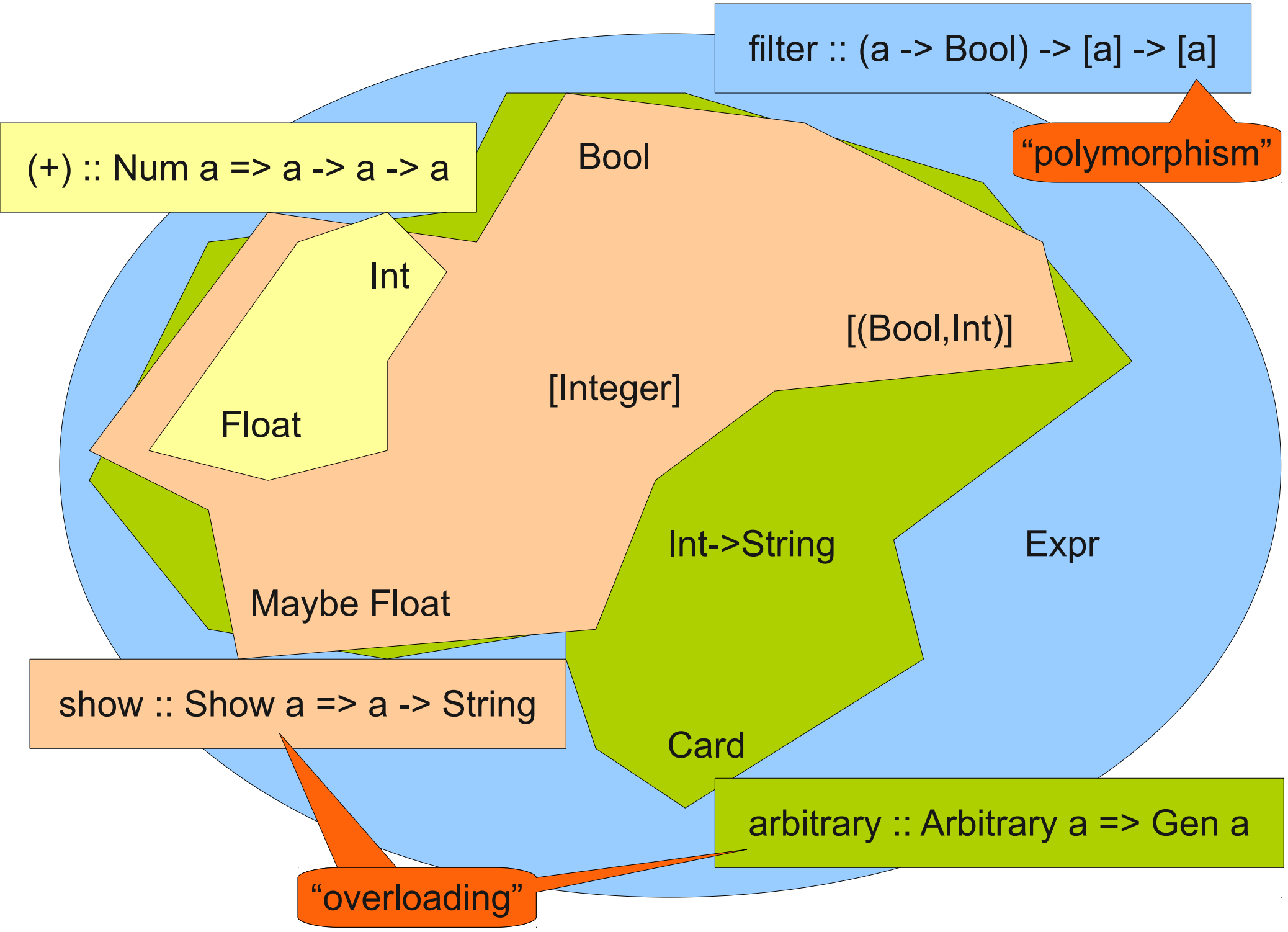
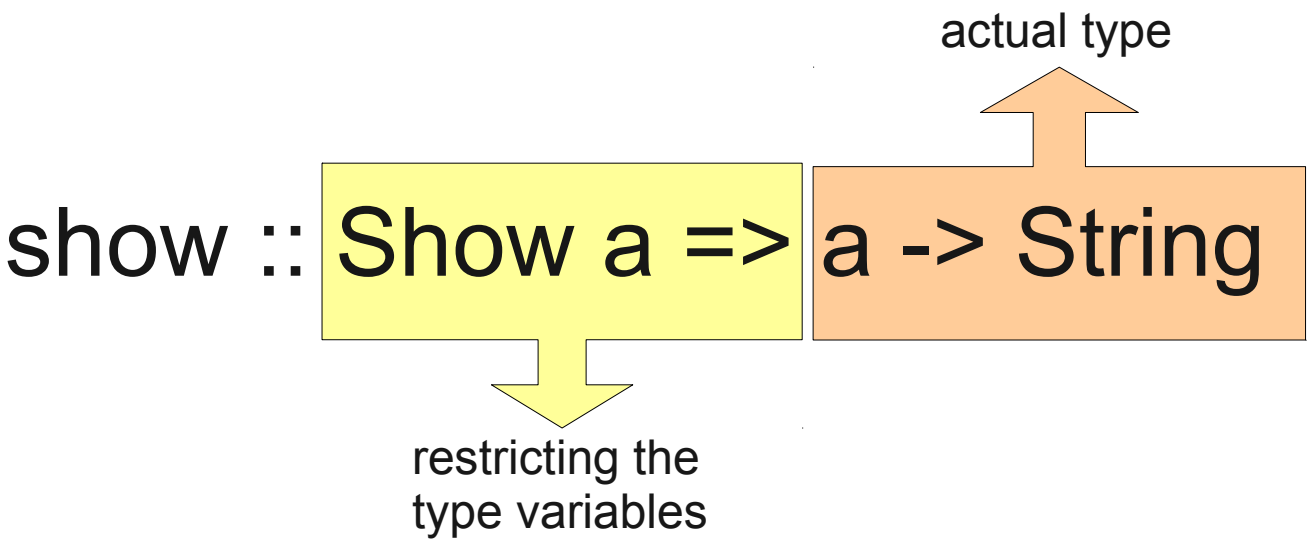
Type Classes in Haskell


