-} y) + (y \dot{-} x),\ x, y \in \mathbb{N}$$

```haskell
instance Real Nat where
    toRational n = toRational $ fromEnum n

-- apsolutna razlika
absDiff :: Nat -> Nat -> Nat
absDiff x y = (x - y) + (y - x)

-- polimorfno potenciranje
pow :: Num a => a -> Nat -> a
pow _ 0        = 1
pow x (Succ n) = x * pow x n

-- sg potez
sgc :: Nat -> Nat
sgc n = 1 - signum n
```

$$\mathrm{sg}(n) = \begin{cases} 0, & \text{ako je } n = 0; \\ 1, & \text{inače.} \end{cases}$$

$$\overline{\mathrm{sg}}(n) = \begin{cases} 1, & \text{ako je } n = 0; \\ 0, & \text{inače.} \end{cases}$$

```haskell
data Fun1 a where
    Fun1 :: (Nat -> a) -> Fun1 a

data Fun2 a where
    Fun2 :: (Nat -> Nat -> a) -> Fun2 a

data Fun3 a where
    Fun3 :: (Nat -> Nat -> Nat -> a) -> Fun3 a

-- ograničena podrška za vektorske funkcije
-- kodomena može biti N^k; mi idemo do k=2
type Fun12 a = Fun1 (a,a)
type Fun22 a = Fun2 (a,a)
```

Neka su $\alpha, \beta : \mathbf{N}^k \to \mathbf{N}$ *te* $f : \mathbf{N}^{k+1} \to \mathbf{Z}$ *rekurzivne funkcije. Neka su funkcije* $g, g', h, h' : \mathbf{N}^k \to \mathbf{Z}$ *definirane sa*

$$g(x) = \sum_{i=\alpha(x)}^{\beta(x)} f(i,x), \; g'(x) = \prod_{i=\alpha(x)}^{\beta(x)} f(i,x),$$

$$h(x) = \min_{0 \le i \le \alpha(x)} f(i,x), \; h'(x) = \max_{0 \le i \le \alpha(x)} f(i,x),$$

$x \in \mathbf{N}^k$, *pri čemu je* $g(x) = 0$ *i* $g'(x) = 1$ *ako je* $\alpha(x) > \beta(x)$. *Tada su* g, g', h, h' *rekurzivne funkcije.*

```haskell
class RingFun2 a where -- ovakvih klasa treba biti onoliko koliko
                       --i mjesnosti k (k > 1) za funkcije
    minFun2  :: (Nat -> Nat) -> Fun2 a -> Nat -> a
    maxFun2  :: (Nat -> Nat) -> Fun2 a -> Nat -> a
    sumFun2  :: (Nat -> Nat) -> (Nat -> Nat) -> Fun2 a -> Nat -> a
    prodFun2 :: (Nat -> Nat) -> (Nat -> Nat) -> Fun2 a -> Nat -> a

class RingFun3 a where
    minFun3  :: (Nat -> Nat -> Nat) -> Fun3 a -> Nat -> Nat -> a
    maxFun3  :: (Nat -> Nat -> Nat) -> Fun3 a -> Nat -> Nat -> a
    sumFun3  :: (Nat -> Nat -> Nat) -> (Nat -> Nat -> Nat) -> Fun3 a -> Nat -> Nat -> a
    prodFun3 :: (Nat -> Nat -> Nat) -> (Nat -> Nat -> Nat) -> Fun3 a -> Nat -> Nat -> a
```

Neka su $\alpha, \beta : \mathbf{N}^k \to \mathbf{N}$ *te* $f : \mathbf{N}^{k+1} \to \mathbf{Z}$ *rekurzivne funkcije. Neka su funkcije* $g, g', h, h' : \mathbf{N}^k \to \mathbf{Z}$ *definirane sa*

$$g(x) = \sum_{i=\alpha(x)}^{\beta(x)} f(i,x),\ g'(x) = \prod_{i=\alpha(x)}^{\beta(x)} f(i,x),$$

$$h(x) = \min_{0 \le i \le \alpha(x)} f(i,x),\ h'(x) = \max_{0 \le i \le \alpha(x)} f(i,x),$$

