FP vs OOP?

Mikołaj Fejzer

Versus?

<< | >>
#4591 Dodano: 18-02-2011 13:37. Głosów: 196 + | -

Prawie dekade temu kumpel w gimnazjum:
- Gral ktos w Alien versus Predator? Fajna gra, mi najlepiej sie gra versusem, bo ma najwiecej broni i radar na obcych.

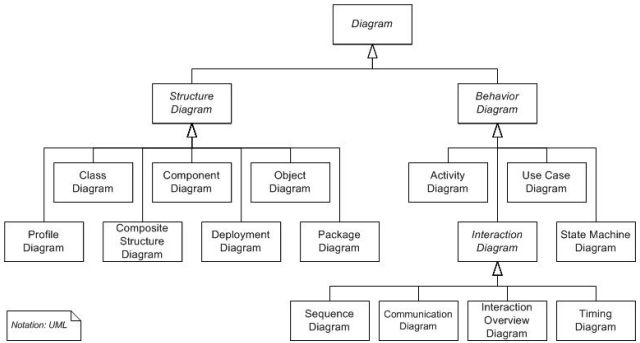
<< | >>

http://roflcopter.pl/4591

Versus

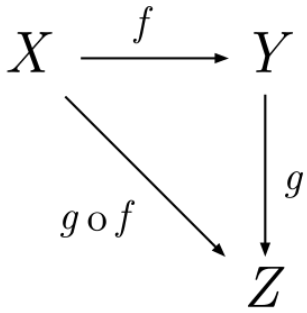
OOP

- Kapsułkowanie
- Hierarhia klas, dziedziczenie
- Zmiana stanu w czasie
- Polimorfizm podtypu



FP

- Funkcje jako typy pierwszego rzędu
- Algebraiczne struktury danych
- Brak efektów ubocznych
- Polimorfizm parametryczny



“The limits of my language mean the limits of my world.” L. Wittgenstein

- Język programowania jest zawsze ograniczony przez jego twórców
- Użytkownicy tworzą “dobrą praktykę” i tworzą struktury/klasy/mechanizmy w podobny sposób
- Pojawiają się “wzorce projektowe”

Czy to jest jeszcze wzorzec projektowy?

- Iterator, Observer
- Builder
- Visitor, Strategy

Trochę teorii o typach

Po co istnieją typy?

- Kompilator
  - odgadnie o co nam chodzi
  - wygeneruje testy
  - sprawdzi kontrakty
- Możemy na nich wykonywać działania, i używać rezultatu jako typu

System F oraz system typów Hindleya–Milnera

- Podstawa dla kilku funkcyjnych języków programowania (Haskell, F#, Idris)
- Daje możliwość dedukcji typu przy zachowaniu silnego i statycznego typowania

Funkcja jako wartość

Kompozycja funkcji

```
-- | Function composition.
{-# INLINE (.) #-}
-- Make sure it has TWO args only on the left, so that it inlines
-- when applied to two functions, even if there is no final argument
(.)    :: (b -> c) -> (a -> b) -> a -> c
(.) f g = \x -> f (g x)
```

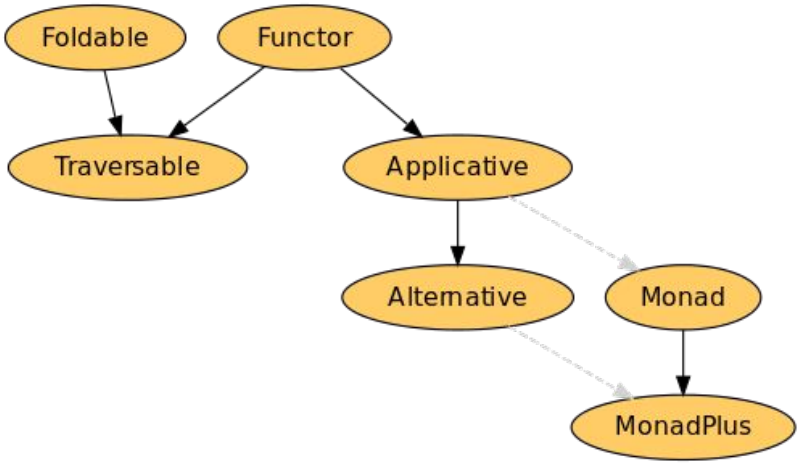
http://hackage.haskell.org/package/base-4.3.1.0/docs/src/GHC-Base.html

Co można zrobić z funkcją?

