А. П. Немытых

Лекции по языку программирования Рефал

Мы представляем конспекты лекций по функциональному языку программирования Рефал. Лекции были прочитаны автором в 2006 году в г. Переславле-Залесском. Библ. 10 наим.

Содержание

§ 1. Данные языка программирования Рефал (Лекция №1)

РЕкурсивный Функциональный АЛгоритмический язык РЕФАЛ[1] был разработан В. Ф. Турчиным как универсальный метаязык программирования в 60-х годах. Первая реализация появилась в 1968 г. Рефал – строгий язык первого порядка, основанный на алгорифмах Маркова, имеет два неизоморфных конструктора; один из которых – приписывание (конкатенация) – ассоциативен, а другой – унарный – используется для построения структуры произвольного дерева. Ассоциативность приписывания превращает множество данных Рефала в моноид. Структура этого моноида отражается в синтаксисе программ и операционной семантике Рефала.

1.1. Уравнения в свободной полугруппе. *Моноидом* называется полугруппа с единицей.

Пусть M – непустое множество. Тогда *свободным моноидом*, порождённым M, называется множество всех конечных последовательностей (включая пустую последовательность) элементов из M с полугрупповой операцией, задаваемой приписыванием последовательностей, т. е.

$$\{a_1, \ldots, a_n\} \circ \{b_1, \ldots, b_m\} = \{a_1, \ldots, a_n, b_1, \ldots, b_m\}$$

Далее пустую последовательность будем обозначать *ничем* (отсутствием какого либо знака).

Рассмотрим свободный моноид, порождённый множеством букв Кириллицы и латинского алфавита. Назовём этот моноид *алфавитной полугруппой*. Элементы алфавитной полугруппы будем писать в одинарных кавычках. Например, `'ы'`. Слово, заключённое в одинарные кавычки, будем считать сокращением записи последовательности букв, входящих в это слово. Например,

`'полугруппа'` = `'п' 'о' 'л' 'у' 'г' 'р' 'у' 'п' 'п' 'а'`

Задача 1. Решить уравнения в алфавитной полугруппе (где x, y и z – переменные):

а). x `'п'` z = `'полугруппа'`;
б). $x^2 = xx =$ `'abababab'`;
в). x^2 `'a'` $y =$ `'aabaabaab'`;

Иногда удобно выделять последовательности, состоящие из одного элемента. Перед именами переменных с областью допустимых значений (О.Д.З.) – множеством одноэлементных последовательностей будем приписывать приставку «`s.`», перед именами переменных с О.Д.З., совпадающим со всей полугруппой, будем приписывать приставку «`e.`» Например: `s.x`, `e.y` .

[1]Название, данное В. Ф. Турчиным, – алгоритмический язык рекурсивных функций. Мы предпочитаем другую расшифровку этого сокращения, избегая перестановки. Не существует общепринятой традиции, какими буквами – прописными (и сколькими) или строчными – писать название языка. В. Ф. Турчин также использовал разные варианты синтаксиса этого названия. Мы будем писать Рефал, делая редкие исключения.

Задача 2. Решить уравнения в алфавитной полугруппе

а). `e.x s.y 'п' e.z = 'полугруппа'`;
б). `s.y e.x`^{3} `s.z = 'abababab'`;
в). `e.u s.z`^{2} `e.x`^{2} `e.y = 'aabaabaab'`;

Теперь рассмотрим уравнения с параметрами. Аналогично переменным, параметры будут двух типов, соответствующих введённым типам переменных. Чтобы отличать параметры от переменных, мы будем помечать параметры знаком диез «#». Например: `#s.p` – параметр типа «s» с именем «p», `#e.q` – параметр с именем «q» и О.Д.З., совпадающим со всей полугруппой.

Задача 3. Решить уравнения с параметрами в алфавитной полугруппе

а). `e.x s.y 'п' e.z = 'пол' #s.p 'группа'`;
б). `e.x 'a' = 'a' #e.p`;
в). `e.y 'a' e.x`^{2} `= 'ab' #s.p #s.p 'abaab'`;

Задача 4. Рассмотрим свободный моноид, порождённый множеством из `'0'` и `'1'`. Придумать в этом моноиде уравнение с параметрами `#s.p`, `#s.q`, `#s.r`

$$\ldots = \texttt{\#s.p \#s.q \#s.r}$$

такое, что одно из его переменных есть `s.d` и решение этого уравнения имеет вид

$$\ldots, \texttt{s.d = '0'}, \ldots$$

тогда и только тогда, когда последовательность `#s.p #s.q #s.r` содержит больше нулей, чем единиц.

1.2. Системы уравнений в свободной полугруппе. Рассмотрим систему из двух уравнений в алфавитной полугруппе

$$\begin{cases} \texttt{e.x e.y e.z = 'abbabaab'} \\ \texttt{e.y s.u = 'bab'} \end{cases}$$

Скобки (и) не принадлежат алфавиту. Алфавитная полугруппа ассоциативна, что позволяет нам опускать эти скобки, не указывая порядок исполнения операции приписывания в полугруппе. Таким образом, скобки (и) можно использовать как метасимволы – для иных целей. Воспользуемся этим замечанием, чтобы формально записать систему уравнений более кратко:

e.x e.y e.z (e.y s.u) = 'abbabaab' ('bab')

Аналогично можно записать любую систему уравнений в алфавитной полугруппе. Например, систему

$$\begin{cases} \texttt{e.y}_1\ \texttt{'н' e.n 'учи' e.y}_2 = \texttt{'Какая' 'ночь' 'Мороз' 'трескучий'} \\ \texttt{e.z}_1\ \texttt{'н' e.m 'учи' e.z}_2 = \texttt{'На' 'небе' 'ни' 'единой' 'тучи'} \\ \texttt{e.x}_1\ \texttt{s.k e.x}_2\ \texttt{s.k e.x}_3 = \texttt{'Как' 'шитый' 'полог' 'синий' 'свод'} \end{cases}$$

можно записать так

```
  e.y1 'н' e.n 'учи' e.y2 (e.z1 'н' e.m 'учи' e.z2) (e.x1 s.k e.x2 s.k e.x3) =
'Какая' 'ночь' 'Мороз' 'трескучий'
('На' 'небе' 'ни' 'единой' 'тучи') ('Как' 'шитый' 'полог' 'синий' 'свод')
```

или так

```
e.y1 'н' e.n 'учи' e.y2 (e.z1 'н' e.m 'учи' e.z2 (e.x1 s.k e.x2 s.k e.x3))
=
'Какая' 'ночь' 'Мороз' 'трескучий'
('На' 'небе' 'ни' 'единой' 'тучи' ('Как' 'шитый' 'полог' 'синий' 'свод'))
```

Задача 5. Решите данные выше уравнения.

Пусть G – свободный моноид. Тогда *древовидной полугруппой* D, порождённой G, назовём наибольшее множество, которое можно получить из элементов G посредством конечного числа применений бинарной операции приписывания и унарной операции заключения в скобки. То есть, $G \subset D$ и

1. если $d_1 \in D$ и $d_2 \in D$, то $d_1 \circ d_2 = d_1 \ \ d_2 \in D$;
2. если $d \in D$, то $(d) \in D$

Задача 6. Докажите, что древовидная полугруппа является полугруппой с единицей относительно операции приписывания.

Пусть Ω – алфавитная полугруппа, тогда правая часть последнего, приведённого выше, уравнения принадлежит к древовидной полугруппе, порождённой Ω. Выражение

```
() (() ())
```

принадлежит любой древовидной полугруппе. Более того, любая правильная скобочная структура (выражение, состоящее только из скобок) принадлежит любой древовидной полугруппе.

Задача 7. *(*числа Каталана): Сколько существует различных правильных скобочных структур из n пар скобок?

1.2.1. *Леса корневых деревьев.* *Ориентированным графом* будем называть тройку $\Gamma = \{V, E, I\}$, состоящую из конечного множества *вершин* V, конечного (возможно, пустого) множества *рёбер* E и *отображения инцидентности* $I : E \to V \times V$, сопоставляющего каждому ребру упорядоченную пару вершин (*начало* и *конец* ребра), которые это ребро соединяет.

Последовательность ребер $e_1, \ldots, e_n$ ориентированного графа назовем *путем*, если начало каждого последующего ребра является концом предыдущего.

Корневым деревом назовем ориентированный граф Γ с выделенной вершиной r (*корнем*), в которую не входит ни одно ребро и из которой существует единственный путь в любую другую вершину. Кроме того, для каждой вершины из Γ множество исходящих из неё рёбер упорядочено.

Задача 8. Докажите, что в корневом дереве всякая вершина, отличная от корня, имеет ровно одно входящее в неё ребро.

Вершина называется *узлом*, если из неё выходит не менее одного ребра. Множество вершин, не являющихся узлами, разобьём на два непересекающихся множества; элементы одного из них назовём *листьями*, элементы другого – *срезами*. Каждому листу припишем некоторое имя.

Пусть дано множество имён $\mathcal{A}$. *Лесом* корневых деревьев $F_{\mathcal{A}}$ назовём множество всех конечных последовательностей (включая пустую последовательность) корневых деревьев, листья которых поименованы элементами из $\mathcal{A}$.

Деревья будем изображать растущими от корня вниз. Каждому выражению с правильной скобочной структурой можно сопоставить лес корневых деревьев. Например,

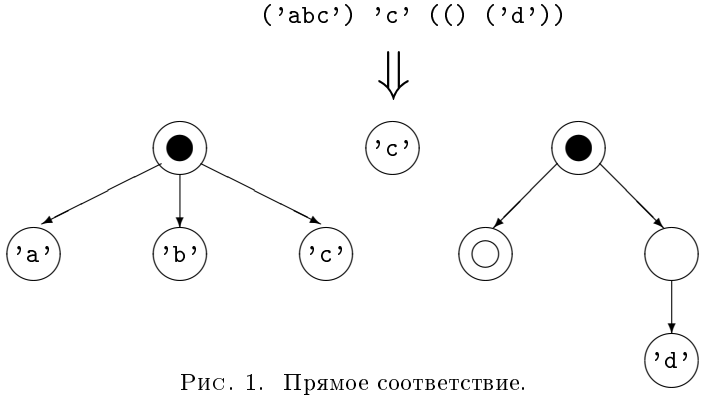

Рис. 1. Прямое соответствие.

Обратно, каждому лесу корневых деревьев можно сопоставить выражение с правильной скобочной структурой.

Задача 9. Опишите обратное соответствие.

1.3. Домашнее задание.

Задача 10. Решить уравнения в алфавитной полугруппе (где x и y переменные)

а). y `'a'` $x^2 =$ `'abbabaab'`;

б). x `'a'` $=$ `'a'` x.

Задача 11. Решить уравнения в алфавитной полугруппе

а). `e.y s.z e.x`^{2} `= 'abbabaab'`;

б). `s.x 'a' = 'a' s.x e.y`.

Задача 12. Решить уравнения с параметрами в алфавитной полугруппе

а). `s.y e.x`^{3} `s.z = 'abab' #e.p`;

б). `e.x`^{2} `'a' e.y = #e.p #e.p 'aab'`.

Задача 13. Дан свободный моноид G, порождённый множеством из букв `'a'` и `'b'`. В G рассматривается уравнение с параметрами:

`e.y e.x`^{2} `e.z = #s.p #s.q #s.r #e.u`

Докажите, что это уравнение имеет решение такое, что длина последовательности `e.x` больше нуля.

Задача 14. Последовательность w_n определяется индуктивно по правилу:

$$w_0 = \texttt{'a'}, w_1 = \texttt{'ab'}, w_{n+2} = w_{n+1}\ w'_{n+1},$$

где w'_{n+1} получается из w_{n+1} заменой всех букв `'a'` на `'b'` и всех букв `'b'` на `'a'`. Например, $w_2 =$ `'abba'`, $w_3 =$ `'abbabaab'`. Пусть $u = w_k$ для некоторого k. Решить уравнения: 1) `e.y 'a'`^{3} `e.z =` u ; 2) `e.y 'b'`^{3} `e.z =` u .

Задача 15. Установите на вашем компьютере одну из реализаций диалектов языка Рефал.

§ 2. Функции индуктивные по Кушниренко (Лекция №2)

РЕФАЛ – РЕкурсивный Функциональный АЛгоритмический, по определению, есть рекурсивный язык программирования. Это означает, что основным выразительным инструментом программирования на Рефале является рекурсия (от recursion – повторение). Понятие рекурсии тесно связано с понятием индукции по структуре данных: Рефал-программу можно часто понимать как доказательство некоторого утверждения о данных методом математической индукции. Мы рассмотрим индукцию по данным, построенным с помощью конструктора приписывания.

2.1. Функции индуктивные по Кушниренко. Скажем, что функция $F : M \to K$ *конструктивна*, если существует алгоритм, который $\forall x \in M$ вычисляет значение $F(x)$. Ниже под функциями мы будем понимать только конструктивные функции.

Понятие алгоритма является *начальным* (*неопределяемым*), как, например, понятия точки и множества в геометрии.

Рассмотрим свободный моноид G, порождённый некоторым множеством M. Пусть N – некоторое множество. Функция $f : G \to N$ называется *индуктивной*, если её значение на последовательности $x_1\ \ldots\ x_n$ можно вычислить по её значению на последовательности $x_1\ \ldots\ x_{n-1}$ и по x_n, то есть существует функция $F : N \times G \to N$, для которой

$$f(x_1\ \ldots\ x_n) = F(f(x_1\ \ldots\ x_{n-1}), x_n)$$

Отметим, что 1) f есть функция от одного аргумента, который является конечной последовательностью, и 2) длина n этой последовательности не дана, хотя и дан её последний элемент x_n (если он существует).