$x \in \mathbf{N}^k$, *pri čemu je* $g(x) = 0$ *i* $g'(x) = 1$ *ako je* $\alpha(x) > \beta(x)$. *Tada su* g, g', h, h' *rekurzivne funkcije.*

```haskell
instance (Num a, Ord a) => RingFun2 a where
    minFun2 alfa (Fun2 f) x = minimum [f i x | i <- [0 .. alfa x]]
    maxFun2 alfa (Fun2 f) x = maximum [f i x | i <- [0 .. alfa x]]
    sumFun2 alfa beta (Fun2 f) x = sum [f i x | i <- [alfa x .. beta x]]
    prodFun2 alfa beta (Fun2 f) x = product [f i x | i <- [alfa x .. beta x]]

instance (Num a, Ord a) => RingFun3 a where
    minFun3 alfa (Fun3 f) x y = minimum [f i x y | i <- [0 .. alfa x y]]
    maxFun3 alfa (Fun3 f) x y = maximum [f i x y | i <- [0 .. alfa x y]]
    sumFun3 alfa beta (Fun3 f) x y = sum [f i x y | i <- [alfa x y .. beta x y]]
    prodFun3 alfa beta (Fun3 f) x y = product [f i x y | i <- [alfa x y .. beta x y]]
```

Korolar 2.4. *Neka su $f, g : \mathbf{N}^k \to \mathbf{Z}$ rekurzivne funkcije. Tada su $f + g, f \cdot g : \mathbf{N}^k \to \mathbf{Z}$ rekurzivne funkcije. Nadalje, $-f, |f| : \mathbf{N}^k \to \mathbf{Z}$ su također rekurzivne funkcije.*

```haskell
instance Num a => Num (Fun1 a) where
    (Fun1 f) + (Fun1 g) = Fun1 (\x -> f x + g x)
    (Fun1 f) - (Fun1 g) = Fun1 (\x -> f x - g x)
    (Fun1 f) * (Fun1 g) = Fun1 (\x -> f x * g x)
    abs (Fun1 f) = Fun1 (\x -> abs $ f x)
    signum (Fun1 f) = Fun1 (\x -> signum $ f x)
    negate (Fun1 f) = Fun1 (\x -> negate $ f x)
    fromInteger i = Fun1 (\_ -> fromInteger i)
```

```haskell
instance Num a => Num (Fun2 a) where
    (Fun2 f) + (Fun2 g) = Fun2 (\x y -> f x y + g x y)
    (Fun2 f) - (Fun2 g) = Fun2 (\x y -> f x y - g x y)
    (Fun2 f) * (Fun2 g) = Fun2 (\x y -> f x y * g x y)
    abs (Fun2 f) = Fun2 (\x y -> abs $ f x y)
    signum (Fun2 f) = Fun2 (\x y -> signum $ f x y)
    negate (Fun2 f) = Fun2 (\x y -> negate $ f x y)
    fromInteger i = Fun2 (\_ _ -> fromInteger i)

instance Num a => Num (Fun3 a) where
    (Fun3 f) + (Fun3 g) = Fun3 (\x y z -> f x y z + g x y z)
    (Fun3 f) - (Fun3 g) = Fun3 (\x y z -> f x y z - g x y z)
    (Fun3 f) * (Fun3 g) = Fun3 (\x y z -> f x y z * g x y z)
    abs (Fun3 f) = Fun3 (\x y z -> abs $ f x y z)
    signum (Fun3 f) = Fun3 (\x y z -> signum $ f x y z)
    negate (Fun3 f) = Fun3 (\x y z -> negate $ f x y z)
    fromInteger i = Fun3 (\_ _ _ -> fromInteger i)
```

```haskell
instance (Num a, Ord a) => RingFun2 a where
    minFun2 alfa (Fun2 f) x = minimum [f i x | i <- [0 .. alfa x]]
    maxFun2 alfa (Fun2 f) x = maximum [f i x | i <- [0 .. alfa x]]
    sumFun2 alfa beta (Fun2 f) x = sum [f i x | i <- [alfa x .. beta x]]
    prodFun2 alfa beta (Fun2 f) x = product [f i x | i <- [alfa x .. beta x]]

instance (Num a, Ord a) => RingFun3 a where
    minFun3 alfa (Fun3 f) x y = minimum [f i x y | i <- [0 .. alfa x y]]
    maxFun3 alfa (Fun3 f) x y = maximum [f i x y | i <- [0 .. alfa x y]]
    sumFun3 alfa beta (Fun3 f) x y = sum [f i x y | i <- [alfa x y .. beta x y]]
    prodFun3 alfa beta (Fun3 f) x y = product [f i x y | i <- [alfa x y .. beta x y]]
```

