Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ №1
по дисциплине «Функциональное программирование»

1. Особенности функционального программирования. Побочные эффекты. Ленивые и аппликативные вычисления. Виды типизации.

2. Частичный порядок представлен списком пар. Написать на Haskell функцию `findEquiv`, находящую все эквивалентные друг другу элементы. Ответ должен представлять собой список списков эквивалентных элементов. Например, на входе

   `findEquiv [('A','B'),('B','C'),('D','E'),('C','A'),('E','D'),('F','F')]`

   получается результат (с точностью до перестановок в списках)

   `[['A','B','C'],['D','E'],['F']]`

3. Естественные преобразования.

4. Рассмотрим следующий алгебраический тип данных.

```haskell
data Trace a = Term a | Fork String (Trace a) (Trace a)
                          deriving (Show, Eq)
instance Functor Trace where
  fmap f (Term a) = Term $ f a
  fmap f (Fork s t1 t2) = Fork s (fmap f t1) (fmap f t2)
```

   Является ли следующая функция `f` естественным преобразованием?

```haskell
f (Fork name t1 t2) | (fmap (\x -> 0) t1) == (fmap (\x -> 0) t2)
                          = Nothing
                    | otherwise = f t1
f (Term a) = Just $ a
```

Билет рассмотрен и утверждён на заседании кафедры ИУ-9
Протокол №10 от 06.12.2021

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ №2
по дисциплине «Функциональное программирование»

1. Пары и размеченные объединения. Соответствие Карри–Ховарда для расширенного λ-исчисления. Типизация термов с конструкторами пары и альтернативы.

2. Построить λ-терм, имеющий следующий тип:
   $(((A \Rightarrow B) \Rightarrow B) \Rightarrow A) \Rightarrow (((A \vee (A \Rightarrow B)) \Rightarrow B) \Rightarrow B)$

3. Функторы в Haskell.

4. Рассмотрим следующий алгебраический тип данных.

```haskell
data Trace a = Term a | Fork String (Trace a) (Trace a)
                          deriving (Show, Eq)
```

   Является ли следующее определение корректным описанием функтора?

```haskell
instance Functor Trace where
  fmap f (Term a) = Term (f a)
  fmap f (Fork s1 (Fork s2 t1 t2) t3)
      = Fork s1 (fmap f (Fork s2 t1 t2)) (fmap f t3)
  fmap f (Fork s (Term t1) t2) = Fork s (Term t1) (fmap f t2)
```

Билет рассмотрен и утверждён на заседании кафедры ИУ-9
Протокол №10 от 06.12.2021

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ №3
по дисциплине «Функциональное программирование»

1. Унификация и алгоритм Хиндли для $\lambda_{\to}$.
2. Типизируется ли следующий терм?
   $\lambda x.x\ (\lambda y.y\ (\lambda z.(x\ ((snd\ z)\ (Right\ y)))))$
3. Карринг и сечения.
4. Рассмотрим следующий алгебраический тип данных.

```
data Trace a = Term a | Fork String (Trace a) (Trace a)
                            deriving (Show, Eq)
```

   Используя частичное применение либо сечения, построить проверку, существуют ли два листа (структуры `Term`) с одинаковыми метками, у которых есть хотя бы один предок (структура `Fork`), помеченный указанной строкой.

   Примеры:

   `eqLeafs "A" (Fork "A" (Term "a") (Fork "B" (Term "b")(Term "a")))` — это `True`

   `eqLeafs "A" (Fork "B" (Term "a") (Fork "A" (Term "b")(Term "a")))` — это `False`

Билет рассмотрен и утверждён на заседании кафедры ИУ-9
Протокол №10 от 06.12.2021

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ №4
по дисциплине «Функциональное программирование»

1. Соответствие Карри–Ховарда между $\lambda_{\rightarrow}$ и минимальной логикой.

2. Построить терм следующего типа:

   $(((A \Rightarrow B) \Rightarrow B) \Rightarrow A) \Rightarrow (((A \Rightarrow B) \Rightarrow A) \Rightarrow A)$

3. Базовые конструкции в Haskell (стражи, сопоставление с образцом, `where`, `let`, управление порядком вычислений).

