Аннотация

Вот задачи. К первым 22 двум приписаны фамилии назначенных на неё решение. Также есть несколько дополнительных задач на случай, если кому-то совсем не понравится то, что попалось. Уведомляйте меня только, если хотите заменить задачу.

На некоторые задачи назначены по двое. Одни из них выглядят трудоёмко, на некоторые мне уважаемые люди ткнули пальцем и сказали, что туда лучше двоих. Изначально я планировал “раздвоить” только задачу про трансформеры монад – она действительно сложная, но зато интересная – она меня впечатлила за время моей аспирантуры больше, чем всё остальное прочитанное. В любом случае в В.Мясникова и Е.Орачева я верю, так как они присылали домашние задачи. К тому же были прецеденты, что человек справлялся с подобной штукой в одиночку.

Оправданно ли ставить на другие задачи по двое – я не знаю. Может это кому-то подсластит пилюлю.

Сдавать парные задачи я предлагаю последовательно – сначала один, потом второй. Желательно даже разным людям. В том году я практиковал конкурентную сдачу: один человек отвечал на вопросы, второй сидел рядом и кивал. У меня сложилось впечатление, что я был недостаточно строг.

В любом случае, я ожидаю, что вы будете ориентироваться в коде, которые будете мне показывать, независимо от того, водиночку вы это делали, парно или нашли код в гугле.

Для некоторых задач выделены обязательные критерии для претендования на четыре или пять. Эти критерии корректны, но условны в том смысле, что если в конце окажется, что вы не знаете законов монад (или т.п.), то вас мало что спасёт от второй попытки сдачи экзамена. Какой формат там будет я пока не думал.

Извините за то, что текст оформлен отвратительно.

Извините, если чью-то фамилию напечатал неправильно.

Задачи к экзамену

Косарев

19 января 2019 г.

Содержание

1 Кирилл. Автомат 3

2 Башкиров. Несколько задач про алгоритмы 3

3 DPLL (Усик) 4

4 Про две стратегии вычислений в одном интерпретаторе (Богданов Егор) 4

5 Scott Encoding (Гогина) 5

6 Простая система переписывания термов (Долгополова) 5

7 Мини-Prolog (Завадский) 6

8 Язык While (Келим) 7

9 SQLite на Haskell (Королихин) 7

10 Символьное интегрирование (Кутленков) 8

11 Мини-язык про трансформеры монад (Орачев + Мясников) 9

12 10

13 Предок delimited continuations – callCC (Осипова) 10

14 Про Delimited Continuations (Погребной) 10

15 Про трансляцию pattern matching в кучу If'ов (Решетников) 11

16 Про трансляцию pattern matching в кучу If'ов (Рыбина) 13

17 Корутины (Сергеев) 13

18 Шестиугольный тетрис (Богданов Евгений + Фунт) 13

19 14

20 Рисование резисторов в ASCII графике (Цырендашиев) 14

21 Задача про Cypher (Чернявский) 14

22 Lisp (Ярош) 15

23 Задача про ассемблер как DSL 15

24 Вывод типов в полиморфном лямбда исчислении с рекурсией 15

25 Задача про симуляцию мира 2D 16

26 Call-by-need 16

1. Кирилл. Автомат

2. Башкиров. Несколько задач про алгоритмы

2.1. Наименьшее натуральное число

Отыскать наименьшее натуральное число, отсутствующее в заданном конечном множестве натуральных чисел X. Множество X задано как список натуральных чисел без повторений, упорядоченных произвольно, например.

```
[ 8,23,9,0,12,11,1,10,12,7,41,4,14,21,5,17,3,19 ]
```

Ожидается решение линейной сложности

2.2. Сортировка попарных сумм

Пусть A это некоторое линейно упорядоченное множество, а операция $(+) :: A \to A \to A$ – это такая монотонная бинарная операция на A, что $x_1 \leq x_2 \wedge y_1 \leq y_2 \Rightarrow x_1 + y_1 \leq x_2 + y_2$. Рассмотрим задачу вычисления

```
sortsums :: [A] -> [A] -> [A]
sortsums xs ys = sort [ (x+y) | x <- xs , y <- ys]
```

Сколько операций потребует вычисление `sortsums xs ys`, если считать только сравнения и предполагать, что длина как `xs`, так и `ys` равна n? Разумеется, $O(n^2 log(n))$ сравнений достаточно. Всего имеется n^2 сумм, а сортировка списка длины может быть выполнена с использованием $(n^2 log(n))$ сравнений. Эта верхняя оценка не зависит от факта монотон ности $(+)$. На самом деле без дополнительной информации относительно $(+)$ и A эта граница является одновременно и нижней. Допущение о монотонности $(+)$ уменьшает лишь константный множитель, не изменяя при этом асимптотической сложности.