Таким образом, мы получаем конкретный алгоритм вычисления значений функции f, если известно её значение $f()$ на пустой последовательности.

Например, $\sum_{i=1}^{n} x_i$ – сумма всех членов конечной числовой последовательности является индуктивной функцией. Мы считаем, что сумма членов пустой последовательности равна 0.

Не все функции $f : G \to N$ являются индуктивными.

Индуктивным расширением функции $f : G \to N$ называется такая *индуктивная* функция $g : G \to K$, что существует функция $Q : K \to N$ такая, что

$$f(x_1 \ \dots \ x_n) = Q(g(x_1 \ \dots \ x_n))$$

Мы снова получаем конкретный алгоритм вычисления значений функции f.

Задача 1. Определить, являются ли следующие функции индуктивными, доопределив их, если необходимо, естественным образом на пустой последовательности. Указать индуктивные расширения для неиндуктивных функций:

а). Функция, сопоставляющая данной последовательности $x_1 \ \dots \ x_n$ перевёрнутую последовательность: $x_n \ \dots x_1$.

б). Дана последовательность $y_1 \ \dots \ y_k$ десятичных цифр, где $y_1 \neq 0$, если $k > 1$. Функция, сопоставляющая последовательности $x_1 \ \dots \ x_n$ десятичных цифр, где $x_1 \neq 0$ (если $n > 1$), произведение целых чисел $\overline{x_1 \dots x_n} * \overline{y_1 \dots y_k}$.

в). Среднее арифметическое последовательности целых чисел.

г). Число элементов последовательности целых чисел, равных её максимальному элементу.

д). Максимальная длина монотонного (неубывающего или невозрастающего) отрезка из идущих подряд элементов в последовательности целых чисел.

е). Максимальная длина возрастающей подпоследовательности данной последовательности целых чисел.

2.2. Примеры программ на языке Рефал. *Языком программирования* L называется тройка $\{D, P, S\}$, где D – множество *данных*, P – множество *программ* и функция *семантики* $S : P \times D \to D_\perp$ (где $\perp \notin D$ и $D_\perp ::= D \cup \{\perp\}$), описывающая алгоритм вычисления конкретной программы $p \in P$ на конкретном данном $d \in D$, то есть $S(p, d) = p(d)$. Причём $p(d) = \perp$ тогда и только тогда, когда p попадает в состояние аварийной остановки или в бесконечный цикл.

Пусть фиксирован некоторый алфавит $\mathcal{A}$, элементы которого мы будем называть символами. Множество данных Рефала D может быть определено индуктивно:

- если $\mathtt{d}_1 \in D$, тогда $\mathtt{(d}_1\mathtt{)} \in D$;
- если $\mathtt{d}_1 \in D$ и $\mathtt{d}_2 \in D$, тогда $\mathtt{d}_1 \ \mathtt{d}_2 \ \in D$;
- пустая последовательность является данным;
- $\forall a \in \mathcal{A} \Rightarrow a \in D$.

Подобные определения принято кратко записывать так:

$\mathtt{d} \ \mathtt{::=} \ \mathtt{d}_1 \ \mathtt{d}_2 \ \mid \ \mathtt{(d}_1\mathtt{)} \ \mid \ a \ \mid \ \mathit{empty}$

$\mathit{empty} \ \mathtt{::=}$

, где знак «::=» читается «по определению равно», а знак «|» есть логическая связка «или».

Алфавит Рефала (множество *символов*) мы будем описывать по ходу дела – постепенно расширяя его. Данные Рефала принято называть *объектными выражениями*. *Объектным термом* Рефала называется символ или объектное выражение, заключённое в скобки. Таким образом, любое объектное

Рефал-выражение есть конечная последовательность объектных Рефал-термов (включая пустую последовательность).

Любой читабельный символ, который позволяет ввести клавиатура компьютера, заключённый в одинарные кавычки, есть символ Рефала. Например: `';'`, `' '`, `'b'`. Исключение составляет сама одинарная кавычка; её необходимо экранировать: `'\''`.

Слово называется *палиндромом*, если европейское прочтение этого слова (слева направо) совпадает с его арабским прочтением (справа налево). Слово «палиндром» в переводе с греческого языка означает бегущий вспять.

Предикатом на множестве M называется функция φ со значениями в двухэлементном множестве $\varphi : M \to \{$`'T', 'F'`$\}$. Здесь `'T'` означает истину (True), а `'F'` – ложь (False).

Ниже дано рекурсивное определение на Рефале предиката на множестве слов: его значение на данном конкретном слове-аргументе w_0 равно `'T'` тогда и только тогда, когда это слово есть палиндром.

```
Palindrome {
      =  'T';
 s.x  =  'T';
 s.x e.middle s.x = <Palindrome e.middle>;
 e.word = 'F';
}
```

Третье предложение сравнивает на равенство первую и последнюю буквы слова w_0 и, если они совпадают, w_0 есть палиндром тогда и только тогда, когда палиндромом является слово, полученное из w_0 отбрасыванием первой и последней буквы. Первые два предложения описывают *базисные случаи индукции*: первое – для слов чётной длины, второе – для слов нечётной длины. Процесс проверки заканчивается, так как на каждом шаге длина слова уменьшается и она ограничена снизу нулём.

Задача 2. Определить предикат палиндрома на языке Haskell и сравнить это определение с данным выше.

Понятие рекурсии тесно связано с понятием математической индукции.

Рассмотрим рекурсивное определение функции, переворачивающей данную последовательность термов. О.Д.З. переменной типа «`t.`» есть множество объектных Рефал-термов.

```
Reverse {
            = ;
 t.x e.rest = <Reverse e.rest> t.x;
}
```

Мы 1) переворачиваем последовательность, полученную из данной последовательности отбрасыванием первого элемента, и 2) приписываем к результату переворачивания сзади этот первый терм-элемент. Длина сокращённой последовательности на единицу меньше длины исходной, следовательно, повторяя процесс шаг за шагом, мы придём к пустой последовательности, результат переворачивания которой есть она сама.

Задача 3. Определить аналогичную функцию на языке Haskell и сравнить это определение с данным выше.

Задача 4. Привести пример программы, которая работает бесконечное время.

Решим на Рефале следующую задачу. Кузнечик прыгает по прямой линии на один метр, каждый раз случайно выбирая направление прыжка (вправо или влево). Будем наблюдать за ним, записывая его выбор последовательностью букв `'L'` (влево) и `'R'` (вправо). Определить, на каком расстоянии от начальной позиции будет находиться кузнечик, когда он остановится.

Если в последовательности прыжков есть рядом стоящие символы `'LR'` или `'RL'`, то они не влияют на ответ и их можно выкинуть из последовательности – далее повторить сокращения, в противном случае вся последовательность состоит из одинаковых букв; их число есть расстояние кузнечика от стартовой позиции – вправо или влево.

Буквальное повторение этого предписания на Рефале выглядит так:

```
Jump {
   e.1 'LR' e.2 = <Jump e.1 e.2>;
   e.1 'RL' e.2 = <Jump e.1 e.2>;
   e.1 = e.1;
}
```

Здесь в качестве имён переменных мы использовали числа. Отметим, что второе предложение можно переставить выше первого – определение останется правильным. Третье же предложение переставить нельзя.

Задача 5. Решить задачу про кузнечика на языке Haskell и сравнить это решение с данным выше.

2.2.1. *Как выполнить программу на языке Рефал.* Пусть дана программа p. Функция семантики $S(p, d) = p(d)$, по определению, может быть вычислена только тогда, когда фиксированы оба её аргумента. Программу мы уже фиксировали. Данное – второй аргумент функции семантики в Рефале – можно задать, например, посредством Рефал-определения с выделенным именем **Go**; именно с этого определения алгоритм, реализующий функцию семантики, начинает рассматривать Рефал-программу (принято говорить, что **Go** является *входной точкой* программы).

Чтобы подчеркнуть, что мы будем общаться с программой именно через эту функцию – явно укажем это в синтаксисе *ключевым словом* `$ENTRY`:

```
$ENTRY Go {
  = <Palindrome 'долог голод'>;
}
```

Мы зафиксировали аргумент предиката `Palindrome`, и теперь его значение может быть вычислено на вычислительной машине, но результат вычисления для нас останется неизвестным. Чтобы компьютер напечатал результат на экране, необходимо попросить его об этом:

```
$ENTRY Go {
  = <Prout <Palindrome 'долог голод'>>;
}
```

Здесь в правой части предложения стоит композиция двух действий: вычислить значение `Palindrome` и напечатать результат вычисления. В результате исполнения этой программы на экране компьютера появится строка: `T`.

2.3. Домашнее задание.

Задача 6. Определить, являются ли следующие функции индуктивными, доопределив их, если необходимо, естественным образом на пустой последовательности. Указать индуктивные расширения для неиндуктивных функций. Дать определение этих функций на Рефале.

а). Максимальный отрезок идущих подряд одинаковых элементов последовательности.

б). Даны две последовательности $x_1 \ \ldots \ x_n$ и $y_1 \ \ldots \ y_k$. Функция для всех $m \leqslant n$ сопоставляющая последовательности $x_1 \ \ldots \ x_m$ максимальное число $j \leqslant k$ такое, что последовательность $y_1 \ \ldots \ y_j$ есть подпоследовательность последовательности $x_1 \ \ldots \ x_m$.

Задача 7. Будем все слова записывать только строчными буквами. *Обобщённым палиндромом* назовём фразу (или последовательность фраз), которая становится палиндромом, если из неё выкинуть все знаки препинания и пробелы. Определить на Рефале предикат на множестве текстов, множество истинности которого совпадает с множеством обобщённых палиндромов.

Задача 8. Даны текст и строка. Определить на Рефале функцию, значение которой есть отрезок данного текста, непосредственно следующий за *первым вхождением* данной строки, если такое вхождение существует, и сам текст, если данная строка в него не входит.

Задача 9. Даны текст и строка. Определить на Рефале функцию, значение которой есть отрезок данного текста, непосредственно следующий за *последним вхождением* данной строки, если такое вхождение существует, и саму строку, если она в данный текст не входит.

Задача 10. В унарной системе счисления натуральные числа представляются счётными палочками. Например, `'III'` = 3. Определить на Рефале функции унарной арифметики: сложения, вычитания, умножения и деления с остатком двух натуральных чисел.

§ 3. Унарный конструктор построения дерева. Рефал-машина. (Лекция №3)

Рефал – строгий язык первого порядка – имеет два неизоморфных конструктора; один из которых – приписывание – ассоциативен, а другой – унарный –

используется для построения структуры произвольного дерева. Нас будет интересовать базисный Рефал (подмножество Рефала), в котором процесс вычисления программы может быть представлен в виде композиции простых действий, называемых шагами Рефал-машины.

Язык программирования называется *строгим* (или *аппликативным*), если значение функции $F(x, \ldots, y)$ не вычисляется, пока не вычислены все её аргументы $x, \ldots, y$. Примером нестрогого (*«ленивого»*) языка является язык программирования Haskell.

Задача 1.

а). Приведите пример программы p на языке Haskell, формальный перевод которой на Рефал (то есть изменяется только синтаксис, но не изменяется структура программы) существенно повышает её эффективность: позволяет вычислить значение $p(d_0)$ существенно быстрее и для этого вычисления понадобится существенно меньше оперативной памяти вычислительной машины.

б). Приведите пример программы p на языке Рефал, формальный перевод которой на Haskell существенно повышает её эффективность.

3.1. Парные скобки как унарный конструктор построения дерева. *Унарный конструктор* парных круглых скобок (`data`$_0$) является средством структурирования данных. Удобно думать о действии этого конструктора, как о помещении его аргумента (внутренности скобок) внутрь матрёшки: мы можем переносить матрёшку из одного места в другое как единое целое – не заботясь о том, что спрятано внутри неё; как и содержимое матрёшки, внутренность скобки имеет конечную *глубину*.

Хотя формально в Рефале все функции, по определению, одного аргумента (*унарны*) – конечной последовательности, конструктор скобок позволяет имитировать произвольное число аргументов. Например, функцию `F` от двух аргументов можно синтаксически представить так

$$\texttt{<F (}x_1 \ldots x_n\texttt{) (}y_1 \ldots y_m\texttt{)>}$$

или так

$$\texttt{<F }x_1 \ldots x_n\ \texttt{(}y_1 \ldots y_m\texttt{)>}$$

Функция сложения двух натуральных чисел, представленных в унарной системе счисления, может быть определена так:

```
Summa {
 (e.n) (e.m) = e.n e.m;
}
```

Аналогичным образом можно имитировать отображение из D в $D \times D$:

```
F {
.............
... = (x1  ... xn) (y1  ... ym);
}
```

Как мы знаем, каждое Рефал-данное представляет лес корневых ориентированных деревьев.

Задача 2. Определить на Рефале функцию, которая по данному лесу:

а). строит последовательность всех листьев этого леса, не изменяя порядок их следования;

б). строит последовательность всех листьев, пропуская повторные вхождения равных листьев, но не изменяя порядок их следования.

Расширим алфавит Рефал-символов: каждое *слово*, написанное в латинице, будем считать символом, если это слово не заключено в одинарные кавычки. Например,

- `refal` – символ языка Рефал, то есть входит в О.Д.З. `s`-переменной;
- `'refal'` – последовательность из пяти символов.

Каждое слово, состоящее из букв Кириллицы и латиницы, *заключённое в двойные кавычки*, есть, по определению, Рефал-символ. Причём, `'b'` $\neq$ `b`, но `"b" = b`.