4. Рассмотрим следующие типы данных

   ```
   data Trace a = Term a | Fork String (Trace a) (Trace a)
                                  deriving (Show, Eq)
   type Rule a = (String, Either a (a, String))
   ```

   `Trace` определяет деревья развёртки с ветвлением 2, а тип `Rule` задаёт правила переписывания, порождающие такие деревья. Первым элементом пары в этом типе стоит метка ветвления, вторым — структура порождаемого за 1 шаг элемента. Обратим внимание, что допустимы лишь два вида правил: порождение листа (т.е. порождение элемента `Term a`) либо порождение ветвления, у которого левый потомок — лист, а правый — ветвление.

   Дан список правил и первая метка ветвления. Построить структуру `Trace`, порождаемую этими правилами, применёнными ровно по разу, или объявить о несуществовании такой структуры.

   Послабление: для того, чтобы задача была зачтена, достаточно вернуть булевское значение, определяющее, возможна ли такая структура.

   Подсказка: это переформулировка одной классической задачи из теории графов, причём последнее применяемое правило определяется однозначно (а если оно не определяется, то нужной структуры быть не может). Подумайте, как.

   Примеры:

```
findTrace [("A", Right ("a", "B")),("A", Right ("b", "B")),("B", Left "s")] "A"
```

— это Nothing. В разбор метки "B"можно войти двумя разными способами, а из разбора "B"нельзя вернуться к разбору "A".

```
findTrace [("A", Right ("a", "B")),("A", Right ("b", "B")),
              ("A", Right ("c", "A")),("B", Left "s"),("B", Right ("a","A"))] "A"
```

— а вот тут ответ существует:

```
Just (Fork "A" (Term "a") (Fork "B" (Term "a")
      (Fork "A" (Term "c") (Fork "A" (Term "b") (Term "s")))))
```

Это один из возможных правильных ответов. Например, фрагмент дерева `Fork "A" (Term "c") (Fork "A" ...)` мог появиться и самым первым (тогда бы его не было потом).

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ №5

по дисциплине «Функциональное программирование»

1. Порядок редукции в λ-исчислении.

2. Построить λ-терм, реализующий комбинатор R, либо показать, что такого терма не существует

   $R\ (x\ y)\ (y\ x) = (x\ x)\ (y\ y)$

3. Монады и законы монад.

4. Рассмотрим следующий алгебраический тип данных с монадической структурой

```haskell
data Trace a = Term a | Fork String (Trace a) (Trace a) deriving (Eq,Show)

terminal (Term a) = a
terminal (Fork name t1 t2) = terminal t1

isTerminal (Term a) = True
isTerminal _ = False

instance Monad Trace where
  return a = Term a
  (Term a) >>= f = f a
  (Fork name t1 t2) >>= f | isTerminal (f (terminal t1))
                              = (Fork name (t1 >>= f) (t2 >>= f))
                          | otherwise
                              = (Fork (name ++ "Root") (t1 >>= f) (t2 >>= f))
```

Выполняются ли для этой структуры законы монад?

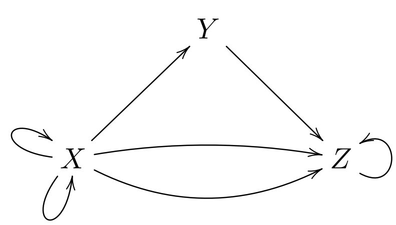
Билет рассмотрен и утверждён на заседании кафедры ИУ-9
Протокол №10 от 06.12.2021

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ №6
по дисциплине «Функциональное программирование»

1. Понятие категории.

2. Является ли категорией следующая диаграмма?



3. Свёртки (list comprehension).

4. Рассмотрим следующий тип данных

```
data Trace a = Term a | Fork String (Trace a) (Trace a)
                              deriving (Show, Eq)
```

Используя технику list comprehension, построить функцию, меняющую метки в структуре Fork на метки вида X1, X2, ... Равные метки должны заменяться на равные.

