Некоторые задачи

Косарев Дмитрий a.k.a. Kakadu

матмех СПбГУ

24 ноября 2018 г.

Оглавление

Outline

Задача за "Жизнь"

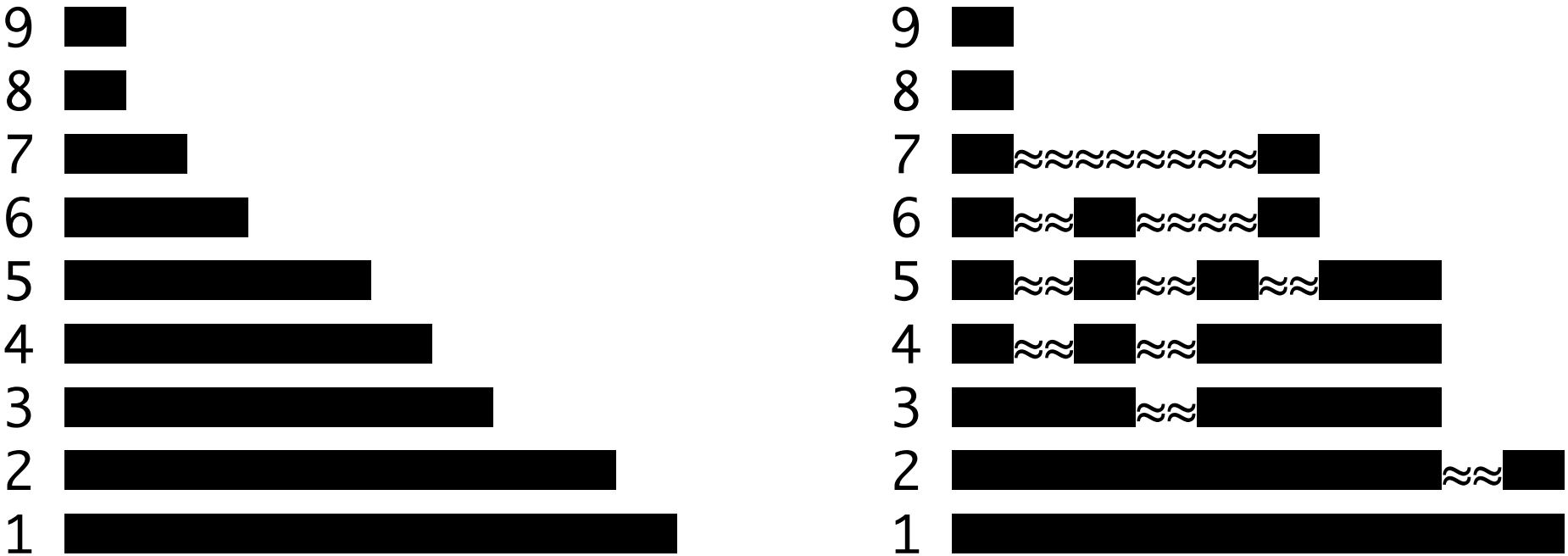
Реализуйте игру "Жизнь", а именно как эволюционирует и моргает поле. В итоге должен получиться бесконечный список состояний, которые будет принимать поле. Разумеется, поле надо уметь распечатывать, чтобы было видно как фигурки моргают, летят и т.д.

Вариации:

- 1мерный вариант – поле является лентой, неограниченной с двух сторон.
- 2мерный вариант – стандартный, как во всех книжках. Структура поля бывает разная
  - Ограниченный прямоугольник – самый примитивный вариант
  - Поле можно завернуть в тор – поинтересней
  - Честно, на неограниченном поле

P.S. Примеры живущих популяций – тут. А вот каноничное решение.

Посчитать сколько дождевой воды скопится в ямках



Это известная задача. Вопрос на засыпку – как она связана с предыдущими?

Outline

Random access list

N.B. Вспомните, что такое список, и какие там асимптотические характеристики добавления в голову и доступа к случайному элементу.

Скрестите деревья и списки, чтобы `cons` и `tail` работали быстро, а `!!` быстрее, чем со связными списками.

P.S. Когда я переводил книжку Окасаки, мне глава про эту штуку показалась наиболее интересной.

Red-black trees

Реализуйте красно-черные деревья

- На C++/C#/Java/любимом языке
- На Haskell
- В каком реализации проще накосячить? Почему?