После нашего расширения алфавита Рефала, следующее стихотворение Валерия Брюсова будет обобщенным палиндромом (см. предыдущую лекцию №2).

```
"жестоко"  "раздумье"  '.'  "ночное"  "молчанье"
"качает"  "виденья"  "былого" ','
"мерцанье"  "встречает"  "улыбки"  "сурово" ','
                                    "страданье"
                    "глубоко"  '-'  "глубоко"  '!'
"страданье"  "сурово"  "улыбки"  "встречает"  '...'
"мерцанье"  "былого"  "виденья"  "качает"  '...'
"молчанье"  '.'  "ночное"  "раздумье"  "жестоко" '.'
```

Задача 3. Дано объектное Рефал-выражение d. Написать программу, заменяющую в d каждое слово `Haskell` на слово `Refal`.

Скелетом Рефал-выражения называется его скобочная структура. Например, скелет выражения `((refal) (()) ("выражение"))` есть `(() (()) ())`.

Задача 4. По данному объектному Рефал-выражению построить его скелет.

3.2. Абстрактная Рефал-машина. В *базисном* подмножестве Рефала программы представляют собой множество *определений-функций*. Каждое Рефал-определение есть непустая последовательность *предложений*, которые разделяются точкой с запятой. Каждое предложение имеет вид:

```
образец = правая_часть ;
```

Образец и *правую часть* определим индуктивно посредством формальной грамматики:

```
образец ::= терм-образец образец1 | empty
терм-образец ::= (образец) | СИМВОЛ | переменная
переменная ::= e.NAME | t.NAME | s.NAME
empty ::=

правая_часть ::= терм-правая_часть правая_часть1 | empty
```

```
терм-правая_часть ::= вызов_функции | (правая_часть) | СИМВОЛ
                    | переменная

вызов_функции ::= <NAME аргументы>
аргументы ::= правая_часть
```

Здесь *базисные понятия* мы написали прописными буквами. Существует дополнительное требование: множество переменных правой части предложения является подмножеством переменных образца этого предложения.

Все ранее рассмотренные нами программы принадлежат базисному Рефалу.

Рефал-машина реализует функцию семантики языка Рефал. Опишем *абстрактную машину базисного Рефала*. (Повсюду ниже, если не оговаривается иное, мы будем под словом Рефал иметь в виду базисный Рефал.)

Рабочая память Рефал-машины называется *полем зрения*.

Перед началом работы в поле зрения помещается вызов функции **Go**:

```
<Go >
```

Процесс работы Рефал-машины есть исполнение конечной последовательности её шагов.

Шагом Рефал-машины называется следующая последовательность действий:

1. Выбираем в поле зрения активный вызов функции `<F` $\mathtt{d}_0$`>` – алгоритм этого выбора мы дадим ниже (стартовый активный вызов есть `<Go >`); `i := 1`.
2. Пусть «образец$_i$ `=` правая_часть$_i$» есть i-тое предложение функции `F`, тогда решается уравнение «образец$_i$ `=` $\mathtt{d}_0$», если это уравнение несовместно, то переходим к решению уравнения «образец$_{(i+1)}$ `=` $\mathtt{d}_0$». Если же i-ое предложение последнее, то значение функции `F` на аргументе $\mathtt{d}_0$ не определено и Рефал-машина переходит в состояние *аварийной остановки*.
3. Пусть уравнение «образец$_i$ `=` $\mathtt{d}_0$» имеет решение. Если решение единственно, тогда переходим к пункту 4, иначе существует `e`-переменная, которая принимает более одного значения. Такие `e`-переменные называются *открытыми* в данном образце. Рассмотрим самую левую открытую переменную `e.y`; среди всех возможных значений `e.y` – конечных последовательностей $x_1^0 \ \ldots \ x_n^0$ выбираем последовательность с наименьшей длиной n и считаем только её значением переменной `e.y`, отбрасывая все решения уравнения «образец$_i$ `=` $\mathtt{d}_0$», которые не удовлетворяют данному условию. Таким образом, мы сузили множество решений M. Если в образце остались другие открытые переменные, то выбираем среди них самую левую и выбираем из множества её значений самое короткое и т. д., пока в образце не останется открытых переменных.
4. Имеем выбранное предложение «образец$_i$ `=` правая_часть$_i$» и выбранное нами его единственное решение. Делаем подстановку значений переменных в правую часть i-ого предложения и заменяем в поле зрения активный вызов `<F` $\mathtt{d}_0$`>` результатом этой подстановки.
5. Если в изменённом поле зрения нет вызовов функций, то машина останавливается, и поле зрения считается результатом её работы.

Заметим, что реальный (не абстрактный) алгоритм шага Рефал-машины может быть иным. Например, в Рефале-5 пункты 2 и 3 шага делаются одновременно и ищется сразу необходимое решение уравнения, а не всё множество решений.

Поле зрения, по определению, есть

`поле_зрения ::= терм-поле_зрения поле_зрения`$_1$ `| empty`
`терм-поле_зрения ::= <NAME поле_зрения> | (поле_зрения) | СИМВОЛ`

Активным вызовом функции в поле зрения называется вызов, правая закрывающая угловая скобка которого является самой левой из всех правых угловых скобок.

Задача 5. Дана последовательность, элементами которой являются скобки `'('`, `')'`, `'<'`, `'>'`. Написать программу, выдающую сообщение об ошибке, если существует скобочный уровень одного типа скобок, на котором нарушена правильность скобочной структуры любого типа скобок. Если же на вход программы подана правильная скобочная структура, то преобразовать каждую пару `'('` ... `')'` в структурные скобки `(` ... `)`, а каждую пару `'<'` ... `'>'` в `(Call` ... `)`. Например,

`'(<()>)'` $\Rightarrow$ `( (Call ()) ).`

3.3. Домашнее задание.

Задача 6. Скобочной унарной системой счисления назовём систему счисления, в которой натуральные числа представляются парными вложенными скобками. Например, `((()))` = 3. Определить на Рефале функции скобочной унарной арифметики: сложения, вычитания, умножения и деления с остатком двух натуральных чисел.

Задача 7. Дана правильная скобочная структура d, содержащая только скобки. Требуется написать программу, результатом которой является последовательность всех различных объектных Рефал-выражений, скелеты которых совпадают с d и которые содержат ровно один символ `b`. Например,

`(()) ()` $\Rightarrow$ `'[' b (()) () ']' '[' (b ()) () ']' '[' ((b)) () ']'`
`'[' (() b) () ']' '[' (()) b () ']' '[' (()) (b) ']'`
`'[' (()) () b ']'`

Задача 8. Определить, может ли быть получен палиндром из данного слова путём перестановки букв.

Задача 9. (*Примитивный корень последовательности*). На входе конечная последовательность объектных Рефал-термов. Найти такую последовательность минимальной длины, повторением которой несколько раз можно получить входную последовательность.

Задача 10. *Анаграмма* – литературный прием, состоящий в перестановке букв определенного слова (или словосочетания), что в результате дает другое слово или словосочетание. Преобразовать текст в анаграмму, где одинаковые символы стоят подряд (так шифровали открытия средневековые ученые).

§ 4. Арифметика (Лекция №4)

Целочисленная арифметика в Рефале является точной. Другими словами: ограничение на разрядность (длину) целых чисел отсутствует. Представление целых чисел зависит от конкретного диалекта Рефала. Мы рассмотрим, каким образом целые числа представлены в Рефале-5.

4.1. Представление целых чисел. Прежде всего, снова расширим алфавит Рефал-символов: натуральное число n принадлежит алфавиту тогда и только тогда, когда

$$0 \leqslant n < 2^{32}$$

Целые числа в Рефале представляются в системе счисления по основанию 2^{32}. Таким образом, по определению, целое число принадлежит О.Д.З. `s`-переменной тогда и только тогда, когда оно является цифрой в этой системе счисления. Чтобы избежать путаницы с десятичными цифрами, мы будем называть цифры в системе счисления по основанию 2^{32} макро-цифрами.

Задача 1. Используя школьные алгоритмы, определить на Рефале функции арифметики в двоичной системе счисления: сложения, вычитания, умножения и деления с остатком двух натуральных чисел.

Как и в десятичной системе счисления, произвольное натуральное число есть последовательность макро-цифр. Например,
$2^{32} = 1\ 0$
$2^{32} - 1 = 4294967295$
Заметим, что `'1'` $\neq 1$ и `'4294967295'` $\neq 4294967295$.
Отрицательные числа начинаются с Рефал-символа `'-'`. Например,
$-(2^{32}+1) =$ `'-' 1 1` .
Аналогично разрешается приписывать символ `'+'` перед неотрицательным числом.

4.2. Встроенные функции арифметики. Некоторые функции существуют в Рефал-машине изначально – они называются встроенными.

Все встроенные функции арифметики целых чисел (сложение – `Add` (или `+`), вычитание – `Sub` (или `-`), умножение – `Mul` (или `*`), деление – `Div` (или `/`), деление с остатком – `Divmod`, остаток от деления – `Mod`, сравнение – `Compare`) логически имеют два аргумента, первый из которых заключён в структурные скобки. Например,

```
<Add (1 0) 1> = 1 1
```

Результат деления с остатком есть пара:

```
<Divmod (e.n) e.m> = (e.q) e.r
```

где `e.n` = `e.q`*`e.m` + `e.r`, $0 \leqslant |$`e.r`$| < |$`e.m`$|$ и знак остатка `e.r` совпадает со знаком делимого `e.n`.

Задача 2. Дана последовательность цифр натурального числа n, данного в системе счисления по основанию 2^{32}. Построить последовательность цифр числа n в десятичной системе счисления.

Функция сравнения двух целых чисел `Compare` возвращает: `'+'`, если первое больше второго; `'-'`, если первое меньше второго, и `'0'`, если числа равны.

В том случае, когда не возникает неоднозначности, структурные скобки, заключающие первый аргумент встроенных функций арифметики, можно опускать.

Задача 3. Реализовать на Рефале алгоритм умножения столбиком двух натуральных чисел, данных в в системе счисления по основанию 2^{32}. Встроенные функции разрешается использовать только тогда, когда их аргументы являются макро-цифрами.

4.3. Два типа рекурсии. На примере функции $n!$ покажем два вида рекурсии.

Рекурсия первого вида соответствует умножению чисел от 1 до n: $n! = 1 * \ldots * n$.

```
Fact {
 1   =  1;
 s.n = <Mul (s.n) <Fact <Sub s.n 1>>>;
}
```

Задача 4. Объясните, почему аргументом функции `Fact` выбрана макроцифра, а не произвольное натуральное число. Как будет работать Рефал-машина при попытке вычислить вызов `<Fact 0>`?

Рекурсия второго вида соответствует умножению чисел в обратном порядке: $n! = n * \ldots * 1$.

```
Fact1 {
 s.n = <Fact2 s.n 1>;
}

Fact2 {
 s.n s.n =  s.n;
 s.n s.m = <Mul (s.m) <Fact2 s.n <Add s.m 1>>>;
}
```

По способу завершения рекурсии, рекурсию первого вида назовём рекурсией с выходом по исчерпанию; рекурсию второго вида – рекурсией с выходом по наполнению.

Задача 5. Дано натуральное число n. Вычислить на Рефале n-ое число Каталана c_n (см. лекцию 1, задачу 7).

4.4. Фибоначчиева система счисления. Числа Фибоначчи определяются индуктивно: $f_1 = f_2 = 1, f_{n+2} = f_n + f_{n+1}$.

Задача 6. Дано натуральное число n. Вычислить на Рефале n-ое число Фибоначчи f_n.

Докажем, что любое натуральное число $m > 0$ можно однозначным образом представить в виде: $m = a_n * f_n + \ldots + a_2 * f_2$, где для всех $i < n$ верно, что a_i равно нулю или единице и $a_n = 1$, для всех i верно $a_i * a_{i+1} = 0$. Таким

образом, получаем фибоначчиеву систему счисления: a_i – цифры числа m в этой системе счисления.

Доказательство будет конструктивно и индуктивно.

Вычтем из $m_0 = m$ наибольшее из не превосходящих его чисел Фиббоначи f_n (с наибольшим номером) и рассмотрим $m_1 = m_0 - f_n$, $a_n = 1$.

Предположим, что k шагов уже выполнены, и мы имеем некоторое число m_k и последовательность фибоначчиевых цифр $a_n, \quad \ldots, a_{n-k}$, состоящую из нулей и единиц. Тогда на $k+1$ шаге построим фибоначчиеву цифру $a_{n-(k+1)}$ и число m_{k+1}.

Если $m_k < f_{n-k}$, то $a_{n-(k+1)} := 0$ и $m_{k+1} := m_k$,
иначе $a_{k+1} := 1$ и $m_{k+1} := m_k - f_{n-k}$.

Покажем корректность приведённого алгоритма.

На каждом шаге k уменьшается номер $n-k$ числа Фибоначчи, сравниваемого с $m_k \geqslant 0$ и, следовательно, уменьшается f_{n-k}.

Индукцией по k покажем, что $\forall k.(m_k \geqslant f_{n-k}) \Rightarrow (m_{k+1} < f_{n-(k+1)})$.

1. Базис индукции: $f_{n+1} > m_0 \geqslant f_n \Rightarrow m_1 = (m_0 - f_n) < (f_{n+1} - f_n) = f_{n-1}$. Базис индукции доказан.
2. Шаг индукции: пусть утверждение верно $\forall i \leqslant k$, тогда либо $m_{k+1} < f_{n-(k+1)}$ и посылка нашего утверждения для $k+1$ ложна; либо $m_{k+1} \geqslant f_{n-(k+1)}$ и тогда, по определению алгоритма, $m_{k+2} = m_{k+1} - f_{n-(k+1)}$.
   Кроме того, по предположению индукции, неравенство $m_{k+1} \geqslant f_{n-(k+1)}$ влечёт $m_k < f_{n-k}$ и, следовательно, $m_{k+1} = m_k$. Таким образом, $m_{k+2} = m_{k+1} - f_{n-(k+1)} = (m_k - f_{n-(k+1)}) < (f_{n-k} - f_{n-(k+1)}) = f_{n-(k+2)}$.
   Шаг индукции доказан.