Uočavamo „univerzalnu konstrukciju":

$$(\mathrm{Fun1}\ f) \oplus (\mathrm{Fun1}\ g) = \mathrm{Fun1}(\lambda x \to fx + gx)$$

Moramo implementirati vlastitu kompoziciju i aplikaciju (poziv) funkcija, za sve različite kombinacije mjesnosti. Iz IA znamo da je kompozicija rekurzivnih funkcija (svih koje smo spomenuli) rekurzivna funkcija.

```haskell
compose11 :: Fun1 a -> Fun1 Nat -> Fun1 a
compose11 (Fun1 f) (Fun1 g) = Fun1 (f . g)

compose12 :: Fun1 a -> Fun2 Nat -> Fun2 a
compose12 (Fun1 f) (Fun2 g) = Fun2 (\x y -> f $ g x y)

compose21 :: Fun2 a -> Fun12 Nat -> Fun1 a
compose21 (Fun2 f) (Fun1 g) = Fun1 (\x -> f (fst $ g x) (snd $ g x))

compose22 :: Fun2 a -> Fun22 Nat -> Fun2 a
compose22 (Fun2 f) (Fun2 g) = Fun2 (\x y -> f (fst $ g x y) (snd $ g x y))

apply1 :: Fun1 a -> Nat -> a
apply1 (Fun1 f) x = f x

apply2 :: Fun2 a -> Nat -> Nat -> a
apply2 (Fun2 f) x y = f x y

apply3 :: Fun3 a -> Nat -> Nat -> Nat -> a
apply3 (Fun3 f) x y z = f x y z
```

Još treba implementirati `Fractional` — to su svi tipovi brojeva koji imaju multiplikativni inverz, te sada želimo proširiti računanje njima na funkcije s tom kodomenom.

```
-- uzimamo da su funkcije oblika N^k -> Z implementirane koristeći Haskellov tip Integer
-- uzimamo da su funkcije oblika N^k -> Q implementirane koristeći Haskellov tip Rational
instance Fractional a => Fractional (Fun1 a) where
    recip (Fun1 f) = Fun1 (\x -> recip $ f x)
    fromRational x = Fun1 (\_ -> fromRational x)

instance Fractional a => Fractional (Fun2 a) where
    recip (Fun2 f) = Fun2 (\x y -> recip $ f x y)
    fromRational x = Fun2 (\_ _ -> fromRational x)

instance Fractional a => Fractional (Fun3 a) where
    recip (Fun3 f) = Fun3 (\x y z -> recip $ f x y z)
    fromRational x = Fun3 (\_ _ _ -> fromRational x)
```

Korolar 3.3. *Neka su $f, g : \mathbf{N}^k \to \mathbf{Q}$ rekurzivne funkcije. Tada su $f + g, f \cdot g : \mathbf{N}^k \to \mathbf{Q}$ rekurzivne funkcije. Nadalje, $-f, |f| : \mathbf{N}^k \to \mathbf{Q}$ su također rekurzivne funkcije. Ako je $f(x) \neq 0$ za svaki $x \in \mathbf{N}^k$, onda je $\frac{1}{f} : \mathbf{N}^k \to \mathbf{Q}$ rekurzivna funkcija.*