Outline

Интерпретатор λ-исчисления (1/2)

У нас был `eval :: Term -> m Term`. Расширяйте язык, подбирая соответствующую монаду, вместо типовой переменной `m`.

- Реализуем интерпретатор с тремя видами термов как есть, используя монаду `Indentity`.
- Расширим язык операциями неоднозначности `noAnswer :: m Term` и `twoAnswers :: m Term -> m Term -> m Term`. Реализуйте с учетом того, что `m` – это список.
- Добавим сложение и константы в язык – у нас появилась возможность встретить сложение функций с числами, что должно вести к ошибке (аварийному завершению интерпретатора). `m` $\sim$ **`Maybe`**.

Интерпретатор λ-исчисления (2/2)

- Пускай сообщения об ошибках будут содержательными (нельзя складывать функции, переменная не объявлена). $\mathtt{m} \sim$ `Either String`.
- Интерпретатор можно переписать в continuation passing style используя монаду Cont

Задача про IO

Очень простая задача. Прочитать из файла каждую строчку и разбить по 8 байт. На выходе всё сложить в тип данных, что получилось распечатать.

```
data Chunk = Chunk    { chunk :: String }
           | LineEnd { chunk :: String
                     , remainder :: String }
     deriving (Show)
```

Делать всё можно через *классический интерфес* в стиле C или через lazy IO. Читать тут. Сравните два похода, какие плюсы и минусы?

Отчасти оффтоп: моноиды

Моноид – это множество M, на котором задана бинарная ассоциативная операция $\cdot$, и в котором существует нейтральный элемент e (т.е. $\forall x \in M e \cdot x = x = x \cdot e$).
Другими словами: моноид – это полугруппа с нейтральным элементом. Прримеры:

- Положительные целые числа, единица и умножение.
- Неотрицательные целые числа, ноль и сложение.
- Все строки, пустая строка и конкатенация строк.
- Все списки, пустой список и слияние списков.
- Тип `()` и вообще все типы с одним жителем.
- Если типы `a` и `b` моноиды, то произведение `(a,b)` тоже моноид.

Про wizard'ы a.k.a. "мастера"

Wizard'ы *вначале* задают вопросы, а если ответы хорошие и их не прервали, то они *в конце* что-то делают. Например:

1. Вопрос "Как Вас зовут?" и в конце ответ "Вас зовут так-то".
2. Вопрос "Сколько Вам лет?" и в конце ответ "Вам столько-то лет".
3. Предыдущие два вопроса объединены в wizard: сначала два вопроса, потом два сообщения.
4. "Удалить это файл?(да/нет) А этот?(да/нет) Этот?(да/нет)" и в конце удалить только те файлы, для которых сказали "да".

Реализуйте wizard'ы. Постарайтесь избежать копипасты.
Попробуйте устроить тут моноиды.

Outline

Про парсер-комбинаторы (1/3)

- Напишите функцию, которая разбирает строчки вида $a^n b^n c^n$, где x^n означает, что x повторяется n раз подряд. Буквы a,b,c захардкожены в парсер, n – произвольное.
- Попробуйте сделать то же самое с помощью LALR анализатора: *bison, FsYacc* или т.п.
- В языке OCaml (да и в C#) для строк можно избежать стандартного экранирования. `{|\"|}` $\sim$ `"\""`, `{asdf|"\n|asdf}` $\sim$ `"\"\\n"`, `{asdf||}|asdf}` $\sim$ `"|}"`. Напишите парсер, который преобразует “хитрую” строку в строку с прямолинейным escaping'ом.
- Попробуйте сделать то же самое с помощью стандартного лексического анализатора *flex, FsLex* или т.п.

Про парсер-комбинаторы (2/3)

- Вообще, что разобраться в парсер-комбинаторах скорее всего достаточно будет написать мини-парсер для языка разметки YAML. На сколько я помню, там сложности из-за используемых отступов.

Про парсер-комбинаторы (3/3)

Задача про Markdown и отчасти про монады.

В wiki разметке Markdown можно указывать ссылки как до, так и после их использования.

```
This is some text with [a link][foo]

[foo]: http://www.example.com

And hey, I can use [another link][foo] too!
```

Распарсьте Markdown так, чтобы на выходе был список из: либо 1) кусок текста типа `String` (или `Text`), либо 2) ссылка из пары строк: URL и как его отображать.