Пример работы:

`rename $`

`Fork "A" (Term "aaa") (Fork "B" (Fork "A" (Term "x") (Term "y")) (Term "z"))`
должна возвращать (с точностью до изменения порядка назначения индексов при X) выражение

`Fork "X1" (Term "aaa") (Fork "X2" (Fork "X1" (Term "x") (Term "y")) (Term "z"))`.

Билет рассмотрен и утверждён на заседании кафедры ИУ-9
Протокол №10 от 06.12.2021

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ №7
по дисциплине «Функциональное программирование»

1. Комбинатор неподвижной точки в λ-исчислении.
2. Построить λ-терм, реализующий комбинатор X, либо показать, что такого терма не существует

   $X\ (x\ (y\ x)) = x\ y$
3. Аппликативные функторы и их законы.
4. Рассмотрим следующий алгебраический тип данных:

```
data Trace a = Term a | Fork String (Trace a) (Trace a)
                          deriving (Show, Eq)
```

   Является ли нижеследующее определение корректным способом описать аппликативный функтор?

```
instance Applicative Trace where
  pure  = Term
  Term f <*> Term a = Term $ f a
  Term f <*> Fork name t1 t2 = Fork name (Term f <*> t1) (Term f <*> t2)
  Fork name f1 f2 <*> Term a = Fork name (f1 <*> Term a) (f2 <*> Term a)
  Fork name f1 f2 <*> Fork name2 a1 a2 = Fork name (f1<*>a1) (f2<*>a2)
```

Билет рассмотрен и утверждён на заседании кафедры ИУ-9
Протокол №10 от 06.12.2021

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ №8
по дисциплине «Функциональное программирование»

1. Бестиповое λ-исчисление. Три вида преобразований (α-, β-, η-). Правила связывания.

2. Привести λ-терм к нормальной форме либо показать, что её не существует:

$$(\lambda xy.(x\ y)\ (y\ x))\ (\lambda xy.y\ (y\ x))\ (\lambda xy.y\ (y\ x))$$

3. Функции высших порядков в λ-исчислении в в Haskell.

4. Рассмотрим следующий алгебраический тип данных.

```haskell
data Trace a = Term a | Fork String (Trace a) (Trace a)
                          deriving (Show, Eq)
```

   Требуется построить функцию `mergeTrace`, которая принимает на вход элемент типа `Trace` и функцию `fst` либо `snd`, и возвращает дерево, линеаризованное справа (в случае `fst`) или слева (в случае `snd`) — то есть такую `Trace`-структуру, у которой все правые (в случае `fst`) или левые (в случае `snd`) потомки представляют собой термы, помеченные конкатенацией строк в листьях соответствующих поддеревьев исходного терма.

   Подсказка: в качестве вспомогательной функции можно построить функцию, сворачивающую всю структуру целиком в единственный терм.

   Ещё одна подсказка: функции `fst` и `snd` будут типизированы так, к каким типам они применяются в самый первый раз в теле функции. Попытка использовать их для других типов приведёт к ошибке типизации. Поэтому нужно придумать, как подогнать все их применения под один и тот же тип.

   Примеры работы функции следующие. Следующий вызов:

```haskell
mergeTrace fst (Fork "A"
                  (Fork "D" (Term "aaa") (Term "bb"))
                  (Fork "B"
                    (Fork "C" (Term "x") (Fork "C" (Term "y") (Term "33")))
                    (Term "z")))
```

возвращает значение

```haskell
Fork "A" (Fork "D" (Term "aaa") (Term "bb")) (Term "xy33z")
```

А вызов:

```haskell
mergeTrace snd (Fork "A"
                  (Fork "D" (Term "aaa") (Term "bb"))
                  (Fork "B"
                    (Fork "C" (Term "x") (Fork "C" (Term "y") (Term "33")))
                    (Term "z")))
```

возвращает значение

```haskell
Fork "A" (Term "aaabb") (Fork "B" (Term "xy33") (Term "z"))
```

Билет рассмотрен и утверждён на заседании кафедры ИУ-9
Протокол №10 от 06.12.2021