floating_point_numbers

May 20, 2020

[1]:
```
restart
```

```
--loading configuration for package "FourTiTwo" from file
/home/hegland/.Macaulay2/init-FourTiTwo.m2
--loading configuration for package "Topcom" from file
/home/hegland/.Macaulay2/init-Topcom.m2
```

0.0.1 A lightweight but optimal implementation of floating point arithmetic

- the following code is for illustrative purposes and might still contain errors

dyadic fractions

- a floating point number x is a **dyadic fraction**, i.e. it is of the form

$$x = \frac{m}{2^e}$$

  where the mantissa $m \in \mathbb{Z}$ and the exponent $e \in \mathbb{N}$
- the dyadic fractions $\mathbb{D}$ form a ring and one has

$$\mathbb{Z} \subset \mathbb{D} \subset \mathbb{Q} \subset \mathbb{R}$$

- here we implement dyadic fractions as an extension of $\mathbb{Z}$ with 1/2
- the ring of dyadic fractions is not a field

[1]:
```
-- the ring of dyadic fractions
DF = ZZ[h]/(2*h-1)
```

```
o1 = DF

o1 : QuotientRing
```

[114]:
```
-- generate a random element of DD
e = -random(10)  -- exponent which is negative
m = random(100)-50  -- mantissa
<< e << endl;
x = m*h^e
```

```
7
```

```
         7    2
o114 = h   + h

o114 : DF
```

[117]:
```
-- recover the mantissa m and exponent e from the dyadic fraction
er = (degree(x))_0
mr = x*2^er
x - mr*h^er  -- this should be zero
```

```
o117 = 0

o117 : DF
```

application of conversion functions

- the following function FQ maps dyadic fractions to rational numbers
  - with this one can apply any function on rational numbers to dyadic fractions, the result is a rational number
- the function FR maps dyadic fractions to real numbers
  - this might give unexact results
  - this is useful for printing results

[120]:
```
-- convert dyadic numbers to QQ and RR
FQ = map(QQ,DF,{h=>1/2}) -- maps DF to QQ and h maps to 1/2
FR = map(RR,DF,{h=>1/2}) -- maps DF to QQ and h maps to 1/2
<<"x = " << x << " = " << FQ(x) <<" = "<< FR(x) << endl;
```

```
     7    2    33
x = h   + h   = --- = .257812
               128
```

floating point numbers

- the dyadic numbers are dense in $\mathbb{R}$ like $\mathbb{Q}$
- they admit a convenient approximation which is implemented as a rounding function

$$\rho_t : \mathbb{Q} \to \mathbb{D}$$

- the parameter t controls the precision of the approximation
- the range of ρ_t is the set of *floating point numbers* $\mathbb{F}$ and one has

$$\{n \in \mathbb{Z} \mid |n| < 2^t\} \subset \mathbb{F} \subset \mathbb{D}$$

- more specifically

$$\mathbb{F} = \{m, 2^e \mid |m| < 2^t, \quad \text{where } m, e \in \mathbb{Z}\}$$

- this is a slight idealisation as in practice e is considered to be in a (sufficiently large) subset of $\mathbb{Z}$
- **Note:** $\mathbb{F}$ is not a ring! Even the sum of two floating point numbers is typically not a floating point number

[122]:
```
-- round rationals to dyadic numbers (output=dyadic fractions)
-- t = precision parameter (as for RR)

rho = (x,t,DF) -> (
    if x == 0 then return 0_DF
    else if x > 0 then (
        m = x; f=1_DF;
        while m < 2^(t-1) do (m=2*m; f=h*f);
        while m >= 2^t  do (m=m/2; f = 2*f);
        return round(m)*f)
    else return -rho(-x,t,h))

FR(rho(1/3,53,DF))-1/3 -- same as for floating point RR
```

```
o122 = -1.85037170770859e-17

o122 : RR (of precision 53)
```

lifting functions defined on $\mathbb{D}$ to $\mathbb{F}$

- simple formula for unary functions

$$f_{\mathbb{F}}(x) = \rho(E(f_{\mathbb{D}}(x))$$

- this is an approximation
- the same for binary functions

$$f_{\mathbb{F}}(x, y) = \rho(E(f_{\mathbb{D}}(x, y)))$$

- here E is the embedding of $\mathbb{D}$ into $\mathbb{Q}$, i.e., the function FQ from above
- one could also implement the rounding function on $\mathbb{D}$ instead of $\mathbb{Q}$

[131]:
```
t = 4
UF = f -> (x -> rho(FQ(f(x)),t,DF)) -- unary functions f
BF = f -> ((x,y) -> rho(FQ(f(x,y)),t,DF))-- binary functions f

x = rho(1/3, t, DF)  -- rho incurs an error
y = rho(1/4, t, DF)  -- rho incurse an error
errplus = FR((BF(plus))(x,y) - (x+y)) -- BF(plus) incurs an error
<< "error in addition of 1/3+1/4 = "<< errplus << endl;
<< "sum of errors in approx of 1/3 and 1/4 ="<< FR(x+y)-(1/3+1/4) <<endl;
```

```
error in addition of 1/3+1/4 = .03125

sum of errors in approx of 1/3 and 1/4 =.0104167
```