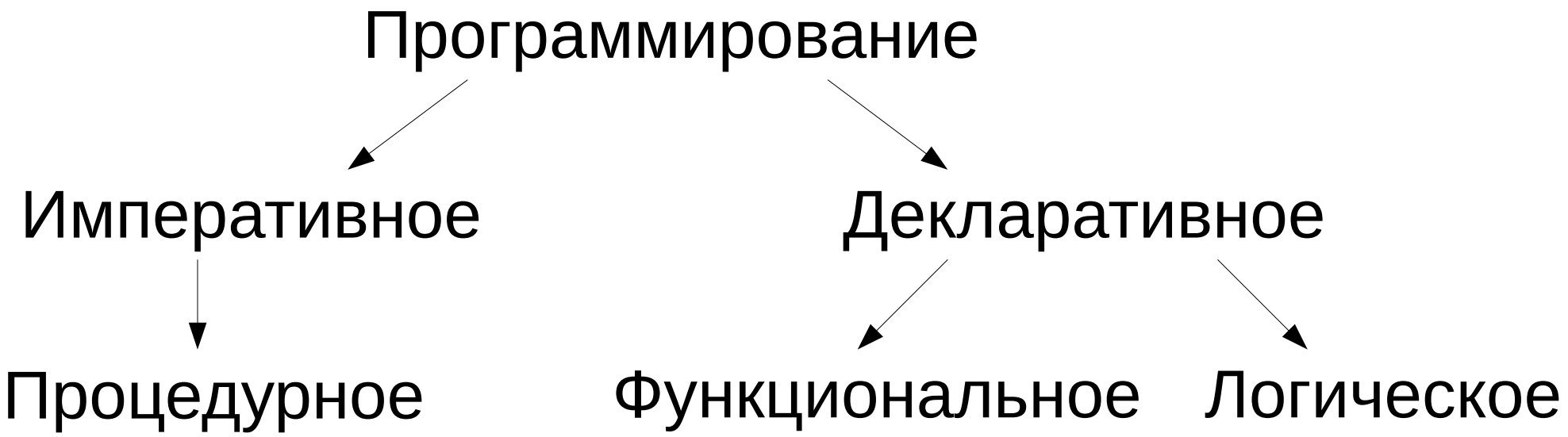
ФУНКЦИОНАЛЬНОЕ И ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Кузнецов Денис Борисович
8-905-86-240-38
kdenisb@mail.ru

Виды программирования



Современное применение

- Облачные вычисления
- Задачи искусственного интеллекта
- Эрланг
- SQL
- XML (xslt)

Математические аспекты

- Рекурсивные функции
- Функции высших порядков
- λ-исчисление
- Комбинаторы
- Монады
- Ленивые вычисления
- Предикаты
- Метод резолюций

Литература

- Хендерсон «Функциональное программирование»
- Сборник статей «Математическая логика в программировании»
- Тей «Логический подход к искусственному интеллекту»
- Братко «Программирование на языке Пролог для искусственного интеллекта»

Пример функциональной программы

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1;$$

$$fact(n) = \begin{cases} 1, n = 0; \\ n \cdot \ fact(n-1), n > 0. \end{cases}$$

```
int fact (int n)
{
int x = 1;
while (n > 0)
{
x = x * n;
n = n - 1;
}
return (x);
}
```

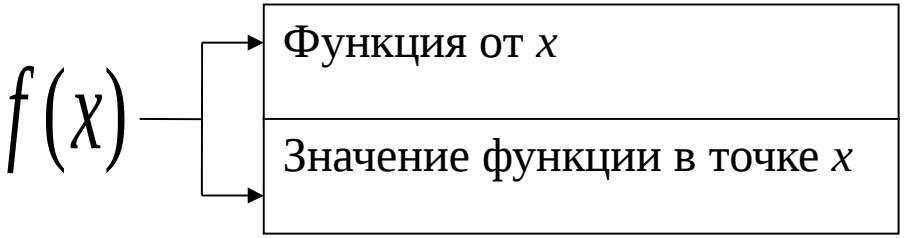
```
fact1 0 = 1
fact1 n = n * fact1 (n-1)

fact2 n | n == 0 = 1
        | n > 0 = n * fact2 (n-1)
```

Сравнение функционального и процедурного программирования

Процедурное	Функциональное
1) переменные	
Именованные ячейки памяти, изменяемые константы.	Настоящие переменные (в частности не используется оператор присваивания).
2) циклы	
Циклы есть.	Не может быть циклов (нет счетчика); используются рекурсивные функции вместо циклов.
3) программа – это …	
Последовательное изменение памяти.	Вычисление функции. Последовательное выполнение команд бессмысленно, поскольку одна команда не может повлиять на выполнение следующей (т.к. негде сохранять промежуточные значения).
4) функции	
При вызове функции возможны побочные эффекты (два вызова одной о той же функции с одними и теми же аргументами могут дать разные результаты; изменение значений глобальных переменных).	Функции используются широко, могут даже использоваться в качестве аргументов для других функций и возвращать в качестве результата; побочных эффектов нет.