Следствие 1. Алгоритм разложения натурального числа к виду $a_n * f_n + \ldots + a_2 * f_2$, где $\forall i < n.\ a_i$ равно нулю или единице и $a_n = 1$, заканчивает свою работу за конечное число шагов.

Действительно, так как $f_{n-(k+1)} \leqslant f_{n-k}$ и $\min(m_k, m_{k+1}) = m_{k+1}$, то доказанное утверждение даёт $0 \leqslant m_{k+1} < f_{n-k}$, а последовательность f_{n-k} при $k < n-1$ строго убывает и $f_{n-(n-2)} = f_2 = 1$.

Пример:

$$m = m_0 = 19, m_1 = 19 - 13 = 19 - f_7 = 6; a_7 = 1;$$
$$m_1 < f_6 = 8 \Rightarrow a_6 = 0; m_2 = m_1;$$
$$m_2 \geqslant f_5 = 5 \Rightarrow a_5 = 1; m_3 = m_2 - f_5 = 6 - 5 = 1;$$
$$m_3 < f_4 = 3 \Rightarrow a_4 = 0; m_4 = m_3;$$
$$m_4 < f_3 = 2 \Rightarrow a_3 = 0; m_5 = m_4;$$
$$m_5 \geqslant f_2 = 1 \Rightarrow a_2 = 1; m_6 = m_5 - f_2 = 1 - 1 = 0;$$

и имеем $19_{10} = 101001_f = f_7 + f_5 + f_2 = 13_{10} + 5_{10} + 1_{10}$.

Задача 7. Напишите программу:

а). переводящую числа из фибоначчиевой системы счисления в десятичную;
б). переводящую числа из десятичной системы счисления в фибоначчиеву;
в). сложения двух натуральных чисел в фибоначчиевой системе счисления;
г). умножения двух натуральных чисел в фибоначчиевой системе счисления.

4.5. Домашнее задание.

Задача 8. В шестнадцатеричной системе счисления цифры представлены символами: `'0'`, `'1'`, `'2'`, `'3'`, `'4'`, `'5'`, `'6'`, `'7'`, `'8'`, `'9'`, `'A'`, `'B'`, `'C'`, `'D'`, `'E'`, `'F'`. Используя школьные алгоритмы, определить на Рефале функции арифметики в шестнадцатеричной системе счисления: сложения, вычитания, умножения и деления с остатком двух натуральных чисел.

Задача 9. Написать программу, вычисляющую наибольший общий делитель двух натуральных чисел методом Евклида.

Задача 10. Написать программу, вычисляющую наименьшее общее кратное двух натуральных чисел.

Задача 11. Придумать представление рациональных чисел посредством данных Рефала. Реализовать рациональную арифметику (функции сложения, вычитания, умножения, деления двух рациональных чисел).

Задача 12. Дано натуральное число n. Вычислить на Рефале n-ое число Фибоначчи f_n, если требуется, чтобы число арифметических операций при этом вычислении было пропорционально $\log n$.

§ 5. Функциональность. Ввод-вывод. (Лекция №5)

Второе определяющее слово реФала есть *функциональный*. Язык программирования $\{D, P, S\}$ называется функциональным, если для всех $p_0 \in P$ $S(p_0, d)$ есть *функция* из D в $D_\perp$ (см. лекцию №2). Другими словами, программа p_0 представляет собой определение некоторого *отображения* из D в $D_\perp$.

Функциональным языком, в строгом смысле приведённого выше определения, может быть только чисто теоретический язык программирования.

Задача 1. Какие из рассмотренных в предыдущих семинарах программы не являются определениями функций из D в $D_\perp$, где D – множество данных Рефала?

Причина состоит в том, что в *реальных* языках программирования всегда существуют программы с *побочными действиями*. И без таких программ мы не сможем узнать, что же вычислил компьютер, и вычислил ли вообще что-нибудь. Мы рассмотрим некоторые из подобных программ ниже.

Среди *реальных* (не чисто теоретических) языков программирования функциональными принято называть те языки, в которых число выразительных синтаксических средств, позволяющих конструировать программы с побочными действиями, сведено до минимума. *Типичным свойством* функционального языка программирования является отсутствие в нём глобальных переменных.

Задача 2. Приведите пример языка программирования, не являющегося функциональным.

5.1. Встроенный ввод-вывод. Мы уже использовали встроенную «функцию» печати `Prout`: её значением всегда является пустое выражение; а *побочным действием* – вывод на терминал своего аргумента.

Часто бывает необходимо вывести результат вычислений Рефал-машины в файл: с целью длительного хранения этого результата, другой причиной может быть большой объём выводимых данных.

Организация работы Рефал-машины с файлами зависит от конкретного диалекта Рефала. Мы будем интересоваться Рефалом-5.

File, в переводе с английского языка, – папка бумаг; она может быть пустая или содержать некоторую текстовую информацию. На обложке папки написано её имя.

Чтобы работать с содержимым файла, его, как и папку, нужно сначала *открыть*. При завершении работы с содержимым файла, файл необходимо *закрыть*. Цель работы может быть разная: чтение содержимого файла; добавление текста; уничтожение всей информации как устаревшей и запись в файл новой информации. У папок, взятых с разных книжных полок, могут быть совпадающие имена; чтобы не возникала путаница при работе с ними, принято прикреплять к ним (в момент открытия) *уникальные этикетки*, которые являются макро-цифрами – описателями (дескрипторами – descriptors). Посредством этих описателей и происходит реальная работа с файлами.

Файл с именем `'name'` открывает вызов

```
<Open s.Mode s.D 'name'>
```

где `s.D` – дескриптор файла, который задаёт программист, а `s.Mode` указывает цель работы с содержимым файла: `'r'` – открыть для чтения, `'a'` – открыть для дополнения, `'w'` – открыть для записи, предварительно *уничтожив его содержимое*. Если происходит попытка открыть для записи несуществующий файл, то он автоматически создаётся.

Закрываем файл посредством вызова

```
<Close s.D>
```

Значения «функций» `Open` и `Close` суть пустые выражения.

«Функция» `Putout` записывает (в виде одной строки) значение `e.Expr` в файл, связанный с дескриптором `s.D`:

```
<Putout s.D e.Expr>
```

Следующий вызов этой «функции» будет производить запись с начала следующей строки.

Задача 3. (*Соревнование*) Написать как можно более короткую программу, записывающую в файл `turnip.txt` русскую народную сказку «Репка».

Аналогично выводу, чтение (ввод) в Рефале-5 также построчный: n-ый вызов

```
<Get s.D>
```

выдаёт n-ую строку файла, связанного с `s.D`.

Чтение с клавиатуры производится вызовом

```
<Card >
```

который завершает чтение по нажатию клавиши `Enter`.

Задача 4. Написать программу, переставляющую строки в данном файле в обратном порядке.

Значение вызова `<Arg s.n>` есть `s.n`-ый аргумент командной строки операционной системы при запуске Рефал-машины.

Задача 5. Написать программу, значением которой является последовательность всех аргументов Рефал-машины, передаваемых командной строкой операционной системы.

5.2. Пример А.В. Корлюкова. Рассмотрим пример Александра Корлюкова, показывающий выразительные возможности ассоциативной конкатенации.

Пусть $B = \{\texttt{'0'}, \texttt{'1'}\}$, $M \supset B^3$. Требуется определить на Рефале функцию $F : M \to B^2$ такую, что $\forall x \in B^3$. $F(x)$ есть число n единиц в x, где n представлено в двоичной системе счисления и, возможно, содержит в своём представлении лидирующие нули.

Задачу начнём решать с полного перебора:

```
F {
 '111'  =  '11';
 '110'  =  '10';
 '101'  =  '10';
 '100'  =  '01';
 '011'  =  '10';
 '010'  =  '01';
 '001'  =  '01';
 '000'  =  '00';
}
```

В этом определении предложения перестановочны. Объединяя первое и последнее предложение, получаем

```
s.d s.d s.d = s.d s.d;
```

Аналогично объединяем все предложения с совпадающими правыми частями. Наше определение теперь выглядит нижеследующим образом:

```
F {
  s.d s.d  s.d  =  s.d s.d;
  e.x  '1' e.y  '1' e.z  = '10';
  e.x  '0' e.y  '0' e.z  = '01';
}
```

Заметим, что во втором и в третьем предложении значения двух переменных равны пустому выражению; значение третьей переменной во втором предложении равно `'0'`, а в третьем – `'1'`. Таким образом, второе и третье предложение также можно объединить:

```
F {
  s.d s.d s.d  =  s.d s.d;
  e.x s.d e.y s.d e.z  =  s.d e.x e.y e.z;
}
```

Теперь нетрудно видеть, что первое предложение лишнее. Действительно, в результате отождествления трёх равных цифр с образцом `e.x s.d e.y s.d e.z`:
`e.x s.d e.y s.d e.z = d0 d0 d0`
имеем
`s.d = d0`
`e.x =`
`e.y =`
`e.z = d0`
и правая часть второго предложения становится равной `d0 d0`, то есть совпадает с правой частью первого предложения.

Окончательно имеем определение:

```
F {
  e.x s.d e.y s.d e.z = s.d e.x e.y e.z;
}
```

В результате наших преобразований область определения функции, данной стартовым определением, была расширена.

Задача 6. Написать программу, стирающую двойные (лишние) пробелы в данном тексте, то есть заменяющую каждую группу стоящих подряд пробелов единственным пробелом.

5.3. Домашнее задание.

Задача 7. Придумать представление понятия множества посредством данных Рефала. Определить функции объединения, пересечения и разности двух множеств.

Задача 8. (*Соревнование*) Написать как можно более короткую программу, записывающую в файл `Jack.txt` нижеследующее стихотворение Джонатана Свифта:

`The House That Jack Built`
`Jonathan Swift`

This is the house that Jack built.

This is the malt
That lay in the house that Jack built.

This is the rat,
That ate the malt
That lay in the house that Jack built.

This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the dog,
That worried the cat,
That killed the rat,
That ate the malt

That lay in the house that Jack built.

This is the cow with the crumpled horn,
That tossed the dog,
That worried the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the maiden all forlorn,
That milked the cow with the crumpled horn,
That tossed the dog,
That worried the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the man all tattered and torn,
That kissed the maiden all forlorn,
That milked the cow with the crumpled horn,
That tossed the dog,
That worried the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the priest all shaven and shorn,
That married the man all tattered and torn,
That kissed the maiden all forlorn,
That milked the cow with the crumpled horn,
That tossed the dog,
That worried the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the cock that crowed in the morn,
That waked the priest all shaven and shorn,
That married the man all tattered and torn,
That kissed the maiden all forlorn,
That milked the cow with the crumpled horn,
That tossed the dog,
That worried the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the farmer sowing the corn,
That kept the cock that crowed in the morn,
That waked the priest all shaven and shorn,
That married the man all tattered and torn,
That kissed the maiden all forlorn,
That milked the cow with the crumpled horn,

That tossed the dog,
That worried the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

Задача 9. Сонет -– стихотворение из 14 строк, которые образуют особым образом построенную строфу. Венок сонетов состоит из 15 стихотворений, где последняя строка каждого из 14 сонетов является началом следующего. Заключительный сонет воспроизводит первые строки всех 14 предыдущих сонетов (см. [9]).

Написать программу, проверяющую, является ли данное в файле стихотворение венком сонетов (см. определение венка сонетов на странице [9]). Программу протестировать на следующем стихотворении Владимира Солоухина (см. также файл [10]).

1

Венок сонетов – давняя мечта.
Изведать власть железного канона!
Теряя форму, гибнет красота,
А форма чётко требует закона.

Невыносима больше маета
Аморфности, неряшливости тона,
До скрежета зубовного, до стона,
Уж если так, пусть лучше немота.

Прошли, прошли Петрарки времена.
Но в прежнем ритме синяя волна
Бежит к земле из дали ураганной.

И если ты всё – мастер и поэт,
К тебе придёт классический сонет –
Вершина формы строгой и чеканной.

2

Вершина формы строгой и чеканной –
Земной цветок: жасмин, тюльпан, горлец,
Кипрей и клевер, лилии и канны,
Сирень и роза, ландыш, наконец.

Любой цветок сорви среди поляны –
Тончайшего искусства образец,
Не допустил Ваятеля резец
Ни одного малейшего изъяна.

Как скудно мы общаемся с цветами.
Меж красотой и суетными нами
Лежит тупая жирная черта.

Но не считай цветенье их напрасным,
Мы к ним идём, пречистым и прекрасным,
Когда невыносима суета.

3

Когда невыносима суета,
И возникает боль в душе глубоко,
И складка горькая ложится возле рта,
Я открываю том заветный Блока.

Звенит строка, из бронзы отлита,
Печального и гордого пророка.
Душа вольна, как дальняя дорога,
И до звезды бездонна высота.

О Блок! О Блок! Мертвею, воскреси!
Кидай на землю, мучай, возноси
Скрипичной болью, музыкой органной!

Чисты твоей поэзии ключи.
Кричать могу, молчанью научи,
К тебе я возвращаюсь в день туманный.

4

К тебе я возвращаюсь в день туманный,
О Родина, ужели это сны?
Кладу букет цветов благоуханный
На холмик глины около сосны.