- Złożyć z inną funkcją
- Wykonać na argumentach
- Wykonać na argumentach w sparametryzowanym typie
- Umieścić w sparametryzowanym typie
- Kombinacje powyższych

Co to jest sparametryzowany typ?

Jakie może mieć właściwości?



Functor

- Definicja

```
class Functor f where
  fmap :: (a -> b) -> f a -> f b
```

- Przykład instancji

```
instance Functor (Either a) where
    fmap _ (Left x) = Left x
    fmap f (Right y) = Right (f y)
```

- Prawa

```
fmap id      = id
fmap (p . q) = (fmap p) . (fmap q)
```

Applicative

- Definicja

```
class Functor f => Applicative f where

    pure :: a -> f a

    (<*>) :: f (a -> b) -> f a -> f b

    (*>) :: f a -> f b -> f b

    (<*) :: f a -> f b -> f a

    (<$>) :: f => (a -> b) -> f a -> f b
```

- Prawa

```
pure id <*> v = v

pure (.) <*> u <*> v <*> w = u <*> (v <*> w)

pure f <*> pure x = pure (f x)

u <*> pure y = pure ($ y) <*> u
```

Typy a klasy typów

- Funkcje generyczne działają na klasach typów

- Możemy pokazać dany typ jest instancją jakiejś klasy typów
  - Nie musimy tego robić przy deklaracji typu

- Typ zyskuje w ten sposób właściwości i funkcje na których działamy

Funkcje a efekty uboczne i świat zewnętzny

Typy sparametryzowane odzwierciedlają to, co robi funkcja

- Czytanie z zewnątrz - Reader
- Pisanie nazewnątrz - Writer
- Oba powyższe - State
- Wejście wyjście - IO
- Strumienie danych - Source, Conduit, Sink
- Transakcyjna pamięć - STM

Niezmienne struktury danych

Dopasowanie do wzorca

- Struktura danych zawsze pamięta jak została stworzona
- Możemy użyć “konstruktora” jako mechanizmu do rozpakowania struktury
- Częściowo sprzeczne z kapsułkowaniem

Lenses aka soczewki

- Rozwiązują problem modyfikacji zagnieżdżonych i niemodyfikowalnych struktur danych

- “Getter” oraz “Setter” w jako jeden mechanizm

- Można je składać (kompozycja) w optykę

Programowanie generyczne

Polimorfizm podtypu vs parametyczny

```
public class Animal {
    public void what() {
        System.out.println("Animal");
    }
}

public class Cat extends Animal {
    public void what() {
        System.out.println("Cat");
    }
}
```

```
fmap :: Functor f => (a -> b) -> f a -> f

filter :: (a -> Bool) -> [a] -> [a]

isRight :: Either a b -> Bool

fmap (filter isRight)

  :: Functor f => f [Either a b] -> f [Either a b]
```

Przykłady podobieństwa w językach funkcyjnych

Haskell Quicksort

```
quicksort [] = []

quicksort (pivot:rest) = (quicksort lesser) ++ [pivot] ++ (quicksort greater)
    where
        lesser  = filter (<  pivot) rest
        greater = filter (>= pivot) rest
```

F# Quicksort

```
let rec quicksort = function
    | [] -> []
    | pivot::rest ->
        let lesser, greater = List.partition ((>=) pivot) rest
        List.concat [quicksort lesser; [pivot]; quicksort greater]
```

Scala Quicksort

```
def quicksort(list: List[Int]): List[Int] = list match {
    case Nil => Nil
    case pivot :: rest => {
        val (lesser, greater) = tail rest (_ < pivot)
        quicksort(lesser) ::: pivot :: quicksort(greater)
    }
}
```

