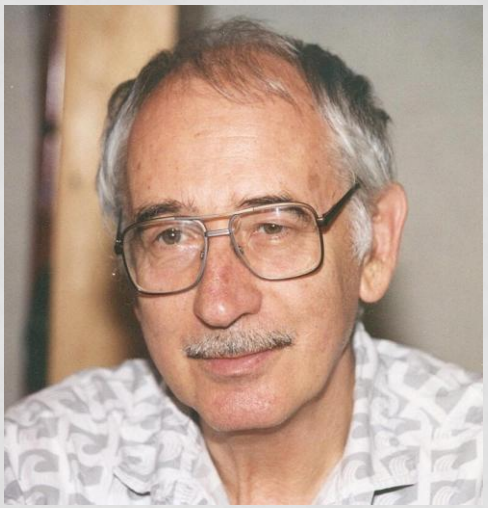
ДУБОВ МИХАИЛ СЕРГЕЕВИЧ,

СТУДЕНТ II КУРСА
ОТДЕЛЕНИЯ ПРОГРАММНОЙ
ИНЖЕНЕРИИ ФАКУЛЬТЕТА
БИЗНЕС-ИНФОРМАТИКИ
НИУ ВШЭ

РЕФАЛ:

АЛГОРИТМИЧЕСКИЙ ЯЗЫК РЕКУРСИВНЫХ ФУНКЦИЙ

РЕФАЛ



- Рефал – функциональный язык, созданный в 1966-1968 гг. в СССР физиком и кибернетиком **Валентином Турчиным** (1931-2010).
- Изначально задумывался как **метаязык** для описания семантики других языков, однако нашел практическое применение и как язык программирования.
- Рефал основан на ***нормальных алгорифмах Маркова***.

НОРМАЛЬНЫЕ АЛГОРИФМЫ МАРКОВА

- Введены А. Марковым-мл. в 40-х годах для формализации понятия **«алгоритм»**.
- Использовались для решения задач по определению алгоритмически неразрешимых проблем.

Нормальный алгорифм Маркова =

алфавит + правила подстановки

НОРМАЛЬНЫЕ АЛГОРИФМЫ МАРКОВА

- **На входе** – любая непустая строка символов алфавита.

1. $ab \to bd$
2. $db \to ba$
3. $bba \to abb$
4. $c \to \lambda$

- **Шаг алгоритма**:
  - Ищем первую «подходящую» подстановку;
  - Ищем в строке самое левое вхождение левой части подстановки;
  - Заменяем это вхождение на правую часть подстановки.

- **Остановка**, если:
  - Нет подходящих подстановок;
  - Установлено, что процесс подстановок бесконечен.

ПРИМЕР

Перевод числа из двоичной в унарную с.с.

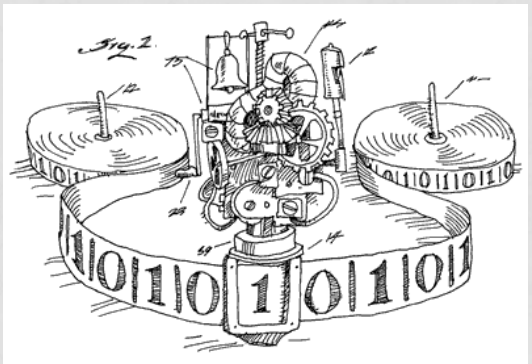
- Алгорифм:
  - Алфавит: {‘0’, ‘1’, ‘|’, λ}
  - Формулы подстановки:
    - |0 → 0||
    - 1 → 0|
    - 0 → λ
- Пример выполнения:

101 → 0|01 → 00||1 → 00||0| → 00|0||| →
000||||| → 00|||||| → 0|||||| → ||||||

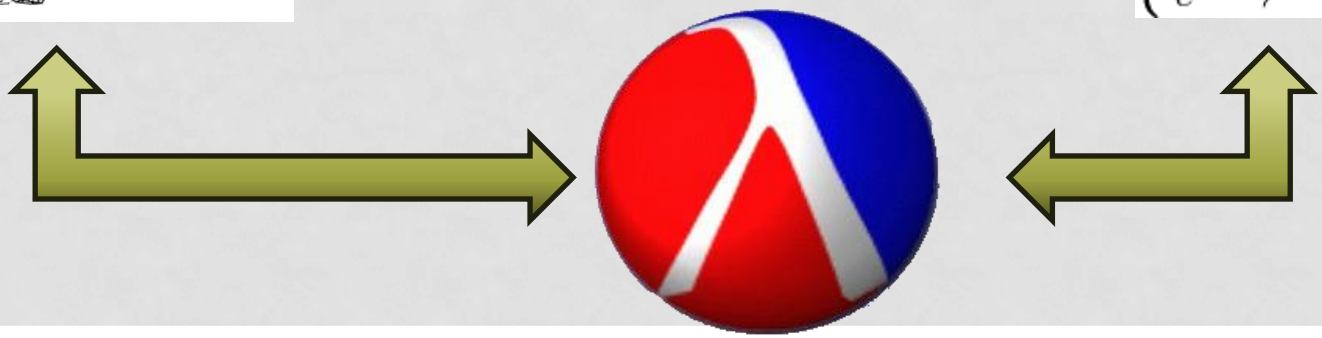
ПРИНЦИП НОРМАЛИЗАЦИИ

- Вариант тезиса Чёрча-Тьюринга для НАМ.

Любой нормальный алгорифм эквивалентен некоторой машине Тьюринга, и наоборот — любая машина Тьюринга эквивалентна некоторому нормальному алгорифму.



$$\begin{cases} |b \to ba| \\ ab \to ba \\ b \to \\ *| \to b* \\ * \to c \\ |c \to c \\ ac \to c| \\ c \to \cdot \end{cases}$$



ЛИТЕРАТУРА

- Марков А.А. *Теория алгорифмов*. – Труды АН СССР, Ин-т математики. – 1954. – 42. – 376 с.
- Пильщиков В.Н., Абрамов В.Г., Вылиток А.А., Горячая И.В. *Машина Тьюринга и алгоритмы Маркова. Решение задач*. - М.: ВМиК МГУ, 2006. – 47 с.

ЯЗЫК ПРОГРАММИРОВАНИЯ РЕФАЛ

- **Рефал** (рекурсивных функций алгоритмический) – функциональный язык программирования. Основой определения функций в рефал'е являются **два механизма**:

Сопоставление с образцом

и

Подстановка (переписывание термов)

- Программы на рефале выполняются абстрактной **Рефал-машиной**.

РЕФАЛ: ФУНКЦИИ

- Программа на Рефале состоит из функций.
- Функция состоит из предложений – правил подстановки:

```
Имя_функции
{                                   // Левая и правая часть могут
   образец = результат;             // содержать переменные;
   образец = результат;             // Правые части предложений
                …                   //  могут также содержать
};                                  //  вызовы функций.
```

- Вызов функции:

```
<Имя_функции   аргумент_функции>
```

HELLO WORLD

```
$ENTRY Go
{
    = <Prout 'Hello World!'>;
};
```

- Служебным словом $ENTRY помечается функция, являющаяся точкой входа в программу;
- Функция Go состоит из одного предложения:
  - Левая часть – пустое выражение (интерпретатор вызывает ENTRY-функцию именно с этим аргументом);
  - Правая часть – вызов библиотечной функции Prout, которая выводит на экран строку и возвращает пустое выражение.

РЕФАЛ: ПЕРЕМЕННЫЕ

- В рефале **3 типа переменных**:
  - **s** – для обозначения **символов**

    → “атомы”: литеры, целые и вещественные числа:

s.1 s.2 s.3	‘ref’	s.1 = ‘r’, s.2 = ‘e’, s.3 = ‘f’

  - **t** – для обозначения **термов**

    → терм – символ или выражение в круглых (“структурных”) скобках:

t.word1 (s.1 s.2 s.3)	(‘tree’)(‘map’)	t.word1 = (‘tree’), s.1 = ‘m’, s.2 = ‘a’, s.3 = ‘p’

  - **e** – для обозначения **объектных выражений**

    → объектное выражение – последовательность термов, возможно, пустая.