Но предположим теперь, что об операции $(+)$ и множестве A мы знаем больше: а именно, что $((+), A)$ является абелевой группой. Таким образом, операция $(+)$ ассоциативна и коммутативна, имеется единица и такая операция `negate :: A -> A`, что `x + negate x = e`. Вооружившись этой до полнительной информацией, один француз доказал, что `sortsums` можно вычислить за $O(n^2)$ сравнений.

2.3. Оценивание

3 Императивное решение обеих задач с использованием мутабельных массивов на Haskell

4 Чисто функциональное решение первой задачи (она проще)

5 Чисто функциональное решение обеих

3. DPLL (Усик)

Необходимо реализовать проверку формул языка высказываний на тавтологичность с помощью алгоритма DPLL. Необходимо реализовать парсер языка высказываний, непосредственно алгоритм и годные тесты. Я уже давал такую задачу в третьей пачке лекций, но тут надо решать именно конкретным алгоритмом. Про алгоритм можно читать много где, например на вики-статье с картинками.

4. Про две стратегии вычислений в одном интерпретаторе (Богданов Егор)

Реализуйте парсер и интерпретатор лямбда-исчисления с call-by-name (или call-by-need) стратегией вычислений. В языке поддерживаем списки, числовые константы, рекурсию. Протестируйте интерпретатор на факториале, числах Фибоначчи и прочих мелких тестах.

Реализуйте вывод простых типов получив в результате модифицированное дерево, где каждый терм языка заменен на пару из терма и типа. Тут есть тонкости, например, с тем, что у функции, которая вычисляет длину списка нет наиболее общего типа в STLC – надо будет либо сужать каким-то образом тип в конце (разумные умолчания); либо добавить в парсер аннотации типов, которые будет проставлять человек; либо проапгрейдиться до полиморфного лямбда исчисления, либо придумайте сами что-нибудь. Этот бы сделан, чтобы была возможность сформулировать следующий.

Идея взята отсюда. Сделайте интерпретатор, который некоторые куски считает CBN, а некоторые – CBV. Например, суммирование списка чисел лучше делать через CBV, а unfold числа `n` в `[1..n]` – наоборот,

так как вызывая каждый раз `(:)` он будет проверять что второй аргумент досчитан до конца и итоговая асимптотика будет квадратичная.

Я не против добавить сюда второго человека, потому что задача мне начала казаться почему-то сложнее чем раньше: думать надо когда можно спотимизировать, а когда не стоит.

5. Scott Encoding (Гогина)

Задача с Codewars

6. Простая система переписывания термов (Долгополова)

Переписывание термов – это область, которая включает большую часть теории лямбда-исчисления, а также предоставляет фреймворк, где упрощение и символьное дифференцирование задается просто как набор правил. У систем переписывания термов (term rewriting systems, TRS) и лямбда исчисления большинство терминологии совпадает. Например, мы говорим. что TRS завершается если для произвольного не существует бесконечной последовательности переписываний. Говорят, что TRS обладает свойством confluence если, всякий раз когда терм может редуцироваться в два других различных терма, существует ещё один терм, в который эти два терма редуцируются. Мы можем наблюдать, как эти два свойства выполняются для STLC (там это называется сильной нормализацией – strong normalisation). Далее, отношении редукции может задаваться в форме малого или большого шага (single-step or many-step form). В случае малого шага свойство confluence превращается в weak confluence.

В этой задаче нужно реализовать TRS систему, которая читает набор правил, а потом переключается в режим взаимодействия с пользователем. Каждый раз, когда пользователь что-то вводит, система пытается сопоставить введенное с правилами. Затем применяется первое правило, которое подходит, терм переписывается (rewritten) и печатается. Далее система продолжает повторять действие до тех пор пока сможет. Разумеется, не все наборы правил будут завершаться для произвольного входа.

