10 Numere

Haskell oferă o bogată colecţie de tipuri numerice, inspirate de cele din limbajul Scheme [7], care la rândul lor se bazează pe limbajul Common Lisp [8]. (Aceste limbaje, spre deosebire de Haskell, au mecansim de tipizare dinamica – tipurile stabilindu-se la executie.) Tipurile standard includ numere întregi fixe şi numere cu precizie arbitrara, fracţiile (numerele raţionale), formate din numarator si numitor de tip întreg, numerele reale cu precizie simpla si dubla şi numere complexe cu virgulă mobilă. Subliniem aici caracteristicile de bază ale structurii de „clasă de tipuri numerice” (clasa Num) cu trimiţând pentru cititor la capitolul 6.4 pentru detalii.

10.1 Structura clasei numerelor: Num

Clasele de tipuri numerice (clasa Num şi cele care sunt subclase pentru ea) se numară printre clasele standard ale limbajului Haskell. De asemenea, notaţi că Num este o subclasă a clasei Eq, dar nu şi a clasei Ord, acest lucru se datorează faptului că predicatele de ordine (comparatiile >,< etc) nu se aplică numerelor complexe. Subclasa Real a clasei Num este deasemenea o subclasă şi pentru clasa Ord.

Clasa Num prevede mai multe operaţii de bază comune pentru toate tipurile numerice, acestea includ, printre altele, adunarea, scăderea, negaţia (schimbarea de semn), înmultirea şi valoarea absolută:

```
(+), (-), (*) :: (Num a) => a -> a -> a
negate, abs   :: (Num a) => a -> a
```

[negaţia este funcţia aplicată de operatorul prefixat minus; putem sa-l numim (-), pentru că aceasta este funcţia care face scăderea 0-x , si acest nume este înţeles imediat. De exemplu, -x*y este echivalent cu negate(x*y). (Prefixul minus are aceeaşi prioritate sintactica ca si infixul minus, ambele avand desigur prioritate mai mica decât a operatiei de inmultire.)]

Clasa Integral oferă toate celelalte operaţii incluzand si împarţirea tuturor numerelor. Instanţele standard ale clasei Integral sunt Integer (numere întregi fără limite sau întregi matematici, de asemenea, cunoscute sub numele de „bignums” - intregi lungi) şi Int (delimitate, numerele întregi din limbajul maşină, cu un interval echivalent cu cel puţin 29 de biţi, codate in binar). O implementare particulară a lui Haskell poate oferi alte tipuri de întregi în plus faţă de acestea. Reţineţi că Integral este o subclasă a clasei Real, mai degrabă decât a clasei Num direct, acest lucru înseamnă că nu există nici o încercare de a oferi întregi Gaussieni.

Toate celelalte tipuri numerice se încadrează în clasa Fractional, care include împărţirea normală notata cu operatorul (/). Următoarea subclasă conţine funcţiile trigonometrice, logaritmică şi exponenţială.

Subclasa RealFrac a clasei Fractional şi Real oferă cate o funcţie properFraction, care descompune un număr în partea sa întreagă şi cea fracţională, şi o colecţie de funcţii care rotunjesc un număr la o valoare întreagă prin diferite metode:

```
properFraction    :: (Fractional a, Integral b) => a -> (b,a)
truncate, round,
floor, ceiling:          :: (Fractional a, Integral b) => a -> b
```

Subclasa RealFloat a clasei Floting și RealFrac oferă câteva funcții specializate pentru accesul eficient la componentele unui număr cu virgulă mobilă, exponent și rădăcină. Tipurile standard Float și Double se încadrează în clasa RealFloat.

10.2 Numere complexe

Tipurile numerice standard, Int, Integer, Float și Double sunt primitive. Alte tipuri sunt realizate din acestea folosind diversi constructori de tipuri .

Complex (găsit în biblioteca Complex) este un constructor de tip care face un tip complex în clasa Floating dintr-un tipul din clasa RealFloat:

```
data (RealFloat a) => Complex a = !a :+ !a deriving (Eq, Text)
```

Simbolurile ! sunt fanioane de strictețe, acestea au fost explicate in secțiunea 6.3. Observați că precizarea din context (RealFloat a), limiteaza tipul argumentelor a, astfel tipurile complexe standard vor fi Complex Float si Complex Double. Putem vedea, de asemenea, din declarația constructorului de date că un număr complex este scris sub forma x:+y; Argumentele sunt partea reală, respectiv imaginară a numarului imaginar. Dat fiind ca :+ este un constructor de date, noi îl putem folosi – de exemplu – la conjugarea numerelor complexe (n.tr. Si la alte operatii care presupun (re)construirea unor numere complexe.)

```
conjugate          :: (RealFloat a) => Complex a -> Complex a
conjugate (x:+y)  = x :+ (-y)
```

În mod similar, constructorul de tip Ratio (găsit în biblioteca Rational) produce un tip rațional din clasa RealFrac dintr-un tip **a,** instanță a clasei Integral. (Rational este in realitate un tip sinonim cu Ratio Integer.) Ratio, cu toate acestea, este un constructor de tip abstract. În loc de un constructor de date cum ar fi : +, raționalele utilizează funcția „%” pentru a forma un raport de două numere întregi. Ca exemlu de folosire a %, in locul exemplului despre conjugarea numerelor complexe, prezentam funcția de extragere a componentelor – de fapt tipurile lor :

```
(%)                          :: (Integral a) => a -> a -> Ratio a
numerator, denominator       :: (Integral a) => Ratio a -> a
```

Este vreo diferența? Numerele complexe în forma carteziană sunt unice - nu există identități netriviale care implică „:+”. Pe de altă parte, rapoartele, fractiile, nu sunt unice (8/6=4/3), dar au o formă canonică (redusă), pe care implementarea tipurilor de date abstracte trebuie să o conserve! Nu este neapărat cazul, de exemplu, ca numărătorul extras din fractia (x % y) să fie egal cu x , deşi partea reală a numarului complex x :+ y este **întotdeauna** x.