НОРМАЛЬНЫЙ АЛГОРИФМ МАРКОВА

- Реализация рассмотренного выше **алгорифма Маркова** на Рефале:

```
$ENTRY Go
{
   = <Prout <BinToUn <Card>>>;
};

BinToUn
{   /* Область определения переменной – одно предложение */
   e.1  '|0'  e.2 = <BinToUn e.1  '0||'  e.2>;
   e.1  '1'  e.2 = <BinToUn e.1  '0|'  e.2>;
   e.1  '0'  e.2 = <BinToUn e.1 e.2>;
   e.1 = e.1;
};
```

- Card – стандартная функция ввода.

РЕФАЛ: ПЕРЕМЕННЫЕ

- **Открытые переменные** – переменные, которым ещё НЕ задано значение.
- **Закрытые переменные** – ссылки на открытые переменные.

- Например, в образце

```
ABCD s.1 F e.2 XW s.1
```

первая слева переменная **s.1** и переменная **e.2** являются открытыми; вторая переменная **s.1** закрытая, её значение должно совпадать со значением первой переменной **s.1**.

ПАЛИНДРОМЫ

```
Palindrom
{
    s.1 e.2 s.1 = <Palindrom e.2> ;
    s.1 = 'True' ;
     = 'True';
    e.1 = 'False' ;
};
```

<Palindrom ‘abcba’> <Palindrom ‘bcb’> <Palindrom ‘c’> True	<Palindrom ‘abba’> <Palindrom ‘bb’> <Palindrom ‘’> True	<Palindrom ‘abca’> <Palindrom ‘bc’> False

ОБРАЩЕНИЕ СТРОКИ

- Простая рекурсия:

```
Reverse {
    /* empty */ = /* empty */;
    s.Next e.Tail = <Reverse e.Tail> s.Next;
};
```

- Хвостовая рекурсия с аккумулятором:

```
Reverse{
    e.String = <RecReverse () e.String>;
};

RecReverse {
    (e.Reversed) = e.Reversed;
    (e.Reversed) s.Next e.Tail = <RecReverse (s.Next e.Reversed) e.Tail>;
};
```

РЕФАЛ: РАБОТА С ЧИСЛАМИ

- **Числа** распознаются Рефалом как символы (**s-переменные**).
- Этот составной символ-число называется **макроцифрой**.
- Стандартная функция **преобразования** строки цифр в макроцифру – Numb:

```
<Numb '123'>
```

- **Арифметические операции** – в **префиксной** записи:

```
<+ 2 3>      // равно 5
```

ФАКТОРИАЛ

- Простая рекурсия:

```
$ENTRY Go
{
   = <Prout <Fact<Numb  <Card>>>>;
};


Fact
{
   0 = 1;
   s.Num = <* s.Num <Fact <- s.Num 1>>>;
};
```

ФАКТОРИАЛ

- Хвостовая рекурсия с аккумулятором:

```
$ENTRY Go
{
   = <Prout <Fact<Numb  <Card>>>>;
};

Fact
{
   s.Num = <RecFact s.Num 1>;
};

RecFact
{
   0 s.Acc = s.Acc;
   s.N s.Acc = <RecFact <- s.N 1> <* s.Acc s.N>>
};
```

РЕФАЛ: WHERE-КОНСТРУКЦИЯ

- В средства *расширенного Рефала* входят так называемые **where-выражения**, которые накладывают дополнительные условия на применимость предложения (конкретизируют образец):

```
Функция
{
    образец = результат;
    образец, функция:
    {
        образец = результат;
        образец = результат;
    }
};
```

ДВОИЧНОЕ ДЕРЕВО ПОИСКА

- Дерево либо пусто, либо имеет вид:

(левое поддерево) число (правое поддерево)

- Функция Insert вставляет число в дерево:

```
Insert
{
    s.x = () s.x ();
    s.x (e.1) s.y (e.2),  <Compare s.x s.y>:
    { // Compare возвращает  один из символов '-'/'+'/'0'
        '-' = (<Insert s.x e.1>) s.y (e.2);
        s.c = (e.1) s.y (<Insert s.x e.2>)
    }
};
```

ДВОИЧНОЕ ДЕРЕВО ПОИСКА

- Функция BuildTree строит дерево из последовательности чисел (макроцифр) в структурных скобках, накапливая результат во втором аргументе:

```
BuildTree
{
    () e.r = e.r;
    (s.x e.z) e.r = <BuildTree (e.z) <Insert s.x e.r>>
};
```

ДВОИЧНОЕ ДЕРЕВО ПОИСКА

- Функция NumbList преобразует входную строку из чисел, разделенных пробелами, в последовательность макроцифр:

```
NumbList
{
    e.Input = <RecNumbList () e.Input>;
};

RecNumbList
{
    (e.Numbs) e.Next ' ' e.Tail = <RecNumbList (e.Numbs <Numb e.Next>) e.Tail>;
    (e.Numbs) e.Next = (e.Numbs <Numb e.Next>);
};
```

РЕФАЛ: ДОСТОИНСТВА

- **Декларативный**, а не командный и не основанный на понятии состояния, стиль программирования. Рефал — один из первых и очень немногих таких языков.
- **Сопоставление с образцом** как способ определения функций и структурирования вычислений разветвлением. И в этом отношении Рефал сильно опередил другие языки.
- **У функции лишь один аргумент и результат**. Этот принцип хорошо известен по современным функциональным языкам, где ему следуют практически везде, но сорок с лишним лет назад господствовали другие представления.
- Реализация посредством компилирования исходной программы в программу на **языке виртуальной машины**, которая затем интерпретируется.

РЕФАЛ: НЕДОСТАТКИ

- У Рефала отсутствует возможность построения и именования **структур данных** и тем более — задания определенных программистом **типов**.
- «Знание **типов**» со стороны языка исчерпывается различением атомарного значения от составного (последовательности) и видов атомарных значений (числа, литеры и символы). Это приемлемо для небольших программ, но сильно затрудняет создание крупных.
- **Отсутствие** в языке **булевых значений** и **булевой арифметики**. Нет и какой бы то ни было формы **условного ветвления**.
- Функции не могут быть **вложенными**.
- Недоступна возможность создания **функций высшего порядка**. Функции в Рефале не есть значения: нельзя создавать безымянные функции, тем более замыкания.

РЕФАЛ: ПРИМЕНЕНИЯ

- Задачи **анализа и преобразования текста** (например, XML/HTML, TeX и др.);
- Алгоритмы над абстрактными структурами – в области **искусственного интеллекта**;
- Оптимизирующие преобразования программ – **суперкомпиляция**.

РЕФАЛ: ДИАЛЕКТЫ

- **Рефал-2** – самый ранний;
- **Рефал-5** – 1980-е – «классический» диалект;
- **Рефал-6** – решает ряд недостатков языка; единственный диалект с реализацией операций над вещественными числами;
- **Рефал+** – самый продвинутый диалект.

ЛИТЕРАТУРА

- Содружество “РЕФАЛ/Суперкомпиляция” (http://refal.net/):
  - Документация, книги по языку;
  - Компиляторы всех основных диалектов.
- Банчев Б. “Язык Рефал – взгляд со стороны” – журнал “Практика функционального программирования”, выпуск 7;
- Васильева Е. “Самостоятельное изучение Рефала-5. Взгляд студента” (http://bit.ly/yUKKRr);
- Турчин В. “Рефал как язык для обработки xml-документов” – журнал “Компьютерра”, №25
- Рублев В. Курс “Языки логического программирования” (http://intuit.ru)