Пример. Будем жить с сокращенным языком правил, а именно в форме

```
read books -> think
think -> sleep
sleep -> write books
find $1 -> read $1
book -> book
```

Для такого набора правил взаимодействие с системой может проходить примерно так: пользователь после приглашения > вводит запрос,

жмет ENTER и программа выдает процесс переписывания прочитанного терма.

```
Processed 5 rules.
> read books
-> think -> sleep -> write books.
> find book
-> read book -> read book -> read book -> ...
> find books
-> read books -> think -> sleep.
```

7. Мини-Prolog (Завадский)

Нужно минималистично реализовать язык логического программирования Пролог, состоящий из парсера, унификации и алгоритма поиска. Здесь у нас сокращенный вариант Пролога – без функциональных символов, поэтому унификация чрезвычайно простая.

Программа на Прологе представляется набором правил (clause).

```
mortal(X) :- human(X), lives(X).
```

Вот пример нерекурсивного (`mortal` не встречается справа) правила. Читается как "X смертен, если X – человек и X жив", т.е. `:-` – это импликация справа налево, а запятая – это конъюнкция (логическое "И"). Правило заканчивается точкой.

Во входной программе также может иметься набор фактов (правил без импликации):

```
human(teacher).
lives(teacher).
likes(bob,alice).
```

После того, как прочитан набор правил и фактов, человек может начать общаться с Пролог-системой. Получив приглашение `?-` человек вводит запрос, а программа печатает ответы: "да", "нет", "ответов не найдено" или "ответ такой-то. Искать дальше?"

```
?- human(machine).
no
?- human(teacher).
yes
?- likes(mary,john).
no
?- likes(bob,alice).
yes
```

Правила могут быть рекурсивными и содержать переменные. Задачей является реализовать интерпретатор Prolog на Haskell.

8. Язык While (Келим)

Реализовать парсер, интерпретатор и годные тесты языка While, в котором присутствуют

- Числа, арифметика, переменные и присваивание.
- Конструкции “запросить и пользователя число” и “распечатать число”.
- (Оценка 4) Конструкции IfThenElse и цикл с предусловием While
- (Оценка 5) Конструкции break и continue.
- Язык лямбд не нужно добавлять, хотя если вы будете дописывать разобранный в течение семестра интерпретатор, то их будет проще не выкидывать.

Хотя в языке нет функций, он достаточно богат, чтобы быть полным по Тьюрингу. А значит можно начать с простых программ (посчитать модуль числа, swap, факториал, Фибоначчи), а потом дополнить интересными программами.

Я нарочито не даю конкретный синтаксис, который надо распарсить. Если вам нравится синтаксис Pascal – берите его, если Си – то Си, и т.д.

9. SQLite на Haskell (Королихин)

Необходимо реализовать минисистему баз данных. Программа должна уметь дампить информацию в файл, восстанавливать из файла и выполнять (парсер + интерпретатор) запросы к базе данных. Для ввода-вывода данных парсер и принтер писать не нужно, можно обойтись более прямолинейным стандартным способом (тот, кто копался в стандартных классах типов – меня поймет). Парсер нужен только для языка запросов SQL. Список запросов возьмем сокращенно-стандартный. Конкретный синтаксис посмотрите в документации к какому-нибудь варианту SQL, здесь я напишу только несколько примеров.

- Создание таблиц. Из типов давайте оставим только Int и String (который в базах данных обычно называется VarChar)

```
CREATE TABLE table1 ( String FirstName
                    , String LastName
                    , Int Id
                    , Int Age)
```

- Добавление данных в таблицу. Если кто-то добавляет Int туда, где ожидался тип String – выругиваться

```
INSERT INTO table1 VALUES ('vasya','pupkin',1,2),
                          ('ivan', 'ivanov',2,2)
```

- Выбор данных из таблицы с выдачей табличного результата

```
SELECT * FROM  table1
```

или

```
SELECT (FirstName,LastName) FROM table1 WHERE Age>18
```

- Удаление данных из таблицы

```
REMOVE FROM table1 WHERE Age>18
```

- Join (он же inner join) таблиц, который формально является декартовом перемножением всех строчек в таблицах с последующей фильтрацией

```
SELECT (a, tableX.id, tableY.id) FROM tableX
   JOIN tableY
   ON table1.id = table2.somekey
```

