# Задачи для НИР 2016

Денис Николаевич Москвин

СПбАУ РАН

10.02.2016

# План лекции

1. Канонические комбинаторы на уровне типов
2. Перепись населения типа

# План лекции

1. Канонические комбинаторы на уровне типов
2. Перепись населения типа

# Линзы ван Лаарховена

```
type Lens s t a b = forall f. Functor f =>
                    (a -> f b) -> s -> f t
```

Упаковка геттера и сеттера в такую линзу:

```
lens :: (s -> a) -> (s -> b -> t) -> Lens s t a b
lens get set = \ret s -> fmap (set s) (ret $ get s)
```

Например, линза для первого элемента пары строится так:

```
_1 :: Lens (a,c) (b,c) a b
_1 = lens (\(x,_) -> x) (\(_,y) v -> (v,y))
```

# Извлечение геттера и сеттера из линз

Геттер вынимается с помощью функтора `Const`:

```
newtype Const c a = Const {getConst :: c}
instance Functor (Const c) where
  fmap _ (Const v) = Const v

view :: Lens s s a a -> s -> a
view lns s = getConst (lns Const s)
```

Сеттер вынимается с помощью функтора `Identity`:

```
newtype Identity a = Identity {runIdentity :: a}
instance Functor Identity where
  fmap f (Identity x) = Identity (f x)

set :: Lens s t a b -> b -> s -> t
set lns b s = runIdentity $ lns (Identity . const b) s
```

# Композиция двух однопараметрических типов

```
newtype Cmps f g x = Cmps {getCmps :: f (g x)}

instance (Functor f, Functor g) => Functor (Cmps f g) where
  fmap f (Cmps x) = Cmps $ (fmap . fmap) f x
```

Это же верно для `Applicative`, `Foldable`, `Traversable`, но неверно для `Monad`.

# Законы для Traversable

```
traverse Identity ≡ Identity

traverse (Cmps . fmap g . f)
  ≡ Cmps . fmap (traverse g) . traverse f

foldMap f ≡ getConst . traverse (Const . f)
```

# Задача

- `Identity` — это комбинатор $\mathbf{I}$ на уровне типов.
- `Const` — это комбинатор $\mathbf{K}$ на уровне типов.
- `Cmps` — это комбинатор $\mathbf{B}$ на уровне типов.
- А что такое комбинатор $\mathbf{S}$ на уровне типов?

```
newtype S f g x = K {getS :: f x (g x)}
```

- Единственное упоминание тут: `https://hackage.haskell.org/package/ess-0.1.0.0/docs/doc-index.html`
- Задача: выяснить свойства `K`, и придумать, для чего его можно использовать.
- Обратить внимание на то, что `f :: * -> * -> *` и, соответственно, нужно смотреть на контексты `Bifunctor` и `Profunctor`.

# План лекции

1. Канонические комбинаторы на уровне типов
2. Перепись населения типа

# Проблема обитаемости типа

- Проблема обитаемости типа замкнутым термом для заданной системы типов: имеется ли терм $M$, такой что для известного $\sigma$ верно

$$\vdash M : \sigma$$

- Для Haskell есть Djinn

```
Djinn> ? x :: Monad m => m (m a) -> m a
-- x cannot be realized.
Djinn> ? x :: a -> a -> a
x :: a -> a -> a
x _ a = a
Djinn> ? n :: (a -> a) -> a -> a
n :: (a -> a) -> a -> a
n a = a
```

# Задача переписи населения

- Оказывается в просто типизированной лямбде задача обитаемости не просто разрешима, но и множество обитателей любого типа перечислимо.
- Задача переписи населения типа: реализовать Inhabitation Machine.

# Проблемы Djinn'a

- Djinn весьма умен

```
Djinn> ? l :: (Either a b -> c) -> (a -> c, b -> c)
l :: (Either a b -> c) -> (a -> c, b -> c)
l a = (\ b -> a (Left b), \ c -> a (Right c))
Djinn> ? l :: (a -> c, b -> c) -> (Either a b -> c)
l :: (a -> c, b -> c) -> Either a b -> c
l (a, b) c = case c of Left d -> a d
                       Right e -> b e
```

# Проблемы Djinn'a

- Djinn весьма умен

```
Djinn> ? l :: (Either a b -> c) -> (a -> c, b -> c)
l :: (Either a b -> c) -> (a -> c, b -> c)
l a = (\ b -> a (Left b), \ c -> a (Right c))
Djinn> ? l :: (a -> c, b -> c) -> (Either a b -> c)
l :: (a -> c, b -> c) -> Either a b -> c
l (a, b) c = case c of Left d -> a d
                       Right e -> b e
```

- Однако есть проблемы с однопараметрическими классами

```
Djinn> ? ($>) :: Functor f => f a -> b -> f b
($>) :: (Functor f) => f a -> b -> f b
($>) a b = fmap (\ _ -> b) a
Djinn> ? void :: Functor f => f a -> f ()
-- void cannot be realized.
```