Лямбда - исчисление

$$f(x)=x^2$$



$$\lambda x.x^2 \qquad \lambda x.f(x)$$

Язык лямбда-исчисления

λ - абстракция
x, y, z - символы переменных
a, b, c - символы констант
• - аппликация M•N (применение аргумента к функции); M(N) - математическая запись аппликации. Символ • иногда опускают, т.е. MN.

λ -терм:
1) переменная или константа;
2) λx.M, где x – переменная, M - λ-терм;
3) M•N, если M и N - -термы

Правила вывода:
Если M=N, то N=M (симметричность);
Если M=N, N=P, то M=P (транзитивность);
Если M=N, то MP=NP;
Если M=N, то λx.M= λx.N;

Конверсии

α - конверсия

$$\lambda x.M = (\lambda y.M)^{x}_{y}$$

β - редукция

$$(\lambda x.M)N = M^{x}_{N}$$

Примеры β-редукций

- $(\lambda x.x+8)(9) \rhd_\beta$
- $(\lambda x.x+8)(a) \rhd_\beta$
- $(\lambda z.x+y+z+t)(a) \rhd_\beta$
- $(\lambda x.((\lambda y.xy)u))(\lambda v.v) \rhd_\beta$
- $(\lambda x.xx)(\lambda x.xx) \rhd_\beta$

β-нормальная форма

Редукция – “уменьшение” (в большинстве случаев терм укорачивается)

Редекс — форма терма, в котором выполнены все редукции

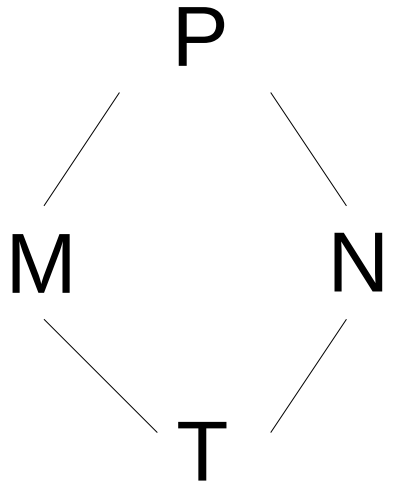
β-нормальная форма — форма без редексов

Теорема Чёрча-Россера

Не для каждого терма существует β-нормальная форма.

β-нормальная форма, если она существует, то единственна.

$$\frac{P \underset{\beta}{\rhd} M, \quad P \underset{\beta}{\rhd} N}{\exists T : M \underset{\beta}{\rhd} T, \; N \underset{\beta}{\rhd} T}$$



Каррирование

Операция **каррирования** позволяет записать функции нескольких аргументов в обычной λ-нотации.

$$(\lambda x.(\lambda y.x+y)) — (\lambda xy.x+y)$$

«Синтаксический сахар» применяется и в других сферах программирования

Свободные и связанные переменные

Примеры из математики:

$$\sum_{i=1}^{n} i \qquad \int_{0}^{x} \sin y dy$$

Переменные i и y являются связанными.

Переменные в λ-выражениях могут быть *свободными* и *связанными*.

(λx.x+y)

x – связанная переменная, *y* – свободная переменная

Следует понимать, что в каком-либо подвыражении переменная может быть свободной (как в выражении под интегралом), однако во всём выражении она связана какой-либо *операцией связывания переменной*, такой как операция суммирования. Та часть выражения, которая находится “внутри” операции связывания, называется *областью видимости* переменной.

Любое вхождение переменной **x** в терм **λx.P** называется связанным (посредством **λx**).

Комбинаторы

Терм без свободных вхождений переменных, называется замкнутым или **комбинатором**. Замкнутый терм имеет фиксированный смысл независимо от значения любых переменных.