или даже вложенные join'ы

```
SELECT * FROM A
  JOIN (B JOIN C ON B.fkC = C.pk)
  ON A.optionalfkB = B.pk
```

Для оценки 4 сделайте пункты выше.
Для оценки 5 нужно

- Оптимизатор запросов: вложенные join'ы должен вычисляться не в стиле generate&filter, а как-нибдуь более оптимально.
- Поддержите в языке и интепретаторе кроме обычных join'ов другие: LEFT, OUTER, CROSS. Они работают чуть-чуть по-другому.

10. Символьное интегрирование (Кутленков)

Научитесь брать аналитически интегралы вида $\int \frac{P(x)}{Q(x)}dx$, где P и Q – многочлены, степень $Q \leq 2$. Чтобы это сделать, надо уметь

- Решать квадратные уравнения.
- Разбивать большую дробь на маленькие, решая по дороге систему линейных уравнений.
- Может быть делить многочлены с остатком
- Интегрировать простые дроби табличными интегралами

Напишите также тесты для интегратора.
Я нарочито не добавляю сюда требование про парсера, чтобы не раздувать задачу на двоих.

11. Мини-язык про трансформеры монад (Орачев + Мясников)

Эта задача без парсеров.

Сделайте AST для мини-языка и "интерпретатор" для него. Тип термов мини-языка должен называться `M m r`, где

- `M` – это название вышележащей монады;
- по типу вложенной монады `m` язык будет полиморфен;
- `r` – типовый параметр вложенной монады. В некотором смысле, что что мы описываем должно быть изоморфно `M (m r)`.

```
class Transformer t where
  promote :: (Monad m) => m a -> t m a
  observe :: (Monad m) => t m a -> m a
```

Функция создания терма мини-языка из вложенной монады будет называться `promote`, а вызов интерпретатора и возвращение ответа будет называться `observe`.

Для этой задачи будем считать, что вышележащей монадой является монада, которая позволяет сэмулировать вброс исключения, где исключение всегда типа `String`.

Что надо сделать (выполнять желательно в указанном порядке):

1. Опишите тип для мини-языка как алгебраический тип. Очевидно, там должны присутствовать конструкторы-аналоги для `return`, `>>=` и для "вброса исключения". Плюс, конструкции специфичные для вышележащей монады (она нам известна). (Подсказка, вроде как должно быть 5 конструкторов). Никаких лямбд, арифметики и парсеров добавлять не нужно. Сделайте также `instance Show`.
2. Напишите интерпретатор миниязыка (полиморфный по внутренней монаде) и какие-нибудь входные программы для него.
3. Покажите, что то, что получилось – это монада (реализуйте три функции: `return`, `>>=` и `fail`). Естественным подходом будет использование композиции уже написанного интерпретатора и/или конструкторов, например `return = Return`
4. Проверьте, что законы монад выполняются.
5. Какие дополнительные, специфичные для вышележащей монады, законы разумно ввести?
6. (Для оценки 4) В получившемся языке получилось некоторое количество конструкторов. Возможно некоторые конструкторы стоят всегда рядом? (или можно применить законы и сделать так, чтобы они стояли всегда рядом) Сократите количество конструкторов с пяти до четырёх, перепишите интерпретатор.

7. (Для оценки 5) Попробуйте каким-нибудь образом переписать интерпретатор так, чтобы можно было избавиться от необходимости введения конструкторов для описания структуры мини-языка (что по сути означает отсутствие необходимости задавать тип для мини-языка), а ограничиться только функциями и композицией функций. (Если вы переписали интерпретатор так, что справа от символа равенства не встречаются конструкторы – вы на пути к успеху).

12.

13. Предок delimited continuations – callCC (Осипова)

Такая же задача как следующая, только надо реализовать обыкновенные (undelimited) continuations в лямбда исчислении, они же callCC (call-with-current-continuation). Конструкция Shift называется callCC, конструкции reset – нет, захват continuation'а происходит до самого конца.

Короче, я обещал задачу про delimCC ещё в начале семестра. Скорее всего вы её сделали. Выкинуть одну конструкцию, чутка исправить интепретатор и сделать новые тесты большого труда составить не должно.