И около берёзы. И в Тарханах.
И у церковной каменной стены.
Поэты спят; те стойкой ресторанной,
Те пошлостью, те пулей сражены.

А нас толпа. Мы мечемся. Мы живы.
Слова у нас то искренни, то лживы.
И без звезды живём и без креста.

Но есть дела. Они первостепенны.
Да ты ещё маячишь неизменно,
О белизна бумажного листа!

5

О белизна бумажного листа!
Ни завитка, ни чёрточки, ни знака.
Ни мысли и ни кляксы. Немота.
И слепота. Нейтральная бумага.

Пока она безбрежна и чиста,
Нужны или наивность, иль отвага
Для первого пятнающего шага –
Оставишь след и не сотрёшь следа.

Поддавшись страшной власти новизны,
Не оскверняй великой белизны
Поспешным жестом, пошлостью пространной.

Та белизна – дорога и судьба,

Та белизна – царица и раба,
Она источник жажды окаянной.

6

Она источник жажды окаянной.
Вся жизнь, что нам назначено прожить.
И соль и мёд, и горечь браги пьяной
Чем больше пьёшь, тем больше хочешь пить.

Сладкó вино за стенкою стаканной,
Мы пьём и льём, беспечна наша прыть,
До той поры, когда уж нечем крыть
И жалок мусор мелочи карманной.

За ледоход! За дождь! За листопад!
За синий свод – награду из наград,
За жаворонка в полдень осиянный,

За все цветы, за все шипы земли,
За постоянно брезжущий вдали
Манящий образ женщины желанной!

7

Манящий образ женщины желанной ...
Да – помыслы, да – книги, да – борьба.
Но всё равно одной улыбкой странной
Она творит героя и раба.

Ты важный, нужный, яркий, многогранный,
Поэт, главарь, – завидная судьба!
Уйдёт с другим, и ты сойдёшь с ума,
И будешь бредить пулею наганной.

Немного надо – встретиться любя.
Но если нет, то всюду ждут тебя
В пустых ночах пустые города,

Да всё-таки – надежды слабый луч,
Да всё-таки – сверкнувшая из туч
В ночи осенней яркая звезда.

8

В ночи осенней яркая звезда,
Перед тобой стою среди дороги.
О чём горишь, зовёшь меня куда,
Какие ждут невзгоды и тревоги?

Проходит лет, событий череда,
То свет в окне, то слёзы на пороге,
Глаза людей то ласковы, то строги,
Всё копится для страшного суда.

Для каждого наступит судный день:
Кем был, кем стал, где умысел, где лень?

Ты сам себе и жертва и палач.

Ну что ж, ложись на плаху головою,
Но оставайся всё-таки собою,
Себя другим в угоду не иначь.

9

Себя другим в угоду не иначь.
Они умней тебя и совершенней,
Но для твоих вопросов и задач
Им не найти ответов и решений.

Ты никуда не денешься, хоть плачь,
От прямиков, окольностей, кружений,
От дерзновенных взлётов и крушений,
От всех своих побед и неудач.

Привалов нет, каникул не бывает.
В пути не каждый сразу понимает,
Что жизнь не тульский пряник, не калач.

Рюкзак годов всё крепче режет плечи,
Но если вышел времени навстречу,
Души от ветра времени не прячь!

10

Души от ветра времени не прячь ...
Стоять среди железного мороза
Умеет наша светлая берёза,
В огне пустынь не гибнет карагач.

Но точит волю вечная угроза.
Но подлецом не должен быть скрипач.
Но губят песню сытость, ложь и проза,
Спасти её – задача из задач.

Берёшь, глядишь: такие же слова,
Похожа на живую, а мертва.
Но если в ней сознанье угадало

Хоть уголёк горячий и живой,
Ты подними её над головой,
Чтобы её как факел раздувало.

11

Чтобы её как факел раздувало,
Ту истину, которая в тебе,
Не опускай тяжёлого забрала,
Летя навстречу буре и борьбе.

Тлен не растлил, и сила не сломала.
И медлит та, с косою на горбе.
Хвала, осанна, ода, гимн судьбе –
Ты жив и зряч, не много и не мало!

С тобой деревья, небо над тобой.
Когда же сердце переполнит боль,
Оно взорвётся ярко, как фугас.

Возможность эту помни и держи,
Для этого от сытости и лжи
Хранится в сердце мужества запас.

12

Хранится в сердце мужества запас,
Как раньше порох в крепости хранили,
Как провиант от сырости и гнили,
Как на морском судёнышке компас.

Пускай в деревьях соки отбродили,
Пусть летний полдень засуху припас,
Пусть осень дышит холодом на нас
И журавли над нами оттрубили,

Пусть на дворе по-зимнему темно,
Согреет кровь старинное вино,
Уздечкой звякнет старенький пегас.

Придут друзья – обрадуемся встрече,
На стол поставим пушкинские свечи,
Чтоб свет во тьме, как прежде, не погас!

13

И свет во тьме, как прежде, не погас.
Да разве свет когда-нибудь погаснет?!
Костром горит, окном манит в ненастье,
В словах сквозит и светится из глаз.

Пустые толки; домыслы и басни,
Что можно, глыбой мрака навалясь,
Идущий день отсрочить хоть на час,
Нет ничего смешнее и напрасней!

И мрак ползёт. То атомный распад.
То душ распад. То свист, а то поп-арт.
Приоритет не духа, а металла.

Но под пустой и жалкой суетой
Он жив, огонь поэзии святой,
И тьма его, как прежде, не объяла.

14

И тьма его, как прежде, не объяла,
Мой незаметный, робкий огонёк.
Несу его то бодро, то устало,
То обогрет людьми, то одинок.

Уже не мало сердце отстучало,
Исписан и исчеркан весь листок,

Ошибок – воз, но этот путь жесток,
И ничего нельзя начать сначала.

Не изорвать в сердцах черновика,
Не исправима каждая строка,
Не истребима каждая черта.

С рассветом в путь, в привычную дорогу.
Ну, а пока дописан, слава Богу,
Венок сонетов – давняя мечта.

15

Венок сонетов – давняя мечта,
Вершина формы строгой и чеканной,
Когда невыносима суета,
К тебе я обращаюсь в день туманный.

О белизна бумажного листа!
Она источник жажды окаянной,
Манящий образ женщины желанной,
В ночи осенней яркая звезда!

Себя другим в угоду не иначь.
Души от ветра времени не прячь,
Чтобы её, как факел, раздувало.

Хранится в сердце мужества запас.
И свет во тьме, как прежде, не погас,
И тьма его, как прежде, не объяла!

§ 6. Алгоритмическая полнота. Алгорифмы Маркова. (Лекция №6)

Язык рефАЛ *алгоритмический*. Естественно встаёт вопрос об *алгоритмической полноте* Рефала. Оказывается, даже базисное подмножество Рефала, которое мы до сих пор только и рассматривали, является *алгоритмически полным языком*. То есть любой алгоритм может быть запрограммирован на базисном Рефале.

6.1. Алгоритмическая полнота.

Задача 1. Воспользовавшись тезисом Чёрча, докажите что базисный Рефал является алгоритмически полным языком.

Бóльшая часть языков программирования обладает свойством алгоритмической полноты. Следовательно, любую программу, написанную на одном из таких языков программирования, можно перевести на другой такой язык.

Разнообразие языков программирования объясняется их ориентацией на конкретные классы задач. Вопрос не в том можно или нельзя решить какую-то задачу на данном языке программировании L, а в том *удобно или нет* решать эту задачу на языке L. Язык есть инструмент для решения задачи: хотя и

можно гвозди забивать сковородкой (и даже лбом), но очень неудобно, – предпочитают использовать молоток.

Рефал ориентирован на *преобразование текстов* (символьной информации).

6.2. Алгорифмы Маркова. *Операционная семантика* языка Рефал основана на теоретической модели вычислений, называемой нормальными алгорифмами А. А. Маркова.

Пусть дан некоторый алфавит $\mathcal{A}$. Рассмотрим свободную алфавитную полугруппу G над $\mathcal{A}$. В качестве элементарной операции, на базе которой строятся алгорифмы Маркова, используется подстановка одного элемента G вместо другого. Если $x, y \in G$, то выражения $x \ \to \ y$ и $x \ \to \ \bullet \ y$ будем называть *формулами подстановки*. При этом предполагается, что стрелка $\to$ и точка $\bullet$ не являются буквами алфавита $\mathcal{A}$.

Формула подстановки $x \ \to \ y$ называется *простой*. Формула подстановки $x \ \to \ \bullet \ y$ называется *заключительной*. Пусть $P \ \to \ Q$ обозначает любую из формул подстановки (простую или заключительную). Конечная последовательность формул подстановки

$$\begin{cases} P_1 \ \to \ Q_1 \\ P_2 \ \to \ Q_2 \\ \ldots\ldots\ldots \\ P_r \ \to \ Q_r \end{cases}$$

называется схемой алгорифма.

Скажем, что последовательность (как элемент) $x \in G$ входит в последовательность $y \in G$, если существуют такие (возможно, пустые) $u, v \in G$, что $y = u \ x \ v$. Длину последовательности $x \in G$ обозначим $|x|$.

Задача 2. Пусть дано $y \in G$. Скажем, что существует два перекрывающихся вхождения $x \in G$ в y, если $\exists v \in G$ такое, что v входит в y, $|x| < |v| < 2|x|$ и $\exists p \in G \ \exists q \in G. \ \ v = x \ p = q \ x$. Написать на Рефале программу, проверяющую, что для данной последовательности y не существует последовательности x, которая имеет перекрывающееся вхождение в y.

Работа алгоритма, порождённого схемой алгорифма, может быть описана следующим образом. Пусть дано $p \in G$.

- Находим первую в схеме алгорифма формулу подстановки $P_m \ \to \ Q_m$ такую, что P_m входит в p.
- Подставляем Q_m вместо самого левого вхождения P_m в p. Пусть r_1 – результат такой подстановки.
- Если $P_m \ \to \ Q_m$ – заключительная формула подстановки, то работа алгоритма заканчивается и его значением на p является r_1.
- Если $P_m \ \to \ Q_m$ – простая, то применим к r_1 тот же поиск, который только что применяли к p, и так далее.
- Если мы на i-ом шаге получим такое r_i, что ни одна из $P_1, \ \ldots, P_r$ не входит в r_i, то работа алгоритма заканчивается и r_i будет его значением на p.

- При этом возможно, что описанный процесс никогда не закончится. В таком случае мы скажем, что алгоритм не применим к p.

Алгоритм, определённый таким образом, называется *нормальным алгорифмом Маркова в алфавите* $\mathcal{A}$.

Задача 3. Объясните в чём сходство и различие описанной выше машины Маркова и Рефал-машины. Что в машине Маркова является аналогом:

а). начальных данных?
б). вызова функции?
в). поля зрения?
г). активного вызова функции?
д). аварийной остановки «отождествление невозможно»?

Что в Рефал-машине является аналогом заключительной формулы подстановки?

Пример: Пусть $\mathcal{A} = \{b, c\}$. Пусть G – свободная алфавитная полугруппа над $\mathcal{A}$. Рассмотрим схему:

$$\begin{cases} b \to \bullet \\ c \to c \end{cases}$$

Определяемый этой схемой нормальный алгорифм перерабатывает всякое $p_0 \in G$, содержащее хотя бы одно вхождение буквы b, в элемент G, который получается вычёркиванием в p_0 самого левого вхождения буквы b. Пустая последовательность перерабатывается в саму себя. Алгорифм неприменим к непустым последовательностям, не содержащим буквы b.

Задача 4. Приведите конструктивное доказательство следующего утверждения. Если некоторый алгоритм A может быть описан схемой Маркова, то существует программа U, написанная на базисном Рефале такая, что для всех x из множества определения функции F, соответствующей алгоритму A, значение $F(x)$ может быть вычислено посредством программы U, то есть $F(x) = U(x)$. Под конструктивностью мы здесь понимаем явное предъявление программы U.

Задача 5. Реализуйте машину Маркова посредством Рефал-машины. Как связана данная задача с предыдущей? Проверьте работу построенной вами программы на всех алгорифмах Маркова, схемы которых рассматриваются в задачах или схемы которых требуется построить в задачах (данной лекции и домашнего задания).

6.3. Домашнее задание.

Задача 6. Пусть $\mathcal{A}$ – произвольный алфавит. Пусть G – свободная алфавитная полугруппа над $\mathcal{A}$. Рассмотрим (сокращённо записанную) схему алгорифма в алфавите $\mathcal{B} = \mathcal{A} \cup \{\alpha, \beta\}$, где $\alpha, \beta \notin \mathcal{A}$.

$$\begin{cases} \alpha\,\alpha \;\to\; \beta \\ \beta\,\xi \;\to\; \xi\,\beta & (\xi \in A) \\ \beta\,\alpha \;\to\; \beta \\ \beta \;\to\; \bullet \\ \alpha\,\eta\,\xi \;\to\; \xi\,\alpha\,\eta & (\eta, \xi \in A) \\ \quad \to\; \alpha \end{cases}$$

Эта схема определяет некоторый нормальный алгорифм Rev в алфавите $\mathcal{B}$. Докажите, что $\forall p \in G.\ Rev(p) = \overline{p}$. Где последовательность $\overline{p}$ получена из p обращением: если $p = a_1\ \ldots\ a_n$, то $\overline{p} = a_n\ \ldots\ a_1$.

Задача 7. Пусть алфавит $\mathcal{A}$ не содержит букву α, и пусть $\mathcal{B} = \mathcal{A} \cup \{\alpha\}$. Пусть G – свободная алфавитная полугруппа над $\mathcal{A}$. Построить нормальный алгорифм Маркова в $\mathcal{B}$, стирающий первую букву во всякой непустой последовательности $p \in G$.

