ProgrammierParadigmen

Übung - Gruppe 1 & 2

Tobias Kahlert

Eigene Operatoren/Precedences

Prec-edence	Left associative operators	Non-associative operators	Right associative operators
9	`!!`		`.`
8			`^`, `^^`, `**`
7	`*`, `/`, `` `div` ``,		
	`` `mod` ``, `` `rem` ``, `` `quot` ``		
6	`+`, `-`		
5			`:`, `++`
4		`==`, `/=`, `<`, `<=`, `>`, `>=`,	
		`` `elem` ``, `` `notElem` ``	
3			`&&`
2			`||`
1	`>>`, `>>=`		
0			`$`, `$!`, `` `seq` ``

https://www.haskell.org/onlinereport/decls.html#sect4.4.2

Klammerung

1. Explizite Klammern (plus besondere Syntax wie *let..in*)
   4 * (5 + 6) => (4 * (5 + 6))
2. Lambdaausdrücke erfassen alles auf der rechten Seite
   \x -> \y -> y + y => (\x -> (\y -> (y + y)))
3. Funktionsapplikation
   f a => (f a)
   f a b => (f a) b)
   f a f b => (((f a) f) b)
4. Operatoren mit Precedences und Associativity
   a + b * c => (a + (b * c))
   a + b + c => ((a + b) + c)
   a ^ b ^ c => (a ^ (b ^ c))

Unabhängig davon, was f ist

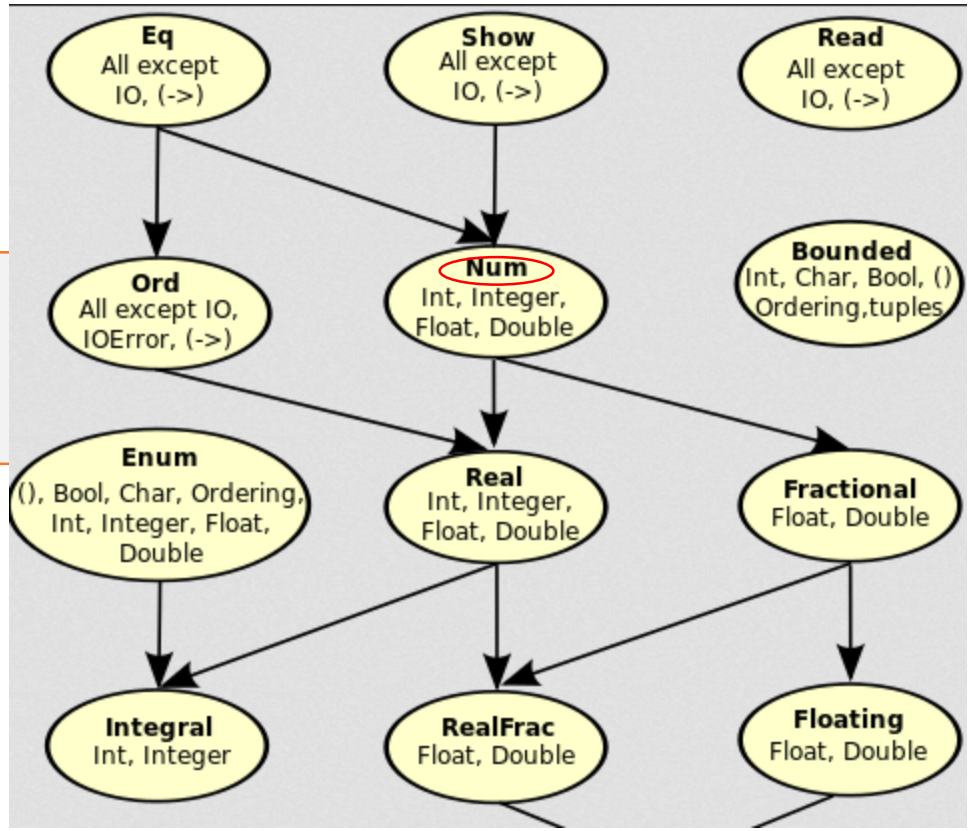
Prec-edence	Left associative operators	Non-associative operators	Right associative operators
9	`!!`		`.`
8			`^`, `^^`, `**`
7	`*`, `/`, `` `div` ``,		
	`` `mod` ``, `` `rem` ``, `` `quot` ``		
6	`+`, `-`		
5			`:`, `++`
4		`==`, `/=`, `<`, `<=`, `>`, `>=`,	
		`` `elem` ``, `` `notElem` ``	
3			`&&`
2			`||`
1	`>>`, `>>=`		
0			`$`, `$!`, `` `seq` ``

a + b c + d => ~~(a + b) (c + d)~~
=> a + (b c) + d

\x -> x z => ~~(\x -> x) z~~
=> \x -> (x z)