$I = \lambda x.x$	тождественный комбинатор	$IX \underset{\beta}{\triangleright} X$
$B = \lambda xyz.x(yz)$	композиция	$BFGX \underset{\beta}{\triangleright} F(GX)$
$C = \lambda xyz.xzy$	коммутатор	$CFXY \underset{\beta}{\triangleright} FYX$
$K = \lambda xy.x$	образование константы	$KXY \underset{\beta}{\triangleright} X$
$W = \lambda xy.xyy$	дублирование	$WFX \underset{\beta}{\triangleright} FXX$
$S = \lambda xyz.xz(yz)$	подстановка и композиция	$SFGX \underset{\beta}{\triangleright} FX(GX)$
$y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$	комбинатор неподвижной точки (для организации рекурсии)	$Yf \underset{\beta}{\triangleright} f(yf)$

Нумерация Чёрча

n-кратная композиция $x(x(x\ldots(xy)))$

$$x^n y$$

$Z_0 = \overline{0} = \lambda xy.y$ $\quad \overline{0}\ xy \underset{\beta}{\rhd} y$

$Z_n = \overline{n} = \lambda xy.x^n y$ $\quad \overline{n}\ xy \underset{\beta}{\rhd} x^n y$

Комбинатор прибавления единицы $\overline{\sigma} = \lambda uxy.x(uxy)$

$\overline{\sigma}\ \overline{n} \underset{\beta}{\rhd} \overline{n+1}$ $\quad (\lambda uxy.x(uxy))(\lambda xy.x^n y)$

Комбинатор упорядоченной пары (проверка на ноль)

$$D = \lambda xyz.z(Ky)x$$

$$DXY\ \overline{0}\ \underset{\beta}{\square}\ X$$

$$DXY(\overline{n+1})\ \underset{\beta}{\rhd} Y$$

Комбинатор примитивной рекурсии

$$R = \lambda xyu.u(Qy)(D\ \overline{0}\ x)1$$

$$Q = \lambda yv.D(\overline{\sigma}(v\ \overline{0}))(y(v\ \overline{0})(v1))$$

$$RGX\ \overline{0} \underset{\beta}{\rhd} X$$

$$RGX\ (\overline{n+1}) \underset{\beta}{\rhd} G\ \overline{n}(RGX\ \overline{n})$$

Примеры комбинаторов

Выполнить бета-редукцию с комбинаторами и лямбда-термами

w k 1

s k k 8

b f1 f1 4

c b f1 f1 4

s f2 f1 5

Шпаргалка

i x = x
b x y z = x (y z)
c x y z = x z y
k x y = x
w x y = x y y
s x y z = x z (y z)
f1 x = x + 1
f2 x y = x + y

S-выражения

S-выражения — форма символьных данных, используемая в lisp

Примеры S-выражений:

(Высшая школа экомики)

((Амкар Рубин) (3 2) (1 4))

((Мой дом) имеет (большие (светлые окна)))

Скобки + элементы (атомы)

Элементы S-выражений

Определить число элементов

(Джон Смит дуб)
(14 градусов 15 минут)
(банан)
(б а н а н)

((Вася химия 4) (Петя литература 5))

S-выражением являются:
1. атом
2. последовательность S-выражений, заключенная в скобки

S-выражения для алгебраических формул

p + q (плюс p q)
p - q (минус p q)
p * q (умн p q)

2 + x * (y - 1)

Оперирование списками (S-выражениями)

car — первый **ЭЛЕМЕНТ** списка

cdr — **СПИСОК** без первого элемента

x - (A B C D)

car(x) = A
cdr(x) = (B C D)
cdr(cdr(x)) = (C D)
car(cdr(cdr(x))) =

Создание списка

cons(ЭЛЕМЕНТ,СПИСОК)

Cons(1, (7 8)) = (1 7 8)

Создание списка

x	y	cons(x,y)
A	(B C)	(A B C)
(A)	(B C)	
(A B)	((C D))	
A	NIL	
A	B	

Создание списка

x	y	cons(x,y)
A	(B C)	(A B C)
(A)	(B C)	((A) B C)
(A B)	((C D))	((A B) (C D))
A	NIL	(A)
A	B	(A . B)

Точечная запись списка

cons(A,B) = (A . B)

(A . NIL) =
(A . (B . C)) =
(A . (B C)) =
(B . (C)) =
(A . (B . (C . (D . (E . (F . NIL)))))) =

Примеры списков

- x=((A B) C (D E))
  F1(x) = C
  F2(x) = (B D)
  F1,F2 - ?