Outline

Язык, хороший по построению (1/2)

Опять расширим язык лямбд целыми числами, булевыми значениями и конструкцией if-then-else.

```
data Term =
    EInt Int
  | EBool Bool
  | EIfThenElse { cond       :: Term,
                  thenBranch :: Term,
                  elseBranch :: Term }
  | ELessThan Term Term
  | ... -- more constructors
```

А теперь мы хотим, чтобы первый аргумент `IfThenElse` представлял собой всегда терм, который вычисляется в `Bool`. Если там оказался какой-нибудь `EInt 199` – считать это ошибкой и падать (по аналогии с ошибками, когда мы пытались сложить не-числа).

Язык, хороший по построению (2/2)

На прошлом слайде были слабо типизированные термы. Надо переписать тип `Term` так, чтобы плохие конструкции `IfThenElse` нельзя было сконструировать.

Это не так просто сделать. Но я планирую рассказать про это позже. Кому интересно, любое из этих понятий поможет решить задачу.

- Leibniz type equality (можно найти в Typing Dynamic Typing, раздел 3)
- GADT (туториалы раз, два, три)

Вторая классическая задача про GADT

Опишите список с дополнительным типовым параметром “пустой” или “не пустой” (перекликается с задачей про фантомные типы).

Опишите список с дополнительным типовым параметром “длина”, используя типы **`data`** `Zero` и **`data`** `Succ n`. В итоге тип у функции `tail` должен быть `List (Succ len) a -> List len a`, потому что мы уменьшаем длину на 1, когда отщепляем голову.

P.S. Сделайте эти типы экземплярами класса `Applicative`

Задача 3 про GADT (1/2)

У нас две “классификации“ и примерно 11 ”сущностей“: 5 принадлежат одной классификации, 5 другой, одна сущность может сама выбирать какой она будет принадлежать.

Опишите необходимые типы и 2 функции, которые не важно что делают, но:

1. Принимают две ”сущности“, обе одинаковой ”классификации“
2. Принимают две ”сущности“, обязательно различной ”классификации“

Задача 3 про GADT (2/2)

Например: у нас две классификации, матмех и ПМПУ. Приматы есть и там, и там: астрономы – только на матмехе. Что есть только на ПМПУ – я хз. Внизу **не решение**:

```
data Specialization
  = SysProg     -- both
  | ApplMath    -- both
  | Astronomy   -- not exists on PMPU, only MM
  | PMPUOnly    -- not exists on MM, only PMPU

thesame :: Specialization -> Specialization -> ()
foo =
  let _ = thesame PMOnly Astronomy in ...
```

не годится, так как мы хотим передавать всегда специальности с одного и того же факультета, а тут мы передаем с разных. Хотелось бы, чтобы в таком случае была ошибка компиляции.

Outline

Печатаем дерево красиво

Как утилита `tree` из GNU или `ps`

```
$ ps -e --forest
1005 ?        00:00:00 lightdm
1019 tty7     00:02:41  \_ Xorg
1220 ?        00:00:00  \_ lightdm
2292 ?        00:00:01      \_ openbox
2357 ?        00:00:00          \_ ssh-agent
```

Предыдущую задачу можно *упростить*, забив на псевдографические палочки и рисуя отступы пробелами.

Предыдущую задачу можно *усложнить*, сделав рисование не прибитым к конкретной рисуемой структуре данных (выше — это дерево).

А ещё можно мастерскѝ реализовать что-то сильно напоминающее типобезопасный `sprintf`. А потом ещё и `scanf`. Типобезопасность в том смысле, что если в формате сказано печатать число, то надо передать агумент, который будет типа `Int`, а не какую-нибудь строку.

Пример

APIшечка, которая должна получиться.

```
tp1 = sprintf (lit  "Hello world")
-- "Hello world"

ts1 = sscanf "Hello world"  (lit "Hello world") ()
-- Just ()

tp2 = sprintf (lit "Hello "  ^  lit "world" ^ char)
              '!'
-- "Hello world!"

ts2 = sscanf "Hello World!"
             (lit "Hello " ^ lit "world" ^ char)
             id
-- Just '!'
```