Izračunljivi realni brojevi

Sada dolazi zanimljiv dio: računanje s (izračunljivim) realnim brojevima. Prvo moramo definirati taj tip broja. Trebaju nam i pomoćne funkcije za aproksimiranje te kombinirano apliciranje i aproksimiranje u slučaju *funkcija* s kodomenom CR.

```haskell
-- realni brojevi
data CR where
    CR :: Fun1 Rational -> CR

approx :: CR -> Nat -> Rational
approx (CR f) k = apply1 f k

approxApply1 :: Fun1 CR -> Nat -> Nat -> Rational
approxApply1 (Fun1 f) x k = approx (f x) k

approxApply2 :: Fun2 CR -> Nat -> Nat -> Nat -> Rational
approxApply2 (Fun2 f) x y k = approx (f x y) k

approxApply3 :: Fun3 CR -> Nat -> Nat -> Nat -> Nat -> Rational
approxApply3 (Fun3 f) x y z k = approx (f x y z) k
```

```haskell
instance Num CR where
    x + y = CR $ Fun1 (\k -> approx x (k+2) + approx y (k+2))
    (CR x) * (CR y) = CR (x * y)
    abs (CR x) = CR (abs x)
    signum (CR x) = undefined -- općenito nije izračunljivo (vidjeti IA)
    negate (CR x) = CR (negate x)
    fromInteger i = CR $ Fun1 (\_ -> toRational i)

instance Fractional CR where
    recip x = CR $ Fun1 (\k -> recip $ approx x (k+fi 0))
      where fi l | abs (approx x l) > toRational (3 % (pow 2 l)) = l
                 | otherwise = fi (succ l)

    fromRational x = CR $ Fun1 (\_ -> x)
```

Uočavamo još jednostavniju konstrukciju (uz iznimku) svih operacija nad izračunljivim realnim brojevima:

$$(\mathrm{CR}\ x) \oplus (\mathrm{CR}\ y) = \mathrm{CR}(x \oplus y)$$

Odgovarajući dokazi iz Izračunljive analize za produkt i recipročnu vrijednost.

Dokažimo (ii). Neka je F rekurzivna aproksimacija od f. Prema lemi 4.3 postoji rekurzivna funkcija $M : \mathbf{N}^{k+1} \to \mathbf{N}$ takva da je

$$2 + |f(i,x)| < M(n,x),$$

za sve $n \in \mathbf{N}$, $x \in \mathbf{N}^k$ i $i \in \{0,\ldots,n\}$. Neka je $G' : \mathbf{N}^{k+1} \to \mathbf{Q}$ funkcija definirana sa

$$G'(x,l) = \prod_{i=\alpha(x)}^{\beta(x)} F(i,x,l).$$

Prema propoziciji 3.1 G' je rekurzivna funkcija.

Koristeći lemu 4.4 dobivamo

$$|g'(x) - G'(x,l)| = \left| \prod_{i=\alpha(x)}^{\beta(x)} f(i,x) - \prod_{i=\alpha(x)}^{\beta(x)} F(i,x,l) \right| <$$

$$< (\beta(x)+1)M(x,\beta(x))^{\beta(x)} 2^{-l}$$

...

Budući da za svaki $x \in \mathbf{N}^k$ postoji $l \in \mathbf{N}$ takav da je $(x,l) \in S$, prema propoziciji 1.6 postoji rekurzivna funkcija $\varphi : \mathbf{N}^k \to \mathbf{N}$ takva da je $(x,\varphi(x)) \in S$, tj.

$$\frac{3}{2^{\varphi(x)}} < |F(x,\varphi(x))| \tag{8}$$

za svaki $x \in \mathbf{N}^k$.

Neka su $x \in \mathbf{N}^k$ i $l \in \mathbf{N}$. Koristeći (9) i (10) dobivamo

$$\left| \frac{1}{f(x)} - \frac{1}{F(x,l+\varphi(x))} \right| = \left| \frac{F(x,l+\varphi(x)) - f(x)}{f(x) \cdot F(x,l+\varphi(x))} \right| < 2^{\varphi(x)} 2^{-l}.$$

Funkcija $\mathbf{N}^{k+1} \to \mathbf{Q}$, $(x,l) \mapsto \frac{1}{F(x,l+\varphi(x))}$, je rekurzivna prema propoziciji 3.1(i) i korolaru 3.3. Sada iz leme 4.1 slijedi da je $\frac{1}{f}$ rekurzivna funkcija. $\square$