  - **((A B)**
    (C D))
    Первый(x) =
    Второй(x) =
    Верхний левый(x) =
    Правый нижний(x) =
    список2(x,y)=
    квадрат(a,b,c,d)=

Рекурсия в ФП

```
(car (cdr (cdr '(3 5 7))))

(define (last1 x) (car (cdr (cdr x))))

(last1 '(3 5 7))
```

Рекурсия в ФП

```
(define (last2 x)
 (if (null? (cdr x))
  (car x)
  (l (cdr x))
 )
)

(last2 '(3 5 7))
```

Способы организации рекурсии

- Возврат
- Накапливающиеся параметры
- Вспомогательные функции

Рекурсия «возврат»

$$\text{длина}(x) = \begin{cases} 0, & x = nil \\ \text{длина}(cdr(x))+1, & x \neq nil \end{cases}$$

длина(3 5 7) = длина(5 7) + 1 =
длина(7) + 1 + 1 = длина(nil) + 1 + 1 + 1 =
0 + 1 + 1 + 1

Рекурсия «возврат»

$$\text{длина}(x) = \begin{cases} 0, & x = nil \\ \text{длина}(cdr(x))+1, & x \neq nil \end{cases}$$

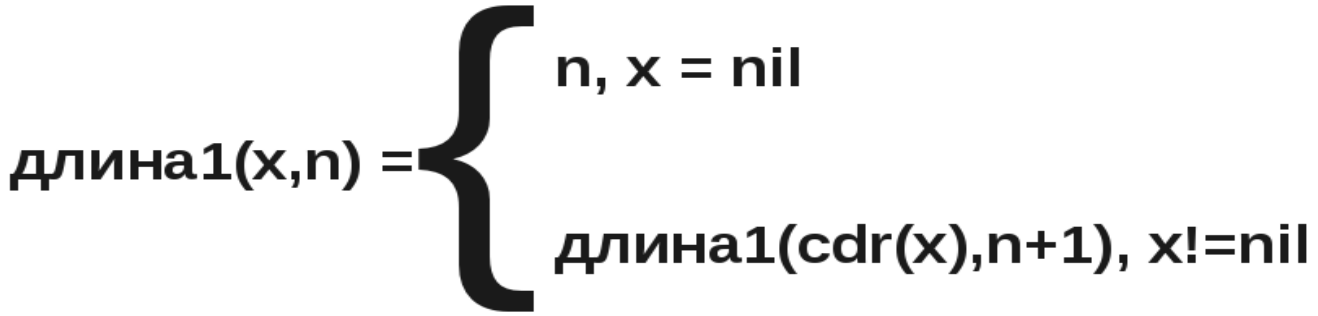
```
(define (dlina x)
 (if (null? x)
  0
  (+ (dlina (cdr x)) 1)
 )
)
```

Рекурсия «накапливающиеся параметры»

$$\text{длина1}(x,n) = \begin{cases} n,\ x = nil \\ \text{длина1}(cdr(x),n+1),\ x!=nil \end{cases}$$

длина1((3 5 7), 0) 

Рекурсия «накапливающиеся параметры»




```
(define (dlina1 x n)
 (if (null? x)
  n
  (dlina1 (cdr x) (+ n 1))
 )
)
(dlina1 '(4 5 2 1) 0)
```

Рекурсия «дополнительная функция»

```
(define (dlina2 x) (dlina1 x 0))
```

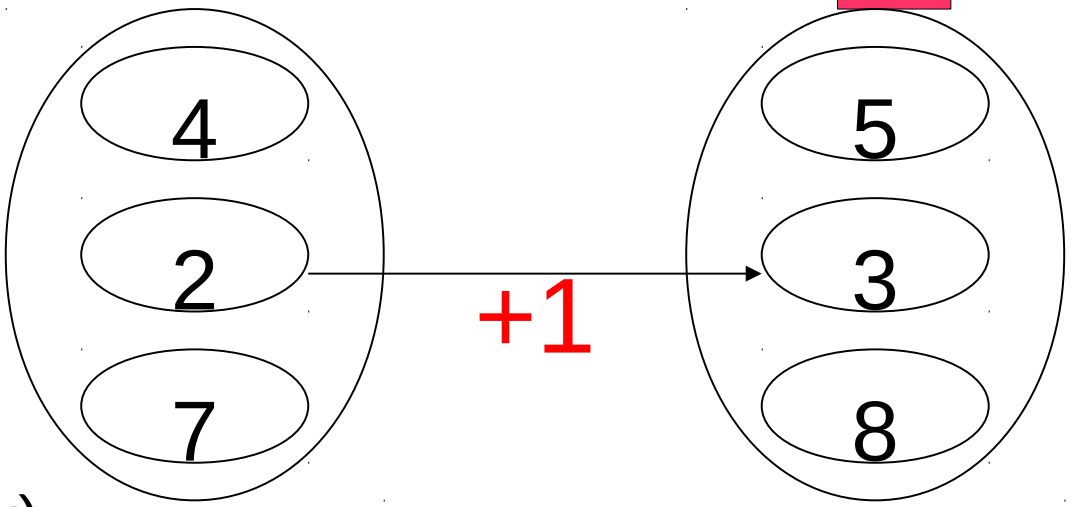
Обращение списка 

Типы функций высших порядков

1. Аргументом функции является функция
   - Отображение (рассмотрено выше)
   - Редукция (результат не список, а значение)
2. Результатом функции является функция