14. Про Delimited Continuations (Погребной)

Это прошлогодняя задача, которую я обещал повторить и в этом году.

Реализовать парсер и интерпретатор бестипового лямбда-исчисления с поддержкой арифметики, списков и delimited continuations. Соответственно, во входном языке должны быть следующие конструкции: три для лямбда-исчисления, числовые константы и арифметические операции, IfThenElse, cons и nil для списков, операции isNil, операции head & tail. Рекурсии нет.

Если что-то вычисляется не от того, что задумано (складываются лямбды, вызывается isHead от чисел, а не от списков), то интерпретатор должен упасть с объяснением почему. Условие в конструкции If вычисляется и проверяется на адекватность: числа ≥ 1 – истина, 0 – ложь, остальное – ошибка интерпретации.

Так как это прошлогоднее задание, приделайте к тому, что получилось дома годные тесты.

Полезную информацию стоит искать неподалёку отсюда и где-то здесь.

15. Про трансляцию pattern matching в кучу If'ов (Решетников)

В этой задаче нужно написать интерпретатор мини-языка, трансляцию мини-языка в мини-язык и тесты. Парсера писать не нужно.

У нас есть язык бестипового лямбда-исчисления без рекурсии. Язык расширяем:

- Числами, арифметикой.
- IfThenElse. Если в условии If число 0 – это ложь; 1 – истина; иное считается ошибкой интерпретации.
- Бинарная операция равенства двух чисел. Если числа равны, то 1; не равны – 0. Если сравниваются не числа, то это ошибка интерпретации. В принципе, если будет удобнее, то можно также добавить сравнение строк на равенство (смотрите ниже про конструкцию `tag`).

Тип для термов языка должен получиться похожим на этот. Разумеется, можно облегчить себе жизнь с помощью индексов де Брауна, но не обязательно.

```
data Op = Add | Mil | Div | Sub | Eq
data Term = App Term   Term
          | Abs String Term
          | Var String
          | IntConst Int
          | BinOp Op Term Term
          | ...
```

Далее расширяем язык "алгебраическими" выражениями.

- Выражения "алгебраического" вида: конструктор с именем (строка с заглавной буквы), а за ним $n \geq 0$ выражений.
- Функция для работы с "алгебраическими" данными `tag` – взятия тега (имени). Операцию `tag` можно сделать по-разному: возвращать строку и сравнивать строки, возвращать хэш строки и сравнивать числа, или выберите что-нибудь другое...
- Функция для работы с "алгебраическими" данными `field` – она ещё часто называется проекцией. Возвращает n-й аргумент конструктора (нумерация с нуля). Определена только для алгебраических значений, падает если значение не состоит из конструктора или если у конструктора нет аргумента с таким номером.

  Пример/ `Just 5` во входном языке после парсинга превращается в `Constructor "Just"[IntConst 5]` (обозначим это за `t`), т.е. конструктор с именем "Just" и одним аргументом `IntConst 5`. Для него `Tag t` должен вычисляться в строку "Just" (или хэш от ней). `Field 0 t` – вычисляется в `IntConst 5`, проекции с другими номерами ведут к ошибке.

- Конструкцией паттерн-матчинга как в Haskell, но без pattern guards, patterns synonyms, т.е. минимально простой вариант.
- Из паттернов разрешены: wildcard, переменная, числовые константы, имя конструктора а за ним $n \geq 0$ паттернов (вложенные паттерны было бы неплохо поддержать), паттерны для списков не поддерживаем.
- Выводом типов не занимаемся, если что-то вызвалось не от того, что нужно, то интерпретатор должен закончиться с аварийно, сообщая что именно пошло не так.

```
data Term = ...
          | Constructor { tname :: String
                        , targs :: [Term] }
          | Tag Term
          | Field Int Term
          | Case { what  :: Term
                 , patts :: [(Pattern, Term)] }

data Pattern = PVar String
             | PWildcard
             | PConst Int
             | PConstructor { pname :: String
                            , pargs :: [Pattern] }
```

Пример. После синтаксического анализа входная строка

```
\ x -> case x of
         Nothing -> 18
         Just _  -> x
```

