Задание 1. Создание простых функций в Haskell

Предисловие

Файл “task-1.lhs” содержит корректную программу на языке Haskell, записанную в стиле “грамотного программирования”.

В программах, написанных в таком стиле, комментарии и исходный код меняются местами:

- Комментарии представляют из себя обычный текст, прямо как в текстовых файлах или заметках в формате Markdown.
- Исходный код отбивается специальным символом ‘>’ (точно как парой ‘–’ отбиваются комментарии в Haskell, или символом ‘#’ комментарии в Python).

Программы на Haskell, написанные в “грамотном” стиле, должны иметь расширение `.lhs`.

Скомпилировать или исполнить “грамотный” код в интерпретаторе можно ровно теми же способами, что и обычный:

```
# Компиляция через GHC
ghc task-1.lhs

# Запуск функции `main` в неинтерактивном интерпретаторе
runhaskell task-1.lhs

# Запуск GHCi и подгрузка в него модуля
ghci task-1.lhs
```

Подробнее о “грамотном” Haskell можно прочитать на Haskell Wiki, а о концепции грамотного программирования – на Википедии.

Пример. Вычисление предела последовательности с заданной точностью

Задан предел числовой последовательности:

$$\lim_{x \to \infty} \frac{x^2 - 1}{2x^2 - x - 1}.$$

Аналитически можно установить, что последовательность сходится к 0.5.

Зная, чему равен предел, мы можем *оценить*, как близко n-ый элемент последовательности приближается к нему:

$$e(n) = \frac{n^2 - 1}{2n^2 - n - 1} - 0.5.$$

Перед нами стоит задача написать функцию, вычисляющую некоторый элемент x исходной последовательности, что $e(x) < \varepsilon$, где $\varepsilon > 0$.

Базовое решение

Для начала напишем функцию, вычисляющую n-й элемент последовательности:

```
-- функция, вычисляющая n-й элемент последовательности
s n = (n^2 - 1) / (2*n^2 - n - 1)
```

Зафиксируем $\varepsilon > 0$ и будем варьировать параметр n на промежутке $[0, +\infty)$ до тех пор, пока разность `s n - 0.5` не станет меньше ε.

На императивном языке программирования, вроде Python, для этого мы могли бы написать что-то вроде:

```
n = 0
result = None
while True:
    result = s(n)
    if result - 0.5 < eps:
        break
    n += 1
```

Однако, в Haskell такой код мы написать ~~пока что~~ не можем – нет изменяемых переменных, да и циклов тоже нет...

Как же тогда писать код, повторяющийся произвольное количество раз?..

Используя рекурсию!

Функцию поиска члена последовательности, удовлетворяющего требованиям, можно представить в Haskell следующим образом:

```haskell
lim eps n =
  if ((s n) - (1 / 2)) < eps
    then s n
    else lim eps (n + 1)
```

На данном этапе рекомендуется загрузить данный файл в GHCi командой `ghci task-1.lhs` и вычислить функцию `lim` от разных аргументов:

- `lim 0.01 0`
- `lim 0.000001 0`
- `lim 0.0000001 0`
- `lim 0.0000001 10000000`

Рефакторинг

Функцию `lim` можно улучшить. Например, сейчас подвыражение `s n` встречается в теле функции два раза. Можно вынести его в локальную переменную с помощью конструкции let .. in ....

В `let` также можно вынести вычисление разности между `n`-м членом последовательности и её пределом:

```haskell
lim' eps n =
  let nth = s n
      estimation = (nth - (1 / 2))
   in if estimation < eps
        then nth
        else lim' eps (n + 1)
```

Также мы можем избавиться от `if`, используя сторожевые условия:

```haskell
lim'' eps n
  | estimation < eps = nth
  | otherwise = lim'' eps (n + 1)
  where
    nth = s n
    estimation = (nth - (1 / 2))
```

(Мы также использовали `where`, т.к. `let` нельзя использовать совместно с несколькими сторожевыми условиями)

Наконец, мы можем скрыть параметр `n` (всё равно он стремится к бесконечности), сделав функцию `lim''` локальной вспомогательной:

```
lim''' eps = helper 0
  where
    helper n
      | estimation < eps = nth
      | otherwise = helper (n + 1)
      where
        nth = s n
        estimation = (nth - (1 / 2))
```

(заметим, что `eps` не является параметром функции `helper`)

Задание 1. Сумма ряда геометрической прогрессии до n-го члена

n-й член геометрической прогрессии вычисляется по следующей формуле:

$$b_n = b_1 q^{n-1}, b_1 \neq 0, q \neq 0,$$

где b_1 – первый член прогрессии, q – знаменатель прогрессии.

Необходимо написать функцию, вычисляющую $\sum_{i=1}^{n} b_i$ для произвольного $n \geq 1$ (т.е. параметрами функции являются b_0, q, n).

Сделать это можно, как минимум, двумя способами:

- через списковое включение и функцию sum;
- через рекурсию*.

(* – такое решение, при верности исполнения, оценивается выше; рекомендуется сначала реализовать функцию через суммирование элементов списка, а затем попробовать написать рекурсивный вариант)

Задание 2. Сумма ряда геометрической прогрессии с заданной точностью ε

При $0 < q < 1$ геометрическая прогрессия является убывающей последовательностью, при $q < 0$ – знакочередующейся, при $q = 1$ – стационарной.

Если $|q| < 1$, то $b_n \to 0$ при $n \to +\infty$, сумма прогрессии $S_n \to \frac{b_1}{1-q}$ при $b \to +\infty$.

Предлагается написать функцию, вычисляющую сумму ряда геометрической прогрессии с заданной точностью ε.

Критерий точности ε может быть определён, как минимум, двумя способами:

- как ошибка при упрощении суммы прогрессии S суммой её n элементов S_n: $e(n) = S_n - S$;
- как “близость” $n-1$ и n-го элементов прогрессии: $e(n) = b_{n-1} - b_n$.

В обоих случаях вычисление завершается, когда $e(n) < \varepsilon$.

Рекомендуется обратить внимание на обработку крайних случаев, например, когда q задаёт возрастающую или стационарную последовательность, и на тот факт, что $\varepsilon > 0$.