Задача 8. Пусть алфавит $\mathcal{A}$ не содержит букв α, β, γ, и пусть $\mathcal{C} = \mathcal{A} \cup \{\alpha, \beta, \gamma\}$. Пусть G – свободная алфавитная полугруппу над $\mathcal{A}$. Построить нормальный алгорифм Маркова F в $\mathcal{C}$ такой, что $\forall p \in G$ было бы выполнено равенство $F(p) = p\,p$.

Задача 9. Даны две последовательности (два данных Рефала) $x_1, \ldots, x_n$ и $y_1, \ldots, y_k$. Требуется построить последовательность термов $z_1, \ldots, z_m$, состоящую из совпадающих и имеющих равные порядковые номера термов двух исходных последовательностей. Порядок термов z_i друг относительно друга не изменять.

Задача 10. Длина $Ln(\texttt{d})$ объектного Рефал-выражения `d` определяется индуктивно:

а). длина пустого выражения равна нулю;
б). $Ln(\texttt{t.x e.d}) = 1 + Ln(\texttt{e.d})$.

Определить на Рефале функцию, значение которой есть длина данного объектного выражения.

Задача 11. Размер $Size(\texttt{d})$ объектного Рефал-выражения `d` определяется индуктивно:

а). размер пустого выражения равен нулю;
б). $Size(\texttt{s.x e.d}) = 1 + Size(\texttt{e.d})$;
в). $Size(\texttt{(e.x) e.d}) = 2 + Size(\texttt{e.x}) + Size(\texttt{e.d})$.

Определить на Рефале функцию, значение которой есть размер данного объектного выражения.

§ 7. Вызов функции по определению (Лекция №7)

Кроме вызова функции по имени `<F e.args>`, то есть через указание имени данного определения функции, а не самого определения, синтаксис Рефала позволяет определить функцию непосредственно в точке её вызова, если обращение к такой функции в программе происходит лишь один раз. Начиная с этого места нашего изложения, мы выходим за рамки базисного Рефала.

7.1. Блоки. *Вызов функции по определению* иногда бывает удобен с точки зрения прозрачности исходного текста программы. Напомним, что в *базисном* Рефале определение функции имеет вид последовательности предложений, заключённой в фигурные скобки; а имя функции ставится перед её определением:

```
FunctionName {
   sentence_1;
   .........
   sentence_n;
}
```

где левая и правая части предложений разделены знаком равенства. Таким образом, определение само по себе (без имени) есть:

```
{
   sentence_1;
   .........
   sentence_n;
}
```

Если мы хотим определить функцию *непосредственно в точке её вызова*, то вместо имени функции (его у нас теперь нет) мы пишем сам текст определения, а аргументы этого вызова пишем перед фигурной скобкой, открывающей данное определение, отделив их от скобки знаком двоеточия «`:`». Непосредственно за *вызовом функции по определению* всегда ставится точка с запятой «`;`». Например, вызов

```
'abcd' (e.x) : {
                e.y (e.y) = True;
                e.y (e.z) = False;
               };
```

определяет, совпадает ли значение переменной `e.x` со строкой `'abcd'`, а вызов

```
<F e.w> (e.x) : {
                 e.y (e.y) = True;
                 e.y (e.z) = False;
                };
```

определяет, совпадает ли значение результата вызова (по имени) функции `F` со значением переменной `e.x`.

Теперь мы расширим понятие Рефал-предложения: кроме предложений, допускаемых базисным Рефалом, введём предложения, в которых левая часть отделяется от правой части запятой «,» (а не знаком равенства, как в базисном Рефале). Мы потребуем, чтобы в этом случае в правой части предложения (после запятой) обязательно стоял один вызов функции по определению, и только он. Естественно, мы допускаем вызовы по определению внутри любого определения – поименованного или непоименованного.

Пример №1: Следующая функция `S-Type` определяет, какого типа символ подан ей в качестве аргумента, и сообщает об ошибке, если аргумент не является Рефал-символом. Мы предварительно, используя встроенную функцию `Lower`, преобразуем все прописные буквы имени данного символа в строчные.

```
S-Type {
 s.x, <BelongsTo <Lower s.x> (<Letters>)>: {
                                            True   = Letter;
                                            False  = Other;
                                           };
 e.y = "Wrong argument: " e.y;
}

BelongsTo {
 s.x (e.y s.x e.z) = True;
 s.x (e.y) = False;
}

Letters {
 = 'abcdefghijklmnopqrstuvwxyz' 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя';
}
```

Заметим, что `<S-Type 'f'>` $\neq$ `<S-Type f>`. Причина сего в том, что `'f'` есть символ-буква, а `f` есть символ-слово, имя которого состоит из одной буквы.

Рассмотрим общий вид Рефал-предложения с вызовом функции по определению:

```
pattern, argument :  {
                        sentence1;
                        .........
                        sentencen;
                       };
```

Множество переменных, входящих в образец `pattern`, может пересекаться с множеством переменных входящих в некоторое предложение $\texttt{sentence}_i$. Например,

```
t.x e.y, <F e.y>: {
                   t.x e.z = e.z;
                   e.y e.y e.z = e.z e.z;
                   e.z = e.y t.x;
                  };
```

Здесь первые два предложения (функции, вызванной по определению) содержат в левых частях переменные из образца `t.x e.y` основного предложения (вызывающего функцию по определению), а третье предложение содержит в правой части переменные из того же образца. Как понимать такую ситуацию? Какой смысл приписать подобному синтаксису? Семантика подобного синтаксиса в Рефале следующая: значения всех переменных, которые определились до запятой «`,`», Рефал-машина подставляет и в аргумент, и в само определение вызова функции по определению; и только после этого вычисляет указанный вызов (естественно, предварительно вычислив его аргументы). Следовательно, в третьем предложении

`e.z = e.y t.x;`

нет синтаксической ошибки, ибо в момент вызова оно превратится в

$$\texttt{e.z = e.y}_0\ \texttt{t.x}_0\texttt{;}$$

где $\texttt{e.y}_0$ и $\texttt{t.x}_0$ конкретные данные (константы) – значения переменных `e.y` и `t.x`.

Другими словами можно сказать, что определение функции, вызываемой по определению, может быть параметризовано. В рассмотренном выше примере определение функции зависит от параметров (неизвестных данных) `e.y`, `t.x`. Сущности `e.y`, `t.x` являются синтаксическими понятиями – переменными с точки зрения вызывающей функции, но семантическими понятиями – параметрами, с точки зрения вызываемой функции.

Пример №2: Следующая функция

```
a-z {
 e.1 'a' e.2, e.2: {
                    e.3 'z' e.4 = (e.1) 'a' e.3 'z' (e.4);
                    e.3 = <Prout 'No substring a-z found.'>;
                   };
 e.1 = <Prout  'No \'a\' found.'>;
}
```

выделяет первое вхождение в данном тексте строки, начинающейся на `'a'` и кончающейся на `'z'`. В последнем предложении мы «экранировали» две одинарные кавычки символом «\», так как они заключены в другую пару одинарных кавычек.

Задача 1. Рассмотрим другое определение функции из примера №2:

```
a-z-1 {
 e.1 'a' e.2 'z' e.3 = (e.1) 'a' e.2 'z' (e.3);
 e.3 = <Prout 'No substring a-z found.'>;
}
```

Объясните, почему определение `a-z` более эффективно, чем определение `a-z-1`. То есть найдется Рефал данное $\texttt{d}_0$ такое, что время вычисления вызова $\texttt{<a-z d}_0\texttt{>}$ будет *значительно* меньше времени вычисления вызова $\texttt{<a-z-1 d}_0\texttt{>}$, но не существует объектного выражения `d` такого, что время вычисления вызова `<a-z d>` будет *значительно* больше времени вычисления вызова `<a-z-1 d>`?

Вызов функции по определению в Рефале принято называть *блоком*. Присутствие блока в определении функции `F` приводит к *неопределённости понятия*

шага Рефал-машины при исполнении `F`: шаг функции `F` ещё не завершился, а уже внутри него начинают вычисляться вызовы функции, готовящие аргументы – входные данные для блока, да и сам блок вычисляется до окончания рассматриваемого шага функции `F`.

Задача 2. Провести пошаговый просмотр выполнения программы из примера №2 посредством отладчика. Понять и объяснить, какое действие Рефал-машины отладчик воспринимает как «шаг» Рефал-5 машины. Изучить поведение отладчика на последовательности команд `p act; com res;`.

7.2. Домашнее задание.

Задача 3. Глубину пустого выражения положим равной нулю. Глубина `Depth` Рефал-символа также, по определению, равна нулю.

```
Depth((e.d)) = 1+Depth(e.d)
```

Глубиной объектного выражения называется максимум глубин всех термов, входящих в это выражение. Определить на Рефале функцию, значение которой есть глубина данного объектного выражения.

Задача 4. В данном объектном Рефал-выражении (со скобками) уничтожить первое вхождение символа `'a'`, если таковое существует.

Задача 5. Даны две конечные неубывающие последовательности макроцифр. Построить неубывающую последовательность, элементами которой являются все элементы (включая их кратные вхождения) двух данных последовательностей.

Задача 6. Нарисовать ёлочку на экране компьютера средствами псевдографики (посредством символов клавиатуры). Высота ёлочки должна быть равна высоте экрана.

Задача 7. Дана конечная последовательность макроцифр $x_1 = 1$, $x_{n+1} = 1 + x_n$. Построить все различные перестановки данной последовательности.

Задача 8. Дана симметрическая группа перестановок S_n. Придумать представление элементов этой группы во множестве данных Рефала и реализовать групповые операции умножения и построения элемента, обратного к данному элементу.

Задача 9. Рассмотрим полугруппу по умножению квадратных матриц размера $n \times n$, состоящих из целых чисел. Придумать представление элементов этой полугруппы как данных Рефала и реализовать операцию умножения двух матриц.

§ 8. Рефал как метаязык программирования (Лекция №8)

Язык Рефал *ориентирован на преобразование текстов.* Рефал изначально разрабатывался В. Ф. Турчиным как *метаязык*, – то есть инструмент для анализа и преобразования текстов, написанных на других языках, в частности, на языках программирования. Программы (тексты на языках программирования) обычно анализируются либо с целью их перевода на другие языки

программирования, либо с целью их исполнения. В обоих случаях семантика соответствующего языка программирования должна быть определена в терминах Рефала.

Объектами преобразований Рефал-программ могут быть программы, написанные также на языке Рефал. В частности, из алгоритмической полноты Рефала следует, что функция семантики Рефала может быть описана в терминах самого Рефала. Рассмотрим некоторые проблемы, связанные с понятием *метаязыка*.

Перед прочим заметим, что примером метаязыка является русский язык, а примером анализа текстов на русском языке посредством самого языка является предложение, *которое вы сейчас читаете*.

При *самоанализе* возникает проблема с синтаксисом: если мы пишем текст о синтаксисе русского языка, то, например, знаки препинания мы вынуждены обозначать не самими этими знаками как таковыми, а *словами, их обозначающими* – иначе мы нарушим правила самого синтаксиса, о котором рассуждаем. И вместо предложения: *«В конце предложения нужно ставить точку.»* у нас получится предложение: *«В конце предложения нужно ставить ..»*, которое заканчивается знаком горизонтального двоеточия, отсутствующим в русском языке.

В Рефале обсуждаемая проблема проявляется, например, в том, что если требуется, при анализе одной Рефал-программы посредством другой Рефал-программы, определить имя некоторой переменной `s.x`, то переменную `s.x` (а не её значение) невозможно передать *напрямую* (как значение аргумента) для анализа:

```
$ENTRY Go {
  = <Analyze s.x>;
}
```

Снова возникает синтаксическая ошибка: переменная в правой части предложения, которой нет в левой части этого предложения. Как и в русском языке, в подобных случаях мы вынуждены представлять одни понятия языка посредством других. Например, так:

```
$ENTRY Go {
  = <Analyze (Variable 's' x)>;
}
```

Представление одних понятий языка посредством других называется кодировкой.

Кодировка должна позволять восстанавливать смысл закодированного понятия, как и позволять узнавать, является ли то или иное понятие кодировкой (или частью кодировки) другого понятия, или оно представляет само себя. В противном случае неизбежно возникнет путаница. В предложении *«Рассмотрим точку A.»* слово *точка* представляет *само себя*, а не знак, которым заканчивается повествовательное предложение. То же самое можно сказать и о слове *точка* как в предыдущем предложении (второе вхождение), так и в данном предложении. В нашем примере мы смогли определить смысл слова *«точка»* из контекста.

Язык программирования $L = \{D, P, S\}$, где D – множество данных, P – множество программ, S – функция семантики, назовём *метаязыком программирования*, если в нём фиксирована инъективная функция кодировки $\nu : P \to D$.

Инъективность здесь нужна для однозначного восстановления смысла понятий. Далее, для лучшей читаемости, мы будем обозначать отображение кодировки подчёркиванием. Например, $\nu(\texttt{s.x}) = \underline{\texttt{s.x}}$.

Задача 1. Дана матрица M размера $n \times n$, состоящая из нулей и единиц. Придумать представление M в множестве данных Рефала и написать программу, результатом которой является строка длины n, состоящая из нулей и единиц и не являющаяся строкой матрицы M.

8.1. Самоприменимые программы. Программу `p` назовём самоприменимой, если вызов `<p p>` не циклится (работает конечное время).