Отображение «+1»

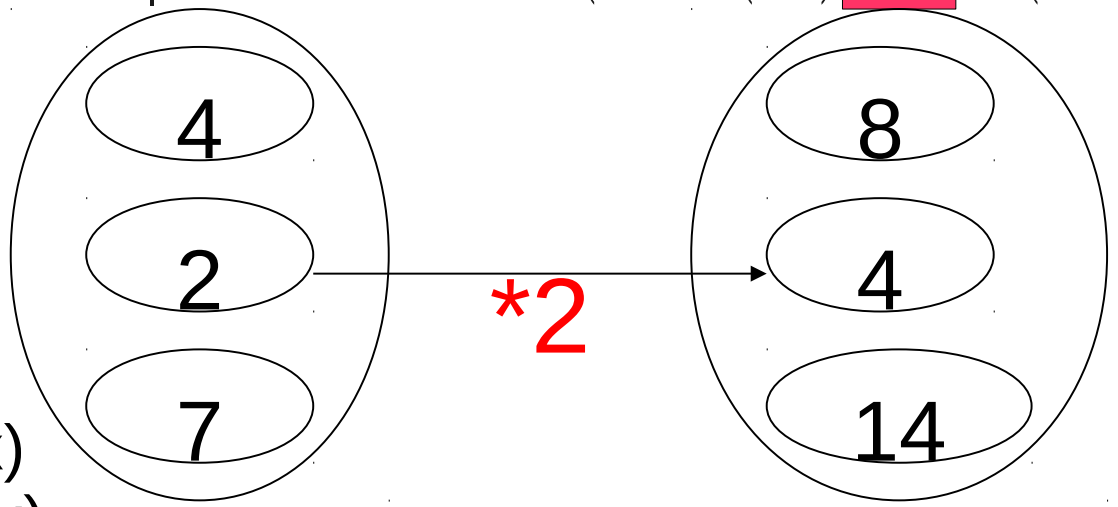
$$S(x) = \begin{cases} x = nil, x \\ x \neq nil, cons(car(x) + 1, S(cdr(x))) \end{cases}$$




```
(define (s x)
    (if (null? x)
        '()
        (cons (+ 1 (car x))
              (s (cdr x))
        )
    )
)
```

Отображение «*2»

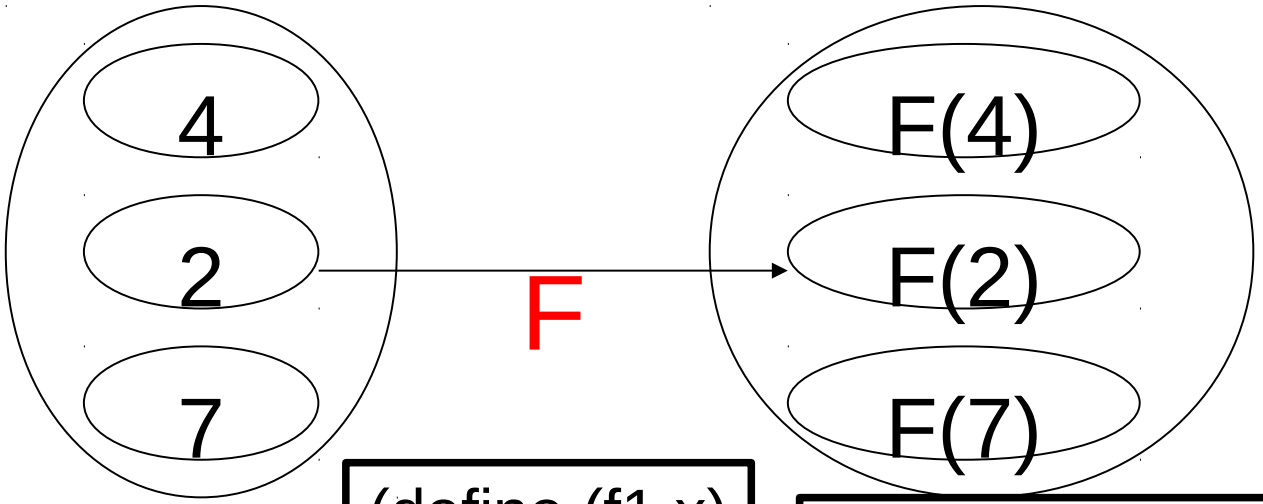
$$S(x)=\begin{cases} x=nil, x \\ x\neq nil, cons(car(x)*2, S(cdr(x))) \end{cases}$$




```
(define (s1 x)
    (if (null? x)
        '()
        (cons (* 2 (car x))
              (s1 (cdr x))
        )
    )
)
```

