**Федеральное агентство связи ордена Трудового Красного Знамени федеральное государственное бюджетное общеобразовательное учреждение высшего образования «Московский технический университет связи и информатики»**

Кафедра МКиИТ

Лабораторная работа №3

Выполнила:

Студентка

группы БВТ1702

Перевозова Ю.В.

Москва 2019

**Цель работы:**

Разобрать различные эффекты языка Haskell и написать программу, согласно заданию.

Выполнение работы:

1. Эффекты

1. Эффект частичности

Типы, которые могут вести себя как контейнеры по отношению

к другим типам, могут быть функторами. тип Maybe. Он может быть «пустой коробкой», и в этом случае имеет значение Nothing, или же в нём хранится какое-то одно значение, например, "ХА-ХА", и тогда он равен Just "ХА-ХА".

Вот как тип Maybe сделан функтором:

instance Functor Maybe where

fmap f (Just x) = Just (f x)

fmap f Nothing = Nothing

На практике также применяется типы Either и Except, реализующие тот же эффект, но позволяющий добавить информацию о том, почему вычисление не может быть завершено.

Пример.

data Either a b

= Left a -- условно ошибка

| Right b -- условно успех

data MyError = EmptyList

headE :: [a] -> Either MyError a

headE [] = Left EmptyList

headE (x:\_) = Right x

2. Эффекты недетерменированности (неопределённости)

2.1. Эффект нестабильности

Если процедура для одного и того же значения аргумента может вернуть от раза к разу разные результаты, это значит, что на самом деле результат зависит от чего-то ещё.

Даже генератор случайных чисел (настоящий, аппаратный) — это «чистая» функция, зависящая от состояния источника энтропии.

Чтобы представить этот эффект чистой функцией, надо всего лишь неявную зависимость сделать явной.

type Reader r b = r -> b

p :: a -> Reader r b

2.2. Эффект множественности

Здесь всё просто и очевидно. Функция, дающая много ответов сразу — это функция, имеющая единственный ответ-множество.

В Хаскеле есть тип Set для множеств, но для моделирования эффекта множественности оказывается более удобным список — [].

rollADie :: Int -> [Int]

rollADie n = [1..n]

3. Побочный эффект

Побочный эффект — это просто неявный результат. Сделаем же неявное явным!

p :: a -> (b, s)

newtype Putter s b = Putter (b, s)

p :: a -> Putter s b

2 + 3. Эффект состояния

Если соединить результат побочного эффекта и источник нестабильности, из их комбинации (композиции) получается эффект состояния — процедура, которая может и зависеть от текущего состояния «переменной», и задавать ей новое состояние.

Проведя рассуждения, аналогичные случаям Reader и Putter, получим

p :: a -> s -> (b, s)

newtype State s b = State (s -> (b, s))

p :: a -> State s b

0. Отсутствие эффектов (продолжение)

Рассмотрим тип

newtype Identity a = Identity a

Тип Identity a полностью аналогичен типу a. То есть это своего рода функция id, только на уровне типов.

Тип Identity не может выражать никаких эффектов. С другой стороны, можно сказать, что он выражает отсутствие эффектов.

Конечно же, конструктор типа Identity принадлежит Functor, Applicative, Monad и многим другим интересным и полезным классам.

4. Все эффекты

В Хаскеле есть специальный тип IO, реализующий сразу все возможные эффекты. В нём можно прерывать программу, обмениваться данными с ресурсами, не указанными явно в аргументах и возвращаемом значении. IO принадлежит Functor, Applicative, Monad и многим другим интересным и полезным классам.

**Задание:**

Опишите функцию, которая из исходного списка формирует список,содержащий только символьные атомы.

1. **Код программы**

|  |
| --- |
| import Data.List  import Data.Monoid  flst = filter (`elem`["N","Al","Br","Ba","H","Fe","I","K","Ca","O","Si",  "Mg","Cu","Na","S","Ag","C","P","Cl","Zn"]) lst |

1. **Работа программы**