Пусть $\texttt{p} \in P$. Рассмотрим функцию $\phi : Im(\nu) \to \{\texttt{finite, infinite}\}$[2]

$$\varphi(p) = \begin{cases} \texttt{finite, если <p p> заканчивает работу в конечное время;} \\ \texttt{infinite, если вызов <p p> циклится.} \end{cases}$$

Везде ранее мы рассматривали только *вычислимые* (конструктивные) функции. Другими словами, функции которые можно определить на алгоритмически полном языке. Покажем, что функция φ *не является вычислимой.*

Теорема: Не существует программы, которая распознавала бы самоприменимые программы.

Доказательство будем вести методом от обратного. Предположим, что существует программа `q`, которая реализует функцию φ. Тогда рассмотрим программу `f` такую, что

`<f p> = finite`, если `<q p> = infinite`,
и `<f p>` циклится, если `<q p> = finite`.

Рассмотрим вызов `<f f>`. Предположим что `<f f> = finite`, тогда по определению `f <q f> = infinite` и, следовательно, теперь уже по определению `q`, программа `f` не является самоприменимой, что противоречит нашему предположению.

Следовательно, вызов `<f f>` циклится. Но тогда по определению `f`, `<q f> = finite` и, следовательно, по определению `q`, программа `f` является самоприменимой, что противоречит тому, что вызов `<f f>` циклится.

Таким образом, оба из логически возможных поведений исполнения вызова `<f f>` приводят к противоречию. И наше предположение о существовании программы `q`, реализующей функцию φ, неверно. Доказательство теоремы закончено.

Задача 2. Объясните, каким образом приведённое выше доказательство связано с задачей №1 про матрицу.

Задача 3. Докажите, что нельзя написать программу, которая для любой программы `p` определяла бы, что существует $d_0 \in D$ такое, что `<p` d_0`>` циклится.

[2] $Im(\nu)$ – множество значений (образ) отображения ν.

Множество M называется *счётным*, если его можно пересчитать. Другими словами: существует биекция $\mathbb{N} \to M$.

Задача 4. Докажите, что множество всех действительных чисел из интервала $(0, 1)$ несчётно. Объясните, каким образом утверждение данной задачи связано утверждением о несуществовании программы, которая распознавала бы самоприменимые программы.

8.2. Функция применения. Приведём пример встроенной *мета-функции* Рефала-5. Вызовы

```
<Mu s.fname e.args>
```

и

```
<Mu (e.fname) e.args>
```

вычисляют значение функции с именем, переданным через переменную, на аргументе `e.args`. В первом случае имя вычисляемой функции есть символ Рефала, во втором случае имя функции представлено последовательностью букв.

Задача 5. Даны имя функции и последовательность объектных термов. Требуется определить функцию, значение которой есть последовательность значений данной функции на каждом из термов данной последовательности. Порядок значений должен совпадать с порядком аргументов.

Задача 6. Даны имя функции и последовательность корневых деревьев (объектных Рефал-выражений). Требуется определить программу, заменяющую каждый лист данных деревьев значением данной функции на этом листе.

8.3. Домашнее задание.

Задача 7. Докажите, что ниже перечисленные множества счётны, определите соответствующие биекции на Рефале:

а). множество целых чисел $\mathbb{Z}$;
б). множество $\mathbb{Z}^2$;
в). множество $\mathbb{Z}^3$;
г). множество $\mathbb{Z}^{k_0}$, где k_0 фиксированное натуральное число.

Задача 8. Является ли счётным множество данных Рефала? Если ответ положителен, то определите соответствующую биекцию на Рефале.

Задача 9. Докажите, что нельзя написать программу, которая для любой программы `p` определяла бы, что для всех $d \in D$. `<p d>` $= d$.

Задача 10. Дан объектный терм T_0 и последовательность имён функций. Требуется определить функцию, значение которой есть значение композиции данных функций, первая из которых вычисляется на T_0.

§ 9. Рекурсивные условия (Лекция №9)

Ранее мы ввели в синтаксис Рефал-программ новое понятие – запятую, тем самым выйдя за рамки базисного Рефала. Кроме отделения левой части от

правой части предложения, при вызове функции по определению знак запятой используется для организации *рекурсивного условия выбора предложения* внутри одной функции. В обоих случаях знак запятой можно понимать как логическую связку – конъюнкцию (то есть «и»).

9.1. Другая интерпретация знака запятой. Вернёмся к нашему примеру выделения в данной строке подстроки, начинающейся на `'a'` и кончающейся на `'z'`. Ранее мы запрограммировали его, используя блок (вызов функции по определению).

```
a-z {
 e.1 'a' e.2, e.2: {
                    e.3 'z' e.4 = (e.1) 'a' e.3 'z' (e.4);
                           e.3 = 'No substring a-z found.';
                   };
 e.1 = 'No \'a\' found.';
}
```

Смысл пары запятая-двоеточие в первом предложении

```
e.1 'a' e.2, e.2:
```

можно понимать и так: если аргумент вызова функции `a-z` является Рефал-выражением вида `e.1 'a' e.2` и значение переменной `e.2` такое, что ... (далее см. определение блока). Или ещё так: если аргумент вызова функции `a-z` является объектным Рефал-выражением вида `e.1 'a' e.2`, где значение переменной `e.2` такое, что ... (далее см. определение блока). Именно такой смысл пары запятая-двоеточие вкладывается в «`, expression :`» при введении в *расширенный* Рефал-рекурсивных условий.

9.2. Рекурсивные условия выбора предложения. В *базисном* Рефале Рефал-машина использует структуру образца и алгоритм сопоставления данных-аргументов функции `F` с образцом для выбора предложения из последовательности предложений, определяющей тело функции `F`. Механизм отождествления не является «алгоритмически полным». (Например, он не может иметь бесконечных циклов.) Используем отмеченный выше смысл синтаксической конструкции «`, expression :`» для введения алгоритмически полного инструмента выбора Рефал-предложения.

В *расширенном* Рефале Рефал-предложение может иметь один из следующих видов:

1. `pattern = expression;`
2. `pattern , arguments :  { block };`
3. `pattern conditions right_part;`

где `right_part` есть либо «`= expression`», либо «`, arguments : { block }`», а

`conditions ::=` *`, expression : pattern`* `conditions` | *`empty`*

Причём `expression` имеет тот же вид что и правая часть предложения в базисном Рефале, и удовлетворяет тем же синтаксическим ограничениям: в `expression` допускаются только переменные, которые уже встречались в рассматриваемом предложении левее данного `expression`.

Мы начнём с простейшего примера, показывающего, что даже при отсутствии рекурсии в выборе предложения, новая конструкция *условия* позволяет писать программы более компактно.

Пример №1: Пусть функция `Look-For-I`(x_1, x_2, x_3), где x_i – строки символов, есть предикат, отвечающий на вопрос «Принадлежит ли символ `'I'` хотя бы одной строке x_i?». Тогда в базисном Рефале определение `Look-For-I`, если не использовать вспомогательных функций, выглядит так:

```
Look-For-I {
 (e.x 'I' e.x1) (e.x2)         (e.x3) = True;
 (e.x1)         (e.y 'I' e.x2) (e.x3) = True;
 (e.x1)         (e.x2)         (e.z 'I' e.x3) = True;
 (e.x1)         (e.x2)         (e.x3)         = False;
}
```

Используя блок, его можно упростить следующим образом:

```
Look-For-I {
(e.x1) (e.x2) (e.x3), e.x1 e.x2 e.x3 : {
                                        e.x 'I' e.y = True;
                                        e.z = False;
                                       };
}
```

Использование условия позволяет написать эту программу ещё короче:

```
Look-For-I {
 (e.x1) (e.x2) (e.x3), e.x1 e.x2 e.x3 :  e.x 'I' e.y = True;
  e.z = False;
}
```

9.2.1. *Откаты при выборе предложения.*

Пример №2: Дана строка `e.string` и лес деревьев `e.forest`. Требуется найти самое левое дерево, среди листов которого содержится самый левый символ строки `e.string`, из тех которые являются листьями данного дерева. Определим соответствующую программу, используя рекурсию в выборе предложений.

```
Tree {
 e.x s.leaf e.string (e.y t.tree e.forest)
     , <Get-leaves t.tree>: e.z s.leaf e.leaves = t.tree;
 e.x                 (e.forest)  = Fail;
}

Get-leaves {
 e.x (e.y) e.z = e.x <Get-leaves e.y e.z>;
 e.x = e.x;
}
```

Здесь рекурсивно происходят откаты при выборе Рефал-машиной предложения функции `Tree`. Два образца

`e.x s.leaf e.string (e.y t.tree e.forest)` и `s.z s.leaf e.leaves` имеют общую переменную `s.leaf` и взаимодействуют при работе алгоритмов отождествления. Согласно общему правилу Рефала при решении уравнения

`e.x s.leaf e.string (e.y t.tree e.forest) = string`$_0$ `(forest`$_0$`)`

, где `string`$_0$ и `forest`$_0$ – данные Рефала, из всего множества решений M_i

1. выбираются те решения, в которых `e.x` приняла значение с наименьшей возможной длиной (количеством деревьев в `e.x`$_0$),
2. если такой выбор не приводит к однозначности решения, тогда из выбранного подмножества решений выбираются те решения, в которых `e.y` приняла значение с наименьшей возможной длиной (количеством деревьев в `e.y`$_0$). Последний выбор всегда приведёт к однозначности, если множество решений не пусто.

Таким образом, в результате решения указанного выше уравнения Рефал-машина имеет значения переменных `t.tree = t.tree`$_0$`, s.leaf = s.leaf`$_0$ и может вычислить вызов `<Get-leaves t.tree`$_0$`>`. Пусть результат этого вычисления равен `leaves`$_0$, тогда Рефал-машина решает уравнение, определённое условием в первом предложении функции `Tree`:

`e.z s.leaf e.leaves = leaves`$_0$

если это уравнение имеет решение, тогда Рефал-машина нашла дерево `t.tree`$_0$ с требуемым свойством. Иначе Рефал-машина возвращается к уравнению

`e.x s.leaf e.string (e.y t.tree e.forest) = string`$_0$ `(forest`$_0$`)`

и считает, что выбранное ранее его решение

`e.x`$_0$`, s.leaf`$_0$`, e.string`$_0$`, e.y`$_0$`, t.tree`$_0$`, e.forest`$_0$

не удовлетворяет условию выбора данного предложения. Пусть $ln($`e.y`$_0)$ есть длина (количество термов-деревьев в) `e.y`$_0$, тогда множество решений данного уравнения сужается условием $ln($`e.y`$) > ln($`e.y`$_0)$.

- Если получившееся подмножество не пусто, то обозначим его $K_{i+1} \subset M_i$ ($K_{i+1} \neq M_i$), переименуем его M_i := K_{i+1} и переходим к пункту 1).
- Иначе Рефал-машина снова возвращается к решению уравнения
  `e.x s.leaf e.string (e.y t.tree e.forest) = string`$_0$ `(forest`$_0$`)`
  Множество решений M_i данного уравнения сужается условием $ln($`e.x`$) > ln($`e.x`$_0)$. Обозначим получившееся подмножество $Q_{i+1} \subset M_i$ ($Q_{i+1} \neq M_i$), переименуем его M_i := Q_{i+1} и переходим к пункту 1).

Задача 1. Доказать, что определённая выше функция `Tree` заканчивает работу за конечное время при любых значениях её аргументов.

Задача 2. Каковы будут значения переменных `e.x` и `e.y` при первом вызове функции `Get-leaves`? Зависят ли эти значения от аргументов вызова функции `Tree`?

Задача 3. Чему равно максимальное число откатов, которое сделает Рефал-машина, при поиске нужного решения уравнения
`e.x s.leaf e.string (e.y t.tree e.forest) = string`$_0$ `(forest`$_0$`)` ?

Вставим в нашу программу отладочную печать, чтобы проследить работу Рефал-машины в деталях.

```
$ENTRY Go {
 = <Prout '\nThe result: ' <Tree A B C ((E) (D (B)) ((B) (N C)))>>;
}

Tree {
 e.x s.leaf e.string (e.y t.tree e.forest),
   <Prout 'Try the leaf: ' s.leaf ' in the tree: ' t.tree
          '    \n\t , where the length of e.y=' e.y ' is ' <Ln e.y>>
   <Prout '    \t   and the length of e.x=' e.x ' is ' <Ln e.x>>
   <Get-leaves t.tree>: e.z s.leaf  e.leaves = t.tree;
 e.x  (e.forest)  = Fail;
}

Get-leaves {
 e.x (e.y) e.z = e.x <Get-leaves e.y e.z>;
 e.x = e.x;
}

Ln {
            = 0;
 t.term e.x = <+ (<Ln e.x>) 1>;
}
```

В программе мы использовали символ перевода строки `'\n'` и символ горизонтальной табуляции `'\t'`, также мы воспользовались тем, что значение вызова `<Prout e.args>` всегда равно пустому выражению. В результате исполнения этой программы на экране компьютера имеем:

```
Try the leaf: A  in the tree: (E )
         , where the length of e.y= is 0
           and the length of e.x= is 0
Try the leaf: A  in the tree: (D (B ))
         , where the length of e.y=(E ) is 1
           and the length of e.x= is 0
Try the leaf: A  in the tree: ((B )(N C ))
         , where the length of e.y=(E )(D (B )) is 2
           and the length of e.x= is 0
Try the leaf: B  in the tree: (E )
         , where the length of e.y= is 0
           and the length of e.x=A  is 1
Try the leaf: B  in the tree: (D (B ))
         , where the length of e.y=(E ) is 1
           and the length of e.x=A  is 1

The result: (D (B ))
```

Задача 4. Выполните данную выше программу с отладочной печатью на вашем компьютере. Вставьте дополнительную печать, показывающую как из-

меняется при выборе предложения длина значения переменной `e.z`. Объясните полученный вами результат.