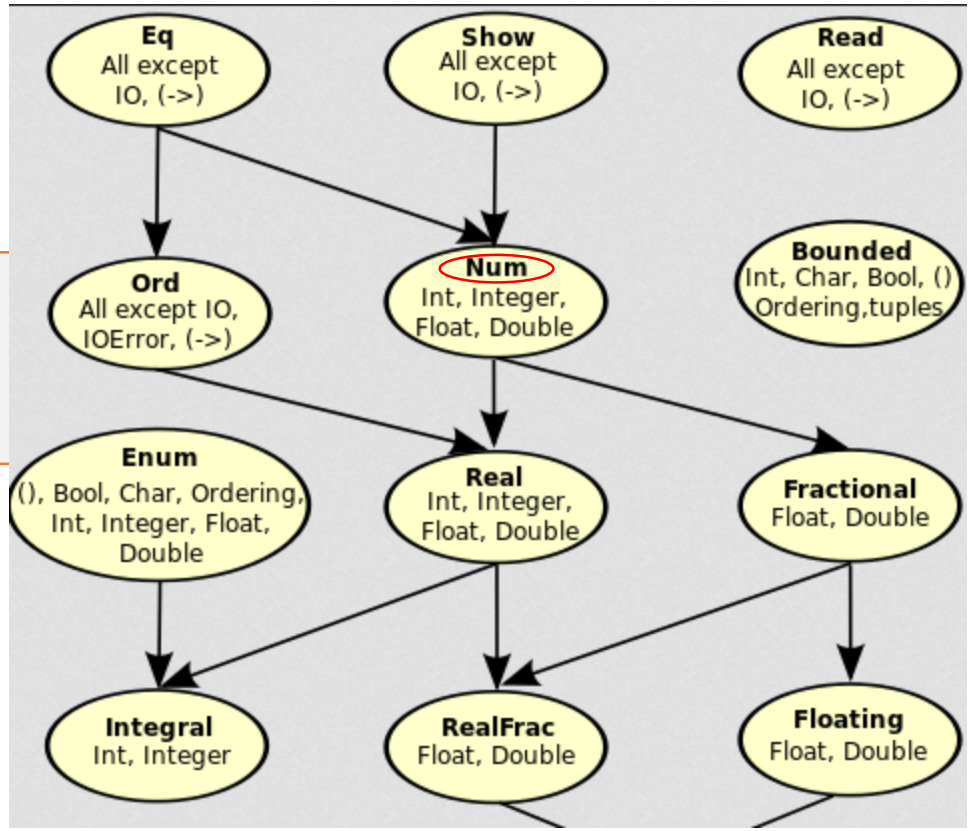
:t 3

```
:t 3
=> 3 :: Num t => t
```


Eq
All except
IO, (->)
Show
All except
IO, (->)
Read
All except
IO, (->)
Ord
All except IO,
IOError, (->)
Num
Int, Integer,
Float, Double
Bounded
Int, Char, Bool, ()
Ordering,tuples
Enum
(), Bool, Char, Ordering,
Int, Integer, Float,
Double
Real
Int, Integer,
Float, Double
Fractional
Float, Double
Integral
Int, Integer
RealFrac
Float, Double
Floating
Float, Double

:t 3

```
:t 3
=> 3 :: Num t => t
```



Haskell-Report Sektion 6.4.1

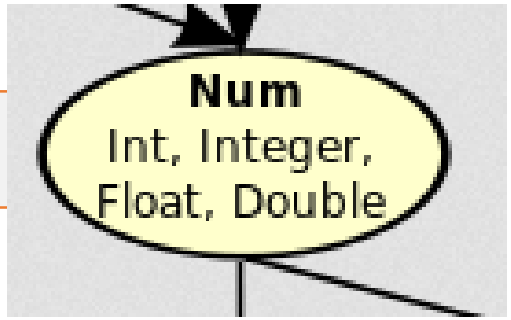
Das ist die „3“

“… An integer literal represents the application of the function fromInteger to the appropriate value of type Integer. …”

```
fromInteger  :: (Num a) => Integer -> a
```

:t 3

"… An integer literal represents the application of the function fromInteger to the appropriate value of type Integer. …"




```
fromInteger  :: (Num a) => Integer -> a
```

a ist im Fall unten Double

```
x :: Double
x = fromInteger 3          -- ist wie ‚x = 3‘
```