должна превратится в терм

```
Abs "x" $
  Case (Var "x")
       [ (PConstructor "Nothing"       [], IntConst 18)
       , (PConstructor "Just" [PWildcard], Var "x")
       ]
```

Что надо сделать:

1. Интерпретатор языка (который описан выше) с поддержкой ошибок и подсчетом количества вычисленных конструкций IfThenElse. Стратегию вычислений (call-by-value, call-by-name, call-by-need или другое) выберите сами.
2. Реализуйте преобразование из языка выше в язык, где конструкции pattern matching заменены на IfThenElse и те две дополнительные функции `field` и `tag`.
3. Преобразование выше можно делать “в лоб” – тогда породится большое количество If’ов, или по-умному. Например, на каждом шаге

смотреть на 1й столбец матрицы паттернов, группировать одинаковые и запускать генерацию кода рекурсивно на получившихся подматрицах. Страшная ссылка с описанием подробностей (скачайте её будучи на матмехе, она может оказаться недоступной с айпишников необразовательных подсетей).

(Оценка 5) Сделайте также оптимизированное преобразование и проверьте, что количество вычисленных If'ов сокращается.

16. Про трансляцию pattern matching в кучу If'ов (Рыбина)

Это кусок предыдущий задачи в том смысле, что я одну большую распилил на две.

Здесь нужно написать парсер и интерпретатор для языка из предыдущей задачи. Часть про интерпретатор общая для обеих задач.

Синтаксис входного языка выберите сами: как в Haskell, F#, OCaml, Scala, двумерный/недвумерный, ... Мне всёравно что вы выберите, выбирайте то, для чего сможете написать парсер. Сложность скорее всего будет заключаться в выборе синтаксиса и парсинга паттернов. С алгебраическими опреациями `field` и `tag` особых проблем быть не должно – они сами распарсятся в применения соответсвующих функций (это будет почти то, что нужно).

17. Корутины (Сергеев)

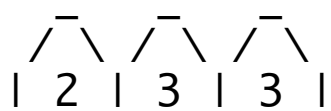
Эта и следующая (Scott Encoding) задача с сайта Codewars. Вроде как, если у пользователя есть большой рейтинг, то ему могут показать решение, но я думаю, что вы не поднимете его до нужного уровня за выходные.

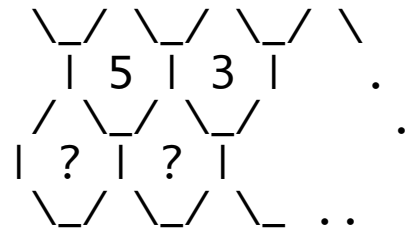
Задача с Codewars

У меня есть смутное подозрение, что что-то подобное я рассказывал.

18. Шестиугольный тетрис (Богданов Евгений + Фунт)

Поле размера m на n (числа должны настраиваться) состоит из сотов, которые выглядят примерно как на рисунке. Три сота сочленяются в одной точке, в этом месте соты можно вращать (всего три степени свободы). Если в какой-то момент в трех соседних сотах находятся одинаковые цифры (от нуля до девяти), то количество очков увеличивается (формулу придумайте сами), соты уничтожаются, соты сверху осыпаются вниз, сверху генерируются новые.





Нужно написать симуляцию мира, где на каждом шаге происходит распечатка поля в ASCII графике и вращение с целью максимизировать очки. Когда нельзя увеличить количество очков, то симуляция заканчивается.

Разумеется, никаких парсеров в этой задаче писать не обязательно.

19.

Уважаемый человек сказал, что задачу про тетрис надо давать на двоих. Я поверил на слово.

20. Рисование резисторов в ASCII графике (Цырендашиев)

Дана электрическая схема состоящая из резисторов последовательно или параллельно соединенных. Надо научиться считать общее сопротивление схемы (это просто) и красиво рисовать схему в ASCII графике (с этим будут проблемы).

21. Задача про Cypher (Чернявский)

Есть такая графовая база данных Neo4j и к ней язык запросов к графам под названием Cypher. Грубо говоря, в вершинах и на ребрах написаны некоторые метки, запрос делается к графу и возвращается подграф подходящих вершин, соединенных подходящими рёбрами. Нужно реализовать парсер подмножества Cypher, интерпретатор и тесты для него.