Digresija: alternativa?

```haskell
-- instance Num (Fun1 CR) where
--     (Fun1 f) + (Fun1 g) = Fun1 $ \x -> CR $ Fun1 (\k -> approx (f x) (succ (succ k)) + approx (g x) (succ (succ k)))
--     (Fun1 f) - (Fun1 g) = Fun1 $ \x -> CR $ Fun1 (\k -> approx (f x) (succ (succ k)) - approx (g x) (succ (succ k)))
--     (Fun1 f) * (Fun1 g) = Fun1 $ \x -> CR $ Fun1 (\k -> approx (f x) k * approx (g x) k)
--     abs (Fun1 f) = Fun1 $ \x -> CR $ Fun1 (\k -> abs (approx (f x) k))
--     signum _ = undefined
--     negate (Fun1 f) = Fun1 $ \x -> CR $ Fun1 (\k -> negate (approx (f x) k))
--     fromInteger i = Fun1 $ \_ -> CR $ Fun1 (\_ -> toRational i)

-- instance Num (Fun2 CR) where
--     (Fun2 f) + (Fun2 g) = Fun2 $ \x y -> CR $ Fun1 (\k -> approx (f x y) (succ (succ k)) + approx (g x y) (succ (succ k)))
--     (Fun2 f) - (Fun2 g) = Fun2 $ \x y -> CR $ Fun1 (\k -> approx (f x y) (succ (succ k)) - approx (g x y) (succ (succ k)))
--     (Fun2 f) * (Fun2 g) = Fun2 $ \x y -> CR $ Fun1 (\k -> approx (f x y) k * approx (g x y) k)
--     abs (Fun2 f) = Fun2 $ \x y -> CR $ Fun1 (\k -> abs (approx (f x y) k))
--     signum _ = undefined
--     negate (Fun2 f) = Fun2 $ \x y -> CR $ Fun1 (\k -> negate (approx (f x y) k))
--     fromInteger i = Fun2 $ \_ _ -> CR $ Fun1 (\_ -> toRational i)

-- instance Num (Fun3 CR) where
--     (Fun3 f) + (Fun3 g) = Fun3 $ \x y z -> CR $ Fun1 (\k -> approx (f x y z) (succ (succ k)) + approx (g x y z) (succ (succ k)))
--     (Fun3 f) - (Fun3 g) = Fun3 $ \x y z -> CR $ Fun1 (\k -> approx (f x y z) (succ (succ k)) - approx (g x y z) (succ (succ k)))
--     (Fun3 f) * (Fun3 g) = Fun3 $ \x y z -> CR $ Fun1 (\k -> approx (f x y z) k * approx (g x y z) k)
--     abs (Fun3 f) = Fun3 $ \x y z -> CR $ Fun1 (\k -> abs (approx (f x y z) k))
--     signum _ = undefined
--     negate (Fun3 f) = Fun3 $ \x y z -> CR $ Fun1 (\k -> negate (approx (f x y z) k))
--     fromInteger i = Fun3 $ \_ _ _ -> CR $ Fun1 (\_ -> toRational i)
```

Ipak moramo posebno definirati funkcije iz naše dodatne klase, jer izračunljivi realni brojevi nemaju odlučiv uređaj, pa ne mogu koristiti prošlu implementaciju.

```
instance RingFun2 CR where
    minFun2 alfa (Fun2 f) x = CR $ Fun1 (\k -> minimum [approx (f i x) k | i <- [0 .. alfa x]])
    maxFun2 alfa (Fun2 f) x = CR $ Fun1 (\k -> maximum [approx (f i x) k | i <- [0 .. alfa x]])
    sumFun2 alfa beta (Fun2 f) x = CR $ Fun1 (\k -> sum [approx (f i x) (k+beta x+1) | i <- [alfa x .. beta x]])
    prodFun2 alfa beta (Fun2 f) x = CR $ Fun1 (\k -> product [approx (f i x) k | i <- [alfa x .. beta x]])

instance RingFun3 CR where
    minFun3 alfa (Fun3 f) x y = CR $ Fun1 (\k -> minimum [approx (f i x y) k | i <- [0 .. alfa x y]])
    maxFun3 alfa (Fun3 f) x y = CR $ Fun1 (\k -> maximum [approx (f i x y) k | i <- [0 .. alfa x y]])
    sumFun3 alfa beta (Fun3 f) x y = CR $ Fun1 (\k -> sum [approx (f i x y) (k+beta x y+1) | i <- [alfa x y .. beta x y]])
    prodFun3 alfa beta (Fun3 f) x y = CR $ Fun1 (\k -> product [approx (f i x y) k | i <- [alfa x y .. beta x y]])
```