Общий случай отображения

$$S(x,F)=\begin{cases} x=nil, x \\ x\neq nil, cons(F(car(x)), S(cdr(x),F)) \end{cases}$$




```
(define (s2 x f)
    (if (null? x)
        '()
        (cons (f (car x))
              (s2 (cdr x) f)
        )
    )
)
```

```
(define (f1 x)
    (* x x)
)

(define (f2 x)
    (list x)
)
```

```
> (s2 '(5 3 1) f1)
'(25 9 1)
> (s2 '(5 3 1) f2)
'((5) (3) (1))
> (s2 '(7 2 55) (lambda (x) (+ x 2)))
'(9 4 57)
```

Редукция

```
(define (reduct x f)
    (if (null? x)
        0
        (f (car x) (reduct (cdr x) f))

    )
)
```

```
(define (f3 x y) (+ x y))
(define (f4 x y) (+ x y 3))
(define (f5 x y) (* x y))
```

```
> (reduct '(7 3 5) f3)
15
> (reduct '(7 3 5) f4)
24
> (reduct '(7 3 5) f5)
0
```

Общий случай?

Результат функции - функция

$$(\lambda n.(\lambda x.x*n))$$

$$(\lambda n.(\lambda x.x*n))\ 5 = (\lambda x.x*5)$$

```
(define (mulN n)
 (lambda (x) (* x n))
)

((mulN 5) 3)
```

Где неточность?

Результат функции - функция

(λfgx.f(g(x)))

```
(define (super f g)
 (lambda (x) (f (g x)))
)

(define (fp2 x) (+ x 2))
(define (fu3 x) (* x 3))

((super fp2 fu3) 5)
((super car cdr) '(8 5 2 4))
```

Haskell

Языки, которые основываются на комбинаторной логике:

- ML (Milner, 1984 год)
- Миранда (Turner, 1985 год)

Haskell 1998 год

Дает возможность построения (комбинирования) вычислимых функций из имеющихся

http://learnyouahaskell.com

ТРАНСЛЯТОРЫ: HUGS, GHC

Комбинаторы в Haskell

```
i x = x
b x y z = x (y z)
c x y z = x z y
k x y = x
w x y = x y y
s x y z = x z (y z)
f1 x = x + 1
f2 x y = x + y
```

```
> f2 2 7
> f1 (f1 2)
> b f1 f1 2
> f2 3 3
> w f2 3
> s f2 f1 2
> c s f1 f2 2
```

Структуры данных в Haskell

- Числа
- Символы
- Булевы константы
- Списки
- Кортежи

Простые операции

```
Prelude> 2 + 3
5
Prelude> 3 / 2
1.5
Prelude> div 3  2
1
Prelude> mod 5 3
2
Prelude> 2 + (2 * 2)
6
```

```
Prelude> 2 > 3
False
Prelude> 4 < 5
True
Prelude> 2 == 3
False
Prelude> 2 /= 3
True
Prelude> 'u' == 'w'
False
```