9.2.2. *Локальное присваивание.* Синтаксическую конструкцию условия можно также использовать для устранения повторных вычислений. В этом случае само условие тривиально – всегда истинно, и мы используем его конструкцию как конструкцию локального присваивания.

Пример №3: Пусть нам нужно определить функцию на множестве целых чисел $g(n) = (n-1) * (n-1)$. Определим `g` в базисном Рефале:

```
g {
  e.n = <* (<- (e.n) 1>) <- (e.n) 1>>;
}
```

Использование синтаксической конструкции условия позволяет устранить повторное вычитание 1.

```
g {
  e.n, <- (e.n) 1>: e.m = <* (e.m) e.m>;
}
```

Задача 5. Переопределить данную в примере №3 функцию `g` в базисном Рефале, но без повторного вычитания 1.

Присутствие условия в определении функции `F` может привести к *неопределённости понятия шага Рефал-машины* при исполнении `F` (как и присутствие блока в `F`): шаг функции `F` ещё не завершился, а уже внутри него начинают вычисляться вызовы функции, готовящие аргумент для этого условия.

Задача 6. Провести пошаговый просмотр выполнения программы из примера №2 посредством отладчика. Понять и объяснить какое действие Рефал-машины отладчик воспринимает как «шаг» Рефал-5 машины. Изучить поведение отладчика на последовательность команд `p act; com res`.

9.3. Домашнее задание.

Задача 7. Переопределить данную в примере №2 функцию `Tree`, ограничившись синтаксисом базисного Рефала.

Задача 8. Дано объектное Рефал-выражение d_0, в которое могут входить структурные скобки. Определить функцию f поиска первого вхождения знака аддитивной операции (`'+'` или `'-'`) на верхнем уровне скобочной структуры данного d_0. Значением f должно быть исходное d_0, в котором часть предшествующая найденному знаку аддитивной операции заключена в структурные скобки. Если требуемого знака не существует, то функция f должна выдавать не изменённое d_0.

Задача 9. Дан лес деревьев (объектное Рефал-выражение со скобками). Требуется определить на Рефале функцию, переставляющую листья деревьев в обратном порядке, если листья находятся на нечётной глубине, и не меняющую порядок листьев, находящихся на чётной глубине. Структуру деревьев не изменять. Задачу решить в двух вариантах:

а). Локально, то есть переставляются в каждом дереве листья, находящиеся на ветках одного и того же ближайшего к ним узла-ветвления. Например,

`(a b c) d t (f g 1 (3 4) 5)` $\Rightarrow$ `(c b a) d t (5 1 g (3 4) f)`;

б). Глобально, то есть последовательности листьев, находящихся на глубине $2 * n + 1$, приписываются друг к другу согласно следованию – слева направо. После этого элементы полученной последовательности переставляются в обратном порядке и затем развешиваются на соответствующие ветви деревьев.

Например, для леса `(a b c) d t (f g 1 (3 4) 5)` имеем последовательность листьев, соответствующую глубине 1: `a b c f g 1 5`. После перестановки элементов этой последовательности она преобразуется в `5 1 g f c b a`. Теперь развешиваем листья на деревья и получаем: `(5 1 g) d t (f c b (3 4) a)`.

§ 10. Сложность Колмогорова (Лекция №10)

Рефал ориентирован на *преобразование текстов*. Какие тексты являются простыми, а какие сложными? Классическое рифмованное стихотворение выучить наизусть много проще, чем белый стих. Кроме того, понятие сложности текста `T` зависит от нашего понимания `T`. Запоминание доказательства теоремы при подготовке к экзамену является делом безнадёжным, когда студент не понимает этого доказательства; в то же время, если доказательство понято, то оно легко восстанавливается, исходя из логики взаимозависимости его шагов. Великий русский математик Андрей Николаевич Колмогоров предложил *аппроксимировать понятие взаимозависимости частей данного текста посредством рекурсии.*

10.1. Сложность текста. Иногда возможно длинный текст описать посредством короткого. Например, текст

`миллион слов 'рефал'`,

состоящий из 20 символов, является описанием текста длиной пять миллионов символов. Ранее мы уже пробовали описывать русскую народную сказку «Репка» посредством как можно более короткой Рефал-программы.

Задача 1. Дано стихотворение Владимира Солоухина, рассмотренное нами на лекции №5 – задача №9. Обозначим его T_1. Создать текст T_2, состоящий из одних нулей, длина которого равна длине текста T_1. Из текста данной лекции №5 вырезать кусок T_3; длина текста T_3 должна равняться длине текста T_1. Сжать тексты T_1, T_2, T_3 всеми известными вам упаковщиками («архиваторами»), составить таблицу (одна строчка для каждого упаковщика) длин получившихся текстов и сравнить эти длины.

Пусть `p` – какая-нибудь программа. *Сложностью* текста T при способе описания `p` называется наименьшая длина текста Z такого, что `<p` Z`>` = T. При этом текст Z называется *описанием* текста T.

Иными словами, мы рассматриваем все сжатые версии текста T, из которых данный распаковщик `p` может восстановить T, и среди них выбираем самое короткое описание. Например,

```
миллион слов 'рефал'
```

и

```
пятьсот тысяч слов 'рефалрефал'
```

– два описания одного текста, но первое описание короче.

Некоторые упаковщики создают самораспаковывающиеся архивы. В таких архивах содержится и распаковщик, и заархивированный текст. Текст распаковщика должен быть каким-то образом отделён от заархивированного текста.

10.2. Структурные скобки и кодирование разделения аргументов. С подобной проблемой мы уже встречались: в Рефале все функции одноместны (имеют только один аргумент). Мы использовали конструктор структурных скобок, чтобы имитировать функции нескольких переменных. То есть, мы кодировали *понятие разделения аргументов* посредством структурных скобок. Другими словами, мы структурировали данные Рефала. И, как мы знаем, структурные скобки позволяют удобно структурировать Рефал-данные в виде леса корневых деревьев; в этом основная роль структурных скобок. За такую возможность мы заплатили ограничением алфавита: скобка `(` не может представлять сама себя в тексте Рефал-программы, – синтаксис требует закодировать её обрамлением в одинарные или двойные кавычки (так `'('` или так `"("`).

Оказывается, можно закодировать понятие разделения аргументов, не вводя никаких дополнительных ограничений на алфавит, если алфавит содержит по крайней мере две буквы. Разделить распаковщик и заархивированный текст можно, например, таким образом:

```
рраассппааккууюющщаааяя  ппррооггрраамммммаааабзапакованный текст
```

где мы каждую букву исходного текста распаковщика продублировали (после `яя` стоят два пробела), а сразу после закодированного текста распаковщика мы написали буквы `аб` ; сам запакованный текст никак не изменили.

Задача 2. Докажите, что указанное кодирующее отображение инъективно.

Задача 3. Решите следующую задачу, не используя структурные скобки. Даны текст и строка. Определить на Рефале функцию, значение которой есть отрезок данного текста, непосредственно следующий за *первым вхождением* данной строки, если такое вхождение существует, и сама данная строка, если она не входит в текст.

10.3. Теорема Колмогорова. Пусть дан алгоритмически полный язык программирования L (например, Рефал без структурных скобок), на котором мы намерены реализовывать упаковывающие программы. Пусть D есть множество данных языка L, а P – множество программ языка L.

Теорема 1. Существует способ, который даёт почти самые короткие описания всех текстов. То есть, существует программа `p` такая, что для любого текста T сложность описания T посредством `p` меньше, чем сложность описания T посредством любой другой программы `q` плюс некоторая константа

`Const`, зависящая только от `q`. То есть, `Const` не зависит от T. Обозначим через $\varphi_{\mathtt{p}}(T)$ сложность описания T посредством `p`, тогда

$$\exists p \in P\ \forall q \in P\ \exists \mathtt{Const} \in \mathbb{R}\ \forall T \in D.\ \varphi_{\mathtt{p}}(T) < \varphi_{\mathtt{q}}(T) + \mathtt{Const}$$

Теорема говорит о том, что можно написать универсальный почти оптимальный упаковщик `p`. Любой другой упаковщик `q` будет сжимать исходные тексты хуже, чем `p` – с точностью до некоторой константы.

Докажем теорему. Данный язык L, по условию, алгоритмически полон. Следовательно, мы можем реализовать на языке L *универсальную программу* $\mathtt{Int} \in P$, которая умеет вычислять значение любой другой данной программы $\mathtt{q} \in P$ (написанной на языке L) на любом конкретном $\mathtt{d}_0 \in D$. Программа `q` есть отображение из D в $D_\perp$ (смысл обозначения см. в лекции №2). Следовательно,

$$\mathtt{Int} : P \times D \to D_\perp$$

Условие универсальности программы `Int` формально выглядит так:

$$\mathtt{<Int\ \underline{qd}>\ =\ <q\ d>}$$

где мы намеренно не разделили программу `q` и её данные `d`, – разделение делает кодировка, обозначенная нами подчёркиванием. Универсальные программы программисты называют *интерпретаторами* (interpreter – исполнитель).

Пусть у нас имеется произвольный способ описания $\mathtt{q} \in P$, тогда длину $\underline{\mathtt{q}Z}$ можно ограничить числом $2 * k + n + 2$, где k – длина `q`, n – длина текста Z, являющего описанием текста T при способе `q`. Далее,

$$\forall Z\ \mathtt{<Int\ \underline{qZ}>\ =\ <q\ Z>\ =}\ T$$

то есть пара (`Int`, $\underline{\mathtt{q}Z}$) распаковывает текст T при способе `Int`. Следовательно, по определению описания,

$$\varphi_{\mathtt{Int}}(T) < 2 * k + n + 3$$

По определению, Z есть описание T при способе `q`. То есть $n = \varphi_{\mathtt{q}}(T)$. Следовательно,

$$\forall q\ \varphi_{\mathtt{Int}}(T) < \varphi_{\mathtt{q}}(T) + 2 * k + 3$$

где k – длина самого способа `q` (длина исходного текста программы `q`), которая зависит только от `q` и не зависит от T. Что и требовалось доказать.

10.4. Домашнее задание.

Задача 4. Выше мы описали кодировку понятия разделения аргументов.

а). Реализуйте описанную кодировку на Рефале.

б). Реализуйте описанную кодировку на Рефале, не используя структурные скобки.

Задача 5. Ограничим алфавит Рефал-символов тремя символами `'a'`, `'b'`, `'_'`. Получившийся язык обозначим $\text{Рефал}_{\mathtt{ab\_}}$. Натуральные числа в языке $\text{Рефал}_{\mathtt{ab\_}}$ будем представлять в троичной системе счисления, цифрами в которой являются символы `'_'`, `'a'`, `'b'` (`'_'` есть 0, `'a'` есть 1, `'b'` есть 2).

Рассмотрим следующую кодировку понятия разделения аргументов. Пусть даны два текста-аргумента `X` и `Y`; вычислим длину k первого аргумента `X`. Пусть $N(k)$ – представление числа k в троичной системе счисления с цифрами `'_'`, `'a'`, `'b'`. Рассмотрим пару $N(k)$ и `XY`, где мы текст `Y` приписали к тексту `X` без какого-либо разделителя между ними. Закодируем пару $N(k)$ и `XY` посредством удвоения букв в тексте $N(k)$ и отделением первого элемента пары от второго двумя неравными буквами (как мы описали выше в нашей лекции).

Докажите, что построенное отображение, кодирующее два текста–аргумента `X` и `Y`, инъективно.

а). Реализуйте описанную кодировку на Рефале.

б). Реализуйте описанную кодировку на Рефале, не используя структурные скобки.

в). Реализуйте описанную кодировку на языке Рефал$_{\texttt{ab_}}$.

г). Реализуйте описанную кодировку на языке Рефал$_{\texttt{ab_}}$, не используя структурные скобки.

Список литературы

[1] V. F. Turchin, *REFAL-5 programming guide and reference manual*, (переработанное и расширенное издание 1999 года доступно как zipped html-файл: http://refal.botik.ru/book/refal-book-html.zip), New England Publishing Co., Holyoke, 1989.

[2] V. F. Turchin, D. V. Turchin, A. P. Konyshev, A. P. Nemytykh, *Refal-5: Sources, Executable Modules*, ([online]: http://www.botik.ru/pub/local/scp/refal5/), 2000.

[3] Р. Ф. Гурин, С. А. Романенко, *Язык программирования РЕФАЛ ПЛЮС*, ИНТЕРТЕХ, Москва, 1991.

[4] А. В. Корлюков, *Введение в программирование на языке РЕФАЛ с приложениями в алгебре*, http://www.refal.net/~korlukov/refbook/index.htm, 2001.

[5] Н. Н. Воробьев, *Числа Фибоначчи*, Наука, Москва, 1978.

[6] Э. Мендельсон, *Введение в математическую логику*, Наука, Москва, 1971.

[7] А. Л. Семёнов, *Математика текстов*, МЦНМО, Москва, 2002.

[8] А. Шень, *Программирование: теоремы и задачи*, (доступна как zipped PDF-файл: ftp://ftp.mccme.ru/users/shen/progbook2/progbookpdf.zip), МЦНМО, Москва, 2004.

[9] И. А. Николаев, *Словарь по литературоведению*, ([online]: http://nature.web.ru/litera/11.2.html), Научная Сеть, 2004.

[10] В. Солоухин, *Стихотворения*, (цитируемое стихотворение доступно в файле: http://botik.ru/pub/local/scp/ugp/venok.html), Советская Россия, Москва, 1990.

А. П. Немытых (A. P. Nemytykh)
Переславль-Залесский, ИПС РАН
E-mail: nemytykh@math.botik.ru