Różnice

Klasy typów

- Haskell
- Scala

Dedukcja typów

- Haskell
- F#

Programowanie obiektowe

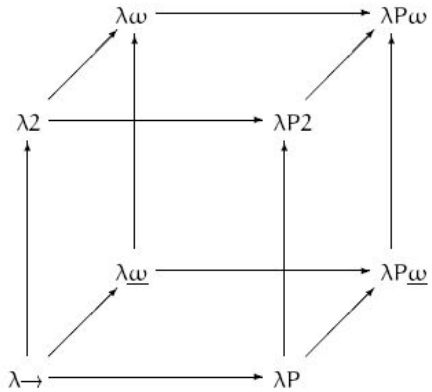
- Scala
- F#

Polimorfizm parametryczny oraz ad hoc funkcji

- Haskell

Typy wyższych rodzajów

- Haskell
- Scala



~~Haskell~~ Java User Group

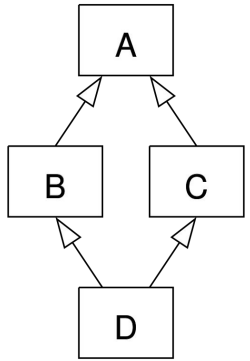
Java

Czego brakuje do podejścia funkcyjnego?

- Typów oraz abstrakcji:
  - Either
  - Try
  - Validation
  - Tuple*
- Konwersji
  - np Optional -> Stream
- Rozwijania funkcji
- Dopasowania wzorca
- Polimorfizmu ad-hoc (polimorfizm parametryczny istnieje od 1.5)

Czego brakuje do podejścia obiektowego?

- Domieszek (Mixin)
- Cech (Trait)
- Modułów**
- Klas zaprzyjaźnionych



Biblioteki - co jest gdzie?

- Krotka (aka Tuple)
  - Functional Java
  - Atlassian Fugue
  - Javaslang
- Suma rozłączna (aka Either)
  - Functional Java
  - Atlassian Fugue
  - Javaslang
- Walidacja
  - Functional Java
  - Javaslang
- Dopasowanie wzorca
  - Javaslang
- Memoizacja
  - Javaslang
- Soczewki
  - Functional Java
  - Atlassian Fugue
- Typ IO
  - Functional Java
- Typy Reader, Writer, State
  - Functional Java
- Parsery kombinatoryczne
  - Functional Java

Scala

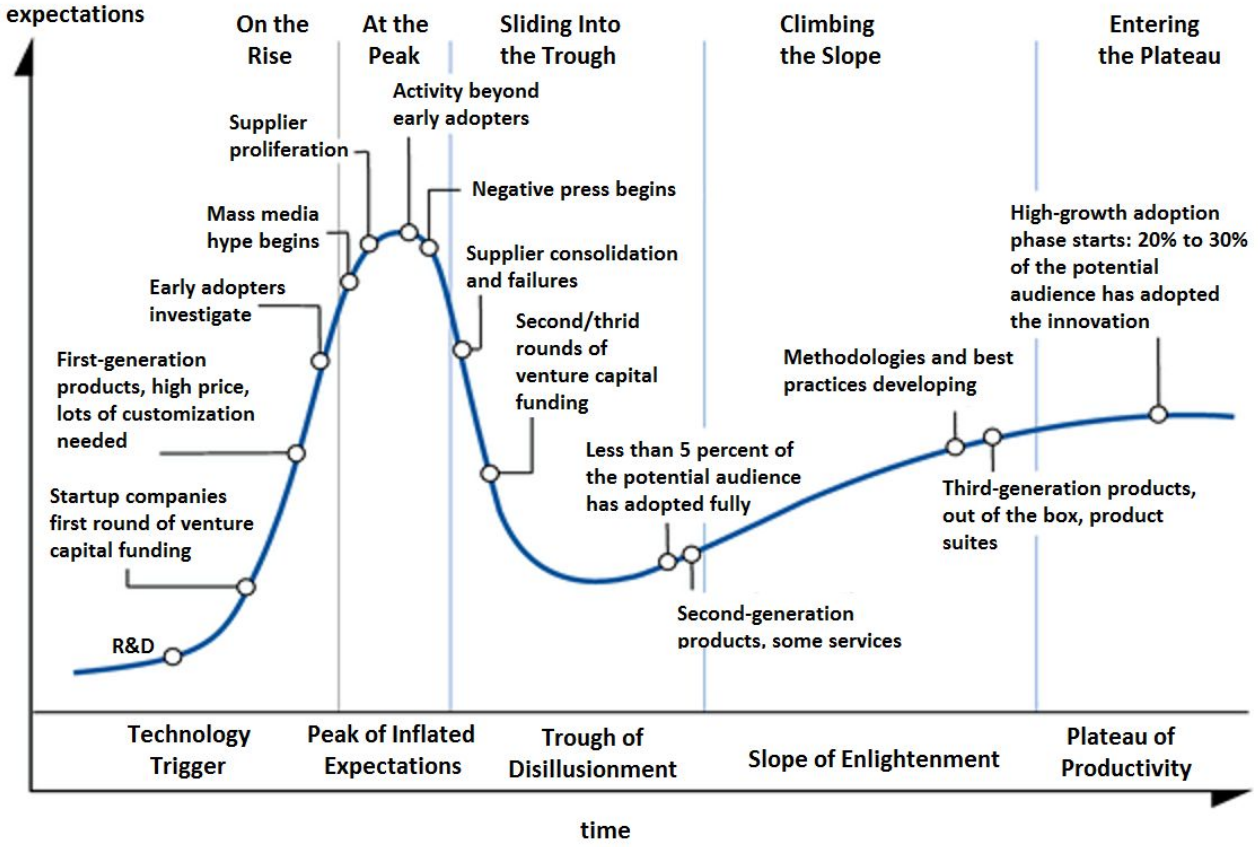
Dlaczego powstają takie biblioteki jak Scalaz czy Cats?