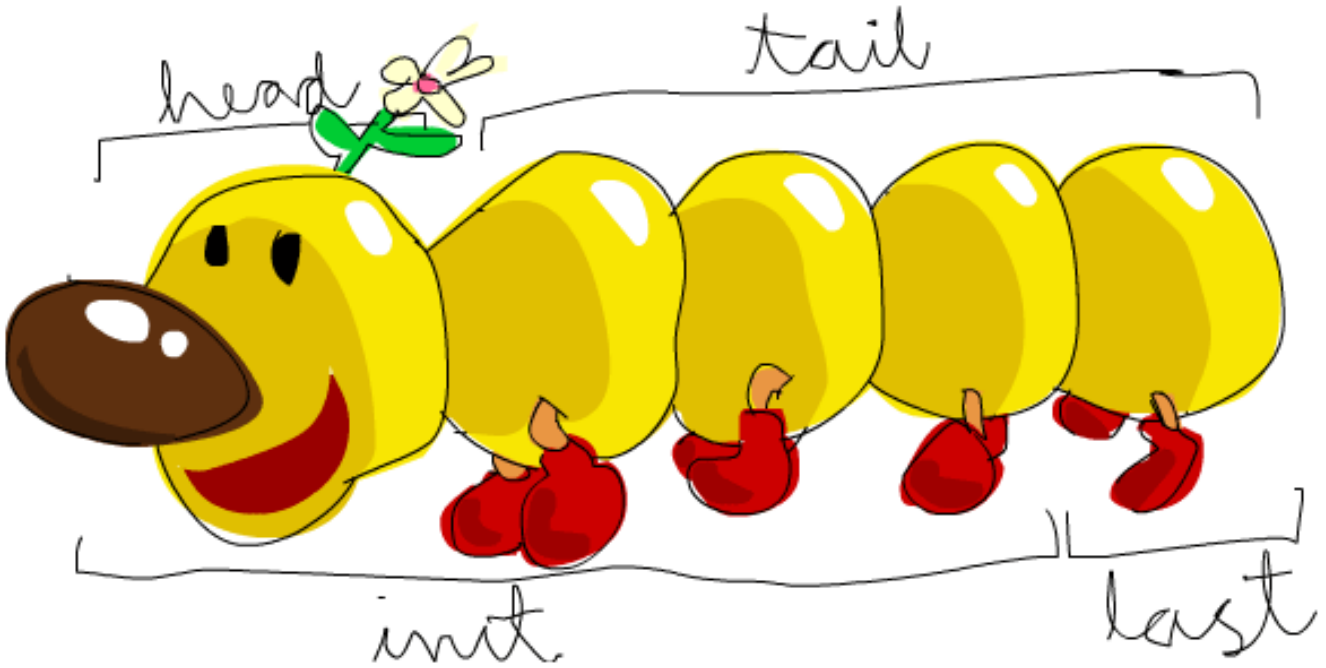
Списки в Haskell

Однородные, состоят из любых данных haskell

```
[1,2,3,4]
['h','e','l','l','o']
"hello"
[[1,2],[3,4]]
[[1,2,3],[4,5]]
```

```
[1,2,'u',4]
[[1],['w']]
[[1,2,3],4,5]
```

Функции для работы со списками в Haskell




```
Prelude> head [1,2,3,4,5]
1
Prelude> tail [1,2,3,4,5]
[2,3,4,5]
Prelude> init [1,2,3,4,5]
[1,2,3,4]
Prelude> last [1,2,3,4,5]
5
```

Операции для работы со списками в Haskell

Операция «cons»

```
Prelude> 7:[1,2,3,4,5]
[7,1,2,3,4,5]
Prelude> 3:1:[1,2,3,4,5]
[3,1,1,2,3,4,5]
```

Операция «cat»

```
Prelude> [3,1] ++ [1,2,3,4,5]
[3,1,1,2,3,4,5]
Prelude> "hel" ++ ['l','o']
"hello"
```

Операции для работы со списками в Haskell

Операция !!

```
Prelude> [1,2,3,4,5] !! 2
3
Prelude> "hello" !! 2
'l'
```

Сравнение

```
Prelude> [3,2,1] > [2,7,8]
True
Prelude> [3,2,1] > [3,2,1,1]
False
```

null
take
drop
minimum
maximum
summ
product

Формирование списков в Haskell

- Перечисление всех элементов
- Заданием диапазона

```
Prelude> [1,4..20]
[1,4,7,10,13,16,19]
Prelude> [0,4..20]
[0,4,8,12,16,20]
Prelude> ['a'..'z']
"abcdefghijklmnopqrstuvwxyz"
```

- Аналитически

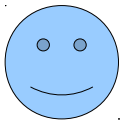
Аналитическое формирование списков в Haskell

```
Prelude> [ 2*x | x <- [3..4] ]
[6,8]
Prelude> [ 2*x | x <- [3,5,4] ]
[6,10,8]
Prelude> [ x | x <- [50..100], x `mod` 7 == 3]
[52,59,66,73,80,87,94]
Prelude> let boomBangs xs = [ if x < 10 then "BOOM!"
                                        else "BANG!" | x <- xs, odd x]
Prelude> boomBangs [2..11]
["BOOM!","BOOM!","BOOM!","BOOM!","BANG!"]
```

Кортежи в Haskell

Элементы могут быть разнородными.
Важен порядок. Число элементов ограничено

```
Prelude> (1,2,"tri",[4,5])
(1,2,"tri",[4,5])
Prelude> [ (a,b,c) | c <- [1..10], b <- [1..c],
a <- [1..b], a^2 + b^2 == c^2 ]
[(3,4,5),(6,8,10)]
Prelude> (1,2,3) > (1,3,2)
False
Prelude> (1,2,3) > (1,3,2,1)
```



Декартово произведение

Управляющие конструкции Haskell

```
case (x1, . . ., xk) of
  (p11, . . ., p1k) -> e1
  . . .
  (pn1, . . . , pnk) -> en
if e1 then e2 else e3
```

```
Prelude> if 2>3 then 2 else 3
3
Prelude> case 2 of 2 -> 2
2
Prelude> case 2>3 of False -> 2
2
```