Neka je S skup svih $(n,k) \in \mathbf{N}^2$ takvih da vrijedi (8). Iz korolara 3.5 slijedi da je S rekurzivan skup. Za svaki $n \in \mathbf{N}$ postoji $k \in \mathbf{N}$ takav da je $(n,k) \in S$ pa prema propoziciji 1.6 postoji rekurzivna funkcija $\varphi : \mathbf{N} \to \mathbf{N}$ takva da je $(n, \varphi(n)) \in S$, tj.

$$u'(n \cdot a(\varphi(n)), b(\varphi(n))) + \overline{\mathrm{sg}}(n) > n \cdot 2^{-\varphi(n)} \tag{9}$$

```haskell
-- iščitavanje decimale *pozitivnog* realnog broja; PAZI: indekse decimala brojimo od 1, a ne od 0
decimalBase :: Nat -> CR -> Nat -> Nat
decimalBase b f n = mod (g (pow b n)) b
   where g :: Nat -> Nat
         g n = floor $ toRational n * abs (approx f (fi 0))
         fi k | u' (n * absnum (approx f k)) (absden (approx f k)) + toRational (sgc n) > toRational n * recip (toRational (pow 2 k)) = k
              | otherwise = fi (succ k)
         u' :: Nat -> Nat -> Rational
         u' x y = toRational $ min ((x % l y) - floor (x % l y) % 1) (floor (x % l y) % 1 + 1 - (x % l y))
         l :: Nat -> Nat
         l n = n + sgc n
         absnum :: Rational -> Nat
         absnum x = fromInteger $ abs (numerator x)
         absden :: Rational -> Nat
         absden x = fromInteger $ abs (denominator x)

decimal = decimalBase 10
binary = decimalBase 2
```

Prema tome za svaki $n \in \mathbf{N}$ vrijedi

$$g(n) = \left\lfloor \frac{n \cdot a(\varphi(k))}{b(\varphi(k))} \right\rfloor$$

pa slijedi da je g rekurzivna funkcija. Time je tvrdnja teorema dokazana. $\square$

Funkcija u' u dokazu:

Neka je $l : \mathbf{N} \to \mathbf{N}$ funkcija definirana sa $l(n) = n + \overline{\mathrm{sg}}(n)$. Tada je $l(n) = n$ za svaki $n \geq 1$ i $l(n) \geq 1$ za svaki $n \in \mathbf{N}$. Definirajmo $u' : \mathbf{N}^2 \to \mathbf{Q}$ sa

$$u'(x, y) = \min \left\{ \frac{x}{l(y)} - \left\lfloor \frac{x}{l(y)} \right\rfloor, \left\lfloor \frac{x}{l(y)} \right\rfloor + 1 - \frac{x}{l(y)} \right\}.$$

Za prikaz proizvoljno mnogo decimala pozitivnog izračunljivog broja (uključujući i njegov cjelobrojni dio), koristimo funkciju `showReal`:

```
-- prikaz proizvoljno mnogo decimala pozitivnog realnog broja (uključujući i cjelobrojni dio)
showReal :: CR -> Nat -> String
showReal f n = show int ++ ('.' : decimals)
 where int = floor $ abs (approx f 1)
       decimals = [head $ show (decimal f i) | i <- [1 .. n]]
```