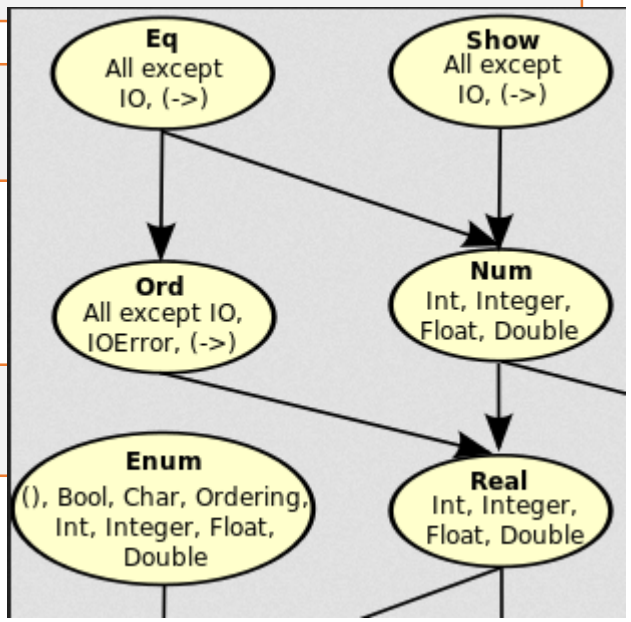
:t 3 … Ord?

```
fromInteger  :: (Num a) => Integer -> a
```

```
(<)  :: (Ord a) => a -> a -> a
```

a ?

```
(fromInteger 4) < (fromInteger 3)
```



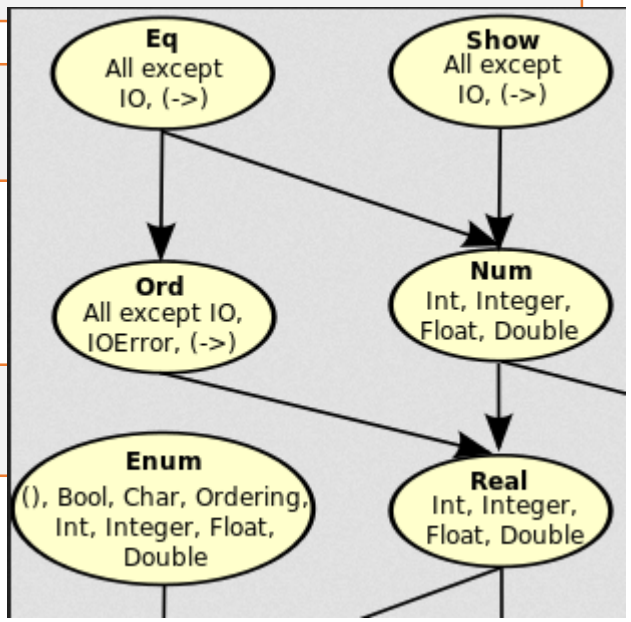
:t 3 … Ord?

```
fromInteger  :: (Num a) => Integer -> a
```

```
(<)  :: (Ord a) => a -> a -> a
```

a wird Integer

```
(fromInteger 4) < (fromInteger 3)
```



Haskell-Report Sektion 4.3.4

"… Ambiguities in the class Num are most common, so Haskell provides another way to resolve them—with a default declaration: default (t_1 , … , t_n) …"

Blatt 3

- Fibonacci Zahlen
- Collatz-Vermutung
- Primepowers
- (Hamming-Zahlen)
- (Pseudozufallszahlen)

Laziness und unendliche Listen

- Vergleiche:

```
even          = 2 : map (+2) even
multiple n    = n : map (+n) (multiple n)
```

- Fix?

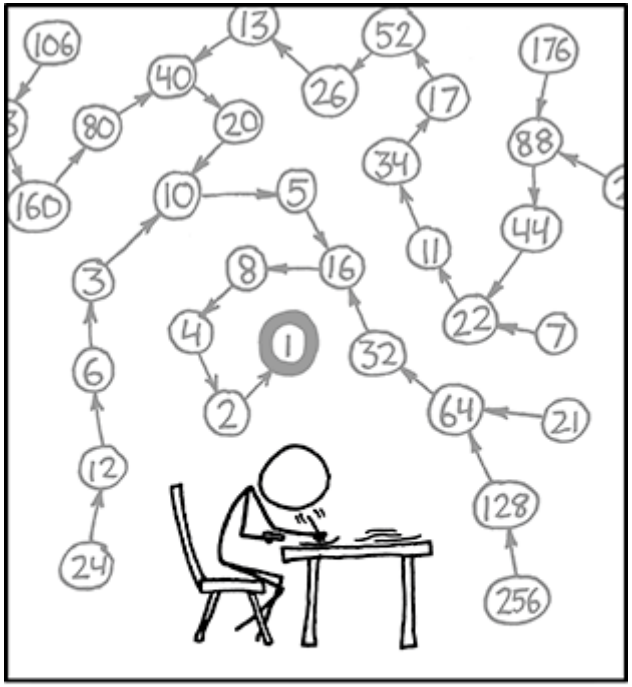
Unendliche Listen

1. Definiere eine unendliche Liste Zs die alle ganzen Zahlen $\mathbb{Z}$ enthält.

2. Definiere eine unendliche Liste Qs die alle positiven rationalen Zahlen $\mathbb{Q}$ enthält.

3. Definiere eine unendliche Liste Rs die alle positiven reellen Zahlen $\mathbb{R}$ enthält?

Collatz Conjecture



THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

https://xkcd.com/710/