- Aby dodać brakujące typy/abstrakcje
  - Monada nie jest wprost zdefiniowana - metody flatmap i unit przychodzą z innych abstrakcji np Traversable

- Aby naprawić te zaimplementowane w języku inaczej niż uważają twórcy biblioteki
  - Czy Either powinien być stronniczy?

Czego warto się uczyć?

Hype cycle?


expectations
On the Rise
At the Peak
Sliding Into the Trough
Climbing the Slope
Entering the Plateau
Activity beyond early adopters
Supplier proliferation
Negative press begins
Mass media hype begins
Supplier consolidation and failures
Early adopters investigate
Second/thrid rounds of venture capital funding
First-generation products, high price, lots of customization needed
Methodologies and best practices developing
High-growth adoption phase starts: 20% to 30% of the potential audience has adopted the innovation
Less than 5 percent of the potential audience has adopted fully
Startup companies first round of venture capital funding
Third-generation products, out of the box, product suites
R&D
Second-generation products, some services
Technology Trigger
Peak of Inflated Expectations
Trough of Disillusionment
Slope of Enlightenment
Plateau of Productivity
time

Podsumowanie

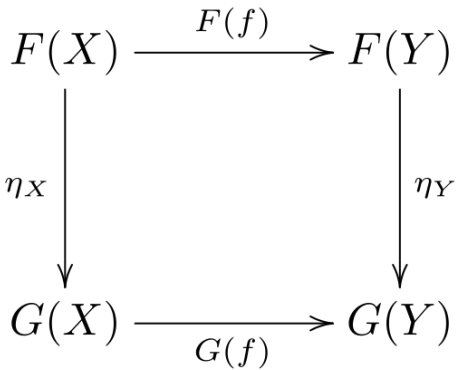
Trendy

- Wzrost znaczenia skalowalności
  - Coraz większe ilości danych
  - Większe ilości rdzeni
- Powstawanie języków specyficznych dla domen
- Przenoszenie odpowiedzialności z programisty na framework/bibliotekę
- Większe znaczenie poprawności programu, nawet kosztem wydajności

Co się raczej nie zdezaktualizuje

- Znajomość algorytmów i struktur danych
- Teoria typów i matematyka za nią stojąca

Dziękuję za uwagę



Referencje

"Functional programming with bananas, lenses, envelopes and barbed wire.", E Meijer

http://adit.io/posts/2013-04-17-functors,_applicatives,_and_monads_in_pictures.html

http://adit.io/posts/2013-07-22-lenses-in-pictures.html

Seria artykułów “What's Wrong with Java 8”, Pierre-Yves Saumont

https://github.com/mfejzer/tjug_26_haskell_examples

https://github.com/mfejzer/tjug_26_jvm_examples