Na Izračunljivoj analizi smo istaknuli da možemo „promovirati” funkcije stupnjevito, pa ovdje to eksplicitno definiramo:

```haskell
-- "promocije" funkcija N^k->N u N^k->Z
--                      N^k->Z u N^k->Q
--                      N^k->Q u N^k->R

fromNatural1 :: Fun1 Nat -> Fun1 Integer
fromNatural1 (Fun1 f) = Fun1 (\x -> toInteger $ f x)

fromNatural2 :: Fun2 Nat -> Fun2 Integer
fromNatural2 (Fun2 f) = Fun2 (\x y -> toInteger $ f x y)

fromNatural3 :: Fun3 Nat -> Fun3 Integer
fromNatural3 (Fun3 f) = Fun3 (\x y z -> toInteger $ f x y z)

fromIntegral1 :: Fun1 Integer -> Fun1 Rational
fromIntegral1 (Fun1 f) = Fun1 (\x -> toRational $ f x)

fromIntegral2 :: Fun2 Integer -> Fun2 Rational
fromIntegral2 (Fun2 f) = Fun2 (\x y -> toRational $ f x y)

fromIntegral3 :: Fun3 Integer -> Fun3 Rational
fromIntegral3 (Fun3 f) = Fun3 (\x y z -> toRational $ f x y z)

fromRational1 :: Fun1 Rational -> Fun1 CR
fromRational1 (Fun1 f) = Fun1 (\x -> CR $ Fun1 (\_ -> f x))

fromRational2 :: Fun2 Rational -> Fun2 CR
fromRational2 (Fun2 f) = Fun2 (\x y -> CR $ Fun1 (\_ -> f x y))

fromRational3 :: Fun3 Rational -> Fun3 CR
fromRational3 (Fun3 f) = Fun3 (\x y z -> CR $ Fun1 (\_ -> f x y z))
```

Račun s funkcijama oblika $f :\subseteq \mathbb{R} \to \mathbb{R}$

- Htjeli bismo još da naš kalkulator može računati i s funkcijama koje kao domenu imaju neki podskup od $\mathbb{R}$.

Račun s funkcijama oblika $f :\subseteq \mathbb{R} \to \mathbb{R}$

- Htjeli bismo još da naš kalkulator može računati i s funkcijama koje kao domenu imaju neki podskup od $\mathbb{R}$.
- Postoji više smislenih načina za definirati izračunljivost nad ovakvim funkcijama.

Račun s funkcijama oblika $f :\subseteq \mathbb{R} \to \mathbb{R}$

- Htjeli bismo još da naš kalkulator može računati i s funkcijama koje kao domenu imaju neki podskup od $\mathbb{R}$.
- Postoji više smislenih načina za definirati izračunljivost nad ovakvim funkcijama.
- Definicija s IA je malo prespecifična za naše potrebe, pa mi uvodimo sljedeću definiciju:

Definicija

Za funkciju $f : S \to \mathrm{CR}$, $S \subseteq \mathbb{R}$ kažemo da je izračunljiva ako za svaki $x \in S$ vrijedi da je $f(x)$ izračunljiv realan broj.

Možemo dakle reći i ovako: $f : S \to \mathrm{CR}$ je izračunljiva ako je $S \subseteq \mathrm{CR}$.

Računanje polinoma

$$p : \mathbb{R} \to \mathbb{R}$$
$$p(x) = a_0 + a_1 x + \cdots + a_n x^n$$

Polinome lako računamo direktno i zadajemo ih listom koeficijenata, gdje je prvi slobodan:

```
poly :: [CR] -> (CR -> CR)
poly [] = undefined
poly as = \x -> g x 0 as
 where g :: CR -> Nat -> [CR] -> CR
       g _ _ [] = 0
       g x i (a:as) = pow x i * a + g x (succ i) as
```

Računanje redova potencija

Neka je zadan rekurzivan (izračunljiv) niz realnih brojeva $(a_n)_{\mathbb{N}}$. Onda za neki $R \in \langle 0, \infty\rangle \cup \{\infty\}$ imamo

$$\sum_{i=0}^{\infty} a_i x^i,\ x \in \langle -R, R\rangle.$$

Računanje redova potencija

Neka je zadan rekurzivan (izračunljiv) niz realnih brojeva $(a_n)_{\mathbb{N}}$. Onda za neki $R \in \langle 0, \infty\rangle \cup \{\infty\}$ imamo

$$\sum_{i=0}^{\infty} a_i x^i,\ x \in \langle -R, R\rangle.$$

Račun ide nešto teže — zahtijeva mnoštvo dodatnih rezultata. Naime, pokazuje se da je račun reda općenito moguć samo ako se uz argument x u području konvergencije zadaju i tri dodatna parametra takva da vrijedi odnos: $|x| < s < r$ te $M \in \mathbb{N}$, $r, s \in \mathbb{Q}$.

