Декларативное программирование

Семинар №5, группа 22215

Завьялов А.А.

3 октября 2022 г.

Кафедра систем информатики ФИТ НГУ

Разминка

Очень простой определенный интеграл

$$\int_0^{42} 42\,dx = 42\int_0^{42} dx = 42\cdot(x|_{x=42} - x|_{x=0}) = 42\cdot(42-0) = 1764$$

K-комбинатор

```
k :: a -> b -> a
```

```
k x y = x
```

Посчитать интеграл, используя функцию `integrate` из ДЗ №3

```
integrate ??? 0 42
```

Разминка

Очень простой определенный интеграл

$$\int_0^{42} 42\,dx = 42\int_0^{42} dx = 42\cdot(x|_{x=42} - x|_{x=0}) = 42\cdot(42-0) = 1764$$

K-комбинатор

```
k :: a -> b -> a
```

```
k x y = x
```

Посчитать интеграл, используя функцию `integrate` из ДЗ №3

```
integrate (k 42) 0 42
```

Разминка

Очень простой определенный интеграл

$$\int_0^{42} 42\,dx = 42\int_0^{42} dx = 42\cdot(x|_{x=42} - x|_{x=0}) = 42\cdot(42-0) = 1764$$

K-комбинатор (функция Prelude.const)

```
const :: a -> b -> a
```

```
const x y = x
```

Посчитать интеграл, используя функцию `integrate` из ДЗ №3

```
integrate (const 42) 0 42
```

Разминка

Очень простой определенный интеграл

$$\int_0^{42} 42\,dx = 42\int_0^{42} dx = 42\cdot(x|_{x=42} - x|_{x=0}) = 42\cdot(42-0) = 1764$$

K-комбинатор[1] (функция Prelude.const)

```
const :: a -> b -> a
```

```
const x y = x
```

Посчитать интеграл, используя функцию `integrate` из ДЗ №3

```
integrate (const 42) 0 42
```

[1] https://en.wikipedia.org/wiki/Combinatory_logic

Списки в Haskell

Списки в Haskell

```
Prelude> :info []
```

Списки в Haskell

```
Prelude> :info []

type [] :: * -> *

data [] a = [] | a : [a]
```

Списки в Haskell

```
Prelude> :info []

type [] :: * -> *

data [] a = [] | a : [a]
```

- Список — параметризованный тип данных;

Списки в Haskell

```
Prelude> :info []

type [] :: * -> *

data [] a = [] | a : [a]
```

- Список — параметризованный тип данных;
- *рекурсивный* тип данных;

Списки в Haskell

```
Prelude> :info []

type [] :: * -> *

data [] a = [] | a : [a]
```

- Список — параметризованный тип данных;
- *рекурсивный* тип данных;
- списком является:
  - пустой список (`[]`);
  - пара `x : xs`, где `x` (голова списка) — значение типа `a`, `xs` (хвост списка) — список элементов типа `a`, `(:)` — *конструктор* списка;

Списки в Haskell

```
Prelude> :info []

type [] :: * -> *

data [] a = [] | a : [a]
```

- Список — параметризованный тип данных;
- *рекурсивный* тип данных;
- списком является:
  - пустой список (`[]`);
  - пара `x : xs`, где `x` (голова списка) — значение типа `a`, `xs` (хвост списка) — список элементов типа `a`, `(:)` — *конструктор* списка;
- *гомогенная* структура данных.

Почему списки *так* устроены?

А вот в Python списки устроены совсем иначе! Почему создатели Haskell спроектировали основную структуру данных таким образом?

Почему списки *так* устроены?

А вот в Python списки устроены совсем иначе! Почему создатели Haskell спроектировали основную структуру данных таким образом?

- Списки в Haskell иммутабельные (неизменяемые), чисто функциональные[2];
- их удобно последовательно обрабатывать;
- прозрачно определены[3] в терминах языка[4];
- списки в Haskell могут быть *бесконечными*.

[2] Chris Okasaki — Purely Functional Data Structures, Cambridge University Press, 1998

[3] для каждой операции очевидна временная сложность и требуемый объём памяти;

[4] для сравнения, операции над списками в Python – builtins.

Базовые функции для работы со списками

Создание списков

- `[]` — пустой список;
- `(:)` — конструктор списка
  - **Пример №1**: `1 : []` ≡ `[1]`;
  - **Пример №2**: `1 : 2 : 3 : []` ≡ `[1, 2, 3]`.

Разделение списков (на "голову" и "хвост")

- `head` — первый элемент списка (если список не пустой);
- `tail` — список без первого элемента (если список не пустой).

Функции для работы со списками

- `length xs` — вычисляет длину списка `xs`;
- `reverse xs` — возвращает новый список с элементами `xs` в обратном порядке;
- `xs !! n` — возвращает *n*-й элемент `xs` (”головной” элемент списка доступен по индексу 0);
- `l1 ++ l2` — конкатенирует списки `l1, l2`;
- `take n xs` — возвращает новый список, состоящий из первых *n* элементов `xs`;
- `drop n xs` — возвращает новый список без первых *n* элементов `xs`;
- `splitAt n xs` — возвращает `(take n xs, drop n xs)`;
- `last xs, init xs`;
- и многие другие... (Data.List, Hoogle it!)

Сопоставление с образцом (pattern matching)

Определение

Метод анализа и обработки структур данных в языках программирования, основанный на выполнении определённых инструкций в зависимости от совпадения исследуемого значения с тем или иным образцом (Википедия).

- Туториал — `https://www.haskell.org/tutorial/patterns.html`;
- формальное определение — `https://www.haskell.org/onlinereport/exps.html#pattern-matching`.

99 Prolog/Lisp/Haskell/[Sample Language] Problems

- Исходная версия для Prolog — https://www.ic.unicamp.br/~meidanis/courses/mc336/2009s2/prolog/problemas/;
- версия для Lisp — https://www.ic.unicamp.br/~meidanis/courses/mc336/problemas-lisp/L-99_Ninety-Nine_Lisp_Problems.html;
- версия для Haskell — https://wiki.haskell.org/H-99:_Ninety-Nine_Haskell_Problems.

Функции высшего порядка для работы со списками

Три кита

- `map`
- `filter`
- `reduce`

И др.

- `zipWith`, `find`, `concatMap`, $\cdots$

Значимость?..

MapReduce, Java Stream API, LINQ (C#), `functools` (Python 3), Kotlin, JavaScript, *etc*

Бесконечные списки

Функции из модуля `Data.List`

- iterate
- repeat
- cycle

Бесконечные списки

Функции из модуля `Data.List`

- iterate
- repeat
- cycle

Но как это работает?

Немного про списковые включения[5]

- Позволяют обрабатывать сразу несколько списков;
- поддерживают сторожевые выражения (guards).

Но как это работает?

[5] https://wiki.haskell.org/List_comprehension

Немного про списковые включения[5]

- Позволяют обрабатывать сразу несколько списков;
- поддерживают сторожевые выражения (guards).

Но как это работает? Узнаем позже!

[5] `https://wiki.haskell.org/List_comprehension`

Домашняя работа №4

Q&A