filter :: (a -> Bool) -> [a] -> [a]
"polymorphism"
(+) :: Num a => a -> a -> a
Bool
Int
[(Bool,Int)]
[Integer]
Float
Int->String
Expr
Maybe Float
show :: Show a => a -> String
Card
arbitrary :: Arbitrary a => Gen a
"overloading"

Notation


actual type
show :: Show a => a -> String
restricting the
type variables

Type Classes

```
class Arbitrary a where
  arbitrary :: Gen a
```

```
instance Arbitrary Bool where
  arbitrary = choose (False,True)
```

```
instance Arbitrary Card where
  arbitrary = arbCard
```

```
class Num a where
  (+) :: a -> a -> a
  (-) :: a -> a -> a
  (*) :: a -> a -> a
  fromInteger :: Integer -> a
```

```
instance Num Bool where
  (+) = (||)
  (-)  = (/=)
  (*)  = (&&)
  fromInteger = even
```

```
instance Num Complex where
  ...
```

Making Instances by Deriving

```
data MyShinyType = ...
   deriving ( Show, Eq )
```

```
Show a
Read a
Eq a
Ord a
Enum a
...
```

```
-- show
-- read
-- (==), (/=)
-- (<=), (>=), (<), (>)
-- [x..y]
```

Effect of using overloading

any function that uses an overloaded function automatically becomes overloaded too!

```
find :: Eq a => a -> [a] -> Int

find x []                          = error “x not found”
find x (y:ys) | x == y       = 1
                    | otherwise = 1+find x ys
```

More Reading

- Book: Chapter 12
- On haskell.org: http://www.haskell.org/tutorial/classes.html