Računanje redova potencija

Neka je zadan rekurzivan (izračunljiv) niz realnih brojeva $(a_n)_{\mathbb{N}}$. Onda za neki $R \in \langle 0, \infty\rangle \cup \{\infty\}$ imamo

$$\sum_{i=0}^{\infty} a_i x^i,\ x \in \langle -R, R\rangle.$$

Račun ide nešto teže — zahtijeva mnoštvo dodatnih rezultata. Naime, pokazuje se da je račun reda općenito moguć samo ako se uz argument x u području konvergencije zadaju i tri dodatna parametra takva da vrijedi odnos: $|x| < s < r$ te $M \in \mathbb{N}$, $r, s \in \mathbb{Q}$. Iz analize znamo da za svaki $r < R$ postoji konstanta $M \in \mathbb{N}$ takva da vrijedi

$$|a_j| \leq M \cdot r^{-j} \text{ za sve } j \in \mathbb{N}.$$

[2]

```haskell
-- pomoćne funkcije za račun redova potencija
-- vidi Weihrauch, Computable Analysis Theorem 4.3.7
lim :: Fun1 CR -> (Nat -> Nat) -> CR
lim f e = CR $ Fun1 (\i -> approx (apply1 f (e (2+i))) (2+i))

slim :: Fun1 CR -> (Nat -> Nat) -> CR
slim xs e = lim (ss xs) e
  where ss :: Fun1 CR -> Fun1 CR
        ss f = Fun1 $ \i -> sum [apply1 f j | j <- [0 .. i]]

-- računanje reda potencija sa zadanim parametrima s, r:
-- |x| < s < r < R (R=radijus konvergencije)
-- vidi Theorem 4.3.11
series :: Fun1 CR -> Rational -> Rational -> Nat -> CR -> CR
series f r s m x = slim (g f x) (h' r s m)
  where g :: Fun1 CR -> CR -> Fun1 CR
        g a x' = Fun1 $ \i -> apply1 a i * pow x' i
        h :: Rational -> Rational -> Nat -> Nat -> Nat
        h r s m n = search 0
         where search k | toRational m * (pow (s / r) k) * (r / (r-s)) <= recip (toRational (pow 2 n)) = k
                        | otherwise = search (succ k)
        h' :: Rational -> Rational -> Nat -> (Nat -> Nat)
        h' r s m = \i -> h r s m i
```

Moguća poboljšanja/proširenja

- odabir najefikasnijih algoritama i notacija/reprezentacija za računanje s realnim brojevima; mnoge dobre ideje npr. u [1];
- omogućavanje računanja uređaja, s tim da on postaje ograničene preciznosti — unutar određene kritične zone ne može biti determinističan;
- dizajn zasebnog programskog jezika specijaliziranog za računanje s tipom CR, funkcijama nad njim i moguće još nekim vezanim strukturama;
- parametrizacija mjesnosti u Haskellu (ne pomoću nekog metajezika poput Template Haskella!), koristeći neke GHC ekstenzije;
- dodati rješavač linearnih sustava, vađenje korijena polinoma. . .
- implicitno/automatsko procjenjivanje aritmetičke složenosti cjelokupnog programa, ovisno o statički specificiranim parametrima preciznosti na ključnim mjestima

Literatura

[1] Vasco Brattka i Peter Hertling. „Feasible real random access machines”. *SOFSEM'96: Theory and Practice of Informatics*. Ur. Keith G. Jeffery, Jaroslav Král i Miroslav Bartošek. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996., str. 335–342. ISBN: 978-3-540-49588-8.

[2] K. Weihrauch. *Computable Analysis: An Introduction*. Texts in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg, 2000.