Довольно много мелких примеров для тестов есть на официальном сайте. Меня в первую очередь интересуют в языке поиск подграфа с учетом меток на вершинах и ребрах, прочие фичи можете отложить на потом (т.е. навсегда).

То, что выше – для оценки 4.

Для оценки 5 я хочу, чтобы граф-ответ выглядел красиво. Можете, например, распечатывать его в формат dot с помощью хаскеллевой библиотеки, а потом стандартной утилитой перегонять в картинку PNG.

22. Lisp (Ярош)

Нужно написать парсер, интерпретатор и тесты для входного языка à la Lisp. Там довольно прямолинейный синтаксис из s-выражений: всё состоит из вложенных списков, вызов функции записывается в префиксной форме, код = данные и т.д.

Напишите парсер входного языка в стиле Lisp (это проще, чем вы думаете), интерпретатор и тесты. В языке должны быть: списки и операции над ними (иначе это не Lisp), арифметика и конструкции лямбда-исчисления.

Также нужно реализовать Lisp-специфичные конструкции Quote и Unquote, которые преобразуют запускаемый код в данные и наоборот. С помощью такой штуки можно запускать квайны (quines): программы, которые возвращают свой код, т.е. вычисляются сами в себя. А потом twine'ы: пары программ, которые вычисляются друг в друга. И даже thrine'ы: тройки программ, где натравливание одной на другую возвращает третью (и так ещё двумя способаим сочетания программ). Если что-то из Найнов заработает – можете претендовать на оценку 5.

23. Задача про ассемблер как DSL

Задача без парсеров. Нужно сделать только AST, интерпретатор и тесты.

Реализовать встраиваемый в Haskell язык, похожий на ассемблер. (Похожий, потому что настоящий ассеблер X86 слишком большой). Требуется некоторое количество регистров, стандартные конструкции арифметики (`ADD`, `SUB`, `MUL`, `DIV`), команды работы со стеком (положить из такого-то регистра на стек и прочитать со стека в такой-то регистр). Попробуйте сделать что-то типа вызова подпрограмм/процедур, которые читают входные данные из захардкоженных регистров (или стека) и возвращают ответ сходным образом.

Для оценки 5 поддержите конструкции "переход на метку, если в таком-то регистре лежит 0". Это получится очень легко, если знать/-нагуглить то, что нужно.

Короче Free Monad всех спасет.

24. Вывод типов в полиморфном лямбда исчислении с рекурсией

Здесь нужно написать алгоритм вывода полиморных типов и тесты к этому алгоритму. Парсер писать не обязательно, но скопипастить оный из задачи про delimCC может улучшить читаемость тестов.

Чтобы типы были более интересные, в языке должны быть константы с типом `Int`. На тип `Bool` и арифметику давайте забьем – так как в этой задаче интерпретатор языка делать не надо.

Разумеется, процедура вывода полиморфных типов должна адекватно работать для полиморфных функций, мономорфных функций и `let`-полиморфизма.

Хорошая реализация алгоритма работает в среднем за линию, но если получится экспонента – ничего страшного. Требование – сделайте так, чтобы к каждому подвыражению большого выражения также приписывался его тип (я уверен, что вы сможете так пофиксить свои типы, чтобы это можно было сделать).

25. Задача про симуляцию мира 2D

Описание нарочито неформальное, чтобы фанатам был доступен простор для фантазии

Нужно реализовать симуляцию мира, населенного NPC. Каждый шаг симуляции сопровождается распечаткой состояния мира на консоль. Симуляция идет потенциально бесконечно.

В идеале должна быть процедурная генерация всего и вся.

В звёздной системе вращаются планеты, которые потребляют или генерируют товары. Между ними летают корабли, которые будут покупать/перевозить/продавать товары между планетами, чтобы срубить по больше “денег”. Сделайте симуляцию процедурно-генерируемого мира (оценка 4)

На оценку 5 надо добавить какие-нибудь свои фичи: раздачу квестов на отвезти что-либо куда-либо, квесты для охотников за головами, или ещё что-нибудь.

Не больше одного человека.

26. Call-by-need

Реализуйте интерпретатор лямбда исчисления с рекурсией и стратегией вычислений call-by-need, а также парсер и тесты к нему.