λ-выражения в Haskell

```
Prelude> (\ x y -> x + y) 2 3
5
```

Типизированное лямбда-исчисление

Структура терма, представляющего функцию, должна нести информацию об области определения и области значений

Функциональные типы

$y=f(x)$ $F:X\to Y$

$(a\to b)$

Если a и b типы,
то $(a\to b)$ тоже тип

Множество функций,
которые определены на a
и принимают значения из b

$(a\to(b\to(c\to d))) = (a\to b\to c\to d)$

Примеры типов

f(x)=x+1	(int→int)
f(x,y)=x+y	(int→(int→int))
f(x,y)=x^y	(float→(int→float))
p(x)=x>0	(int→bool)
p(x,y)=x>y	

Типизированные лямбда-термы

- Каждая типизированная переменная x^a,y^a, x^b,y^b... является типизированным λ-термом
- Если $M^{(a\to b)}$ и N^a — типизированные λ-термы, то $(M^{(a\to b)}N^a)^b$ — также типизированный λ-терм
- Если x^a — типизированная переменная, а M^b — типизированный λ-терм, то $(\lambda x^a.M^b)^{(a\to b)}$ — также типизированный λ-терм

Примеры типизированных лямбда-термов

$$\mathbf{I}_\alpha \equiv (\lambda x^\alpha . x^\alpha)^{(\alpha \to \alpha)};$$

$$\frac{x^\alpha}{(\lambda x^\alpha . x^\alpha)^{(\alpha \to \alpha)}} .$$

$$\mathbf{K}_{\alpha,\beta} \equiv (\lambda x^\alpha\ y^\beta . x^\alpha)^{\alpha \to \beta \to \alpha};$$

$$\frac{\dfrac{x^\alpha}{(\lambda y^\beta . x^\alpha)^{\beta \to \alpha}}}{(\lambda x^\alpha\ y^\beta . x^\alpha)^{\alpha \to \beta \to \alpha}} .$$

$$\mathbf{S}_{\alpha,\beta,\gamma} \equiv (\lambda x^{\alpha \to \beta \to \gamma} y^{\alpha \to \beta} z^\alpha . xz(yz))^{(\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma}$$

$$\frac{\dfrac{\dfrac{\dfrac{x^{\alpha \to \beta \to \gamma} z^\alpha}{(xz)^{\beta \to \gamma}} \quad \dfrac{y^{\alpha \to \beta} z^\alpha}{(yz)^\beta}}{((xz)\,(yz))^\gamma}}{(\lambda z . xz(yz))^{\alpha \to \gamma}}}{(\lambda yz . xz(yz))^{(\alpha \to \beta) \to \alpha \to \gamma}}}{(\lambda xyz . xz(yz))^{(\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma}} .$$

Примеры типизированных лямбда-термов

$$\mathbf{B}_{\alpha,\beta,\gamma} \equiv (\lambda x^{\alpha\to\beta} y^{\gamma\to\alpha} z^{\gamma} . x(yz))$$

$$\mathbf{C}_{\alpha,\beta,\gamma} \equiv (\lambda x^{\alpha\to\beta\to\gamma} y^{\beta} z^{\alpha} . xzv)$$

$$\mathbf{W}_{\alpha,\beta} \equiv (\lambda x^{\alpha\to\alpha\to\beta} y^{\alpha} . xyy)$$

$$(\lambda x.xx)(\lambda x.xx)$$

Примеры типизированных лямбда-термов

$$\mathbf{B}_{\alpha,\beta,\gamma} \equiv \left(\lambda x^{\alpha\to\beta} y^{\gamma\to\alpha} z^{\gamma} . x(yz)\right)^{(\alpha\to\beta)\to(\gamma\to\alpha)\to\gamma\to\beta};$$

$$\mathbf{C}_{\alpha,\beta,\gamma} \equiv \left(\lambda x^{\alpha\to\beta\to\gamma} y^{\beta} z^{\alpha} . xzy\right)^{(\alpha\to\beta\to\gamma)\to\beta\to\alpha\to\gamma};$$

$$\mathbf{W}_{\alpha,\beta} \equiv \left(\lambda x^{\alpha\to\alpha\to\beta} y^{\alpha} . xyy\right)^{(\alpha\to\alpha\to\beta)\to\alpha\to\beta}.$$

Свойства типизированных лямбда-термов

- В системе типизированных λ-термов все β-редукции конечны
- Каждый типизированных λ-терм имеет β-нормальную форму

Описание прототипов функций в Haskell

```
square :: Double -> Double
square x = x*x
```

Функции с несколькими аргументами в Haskell

Для двух аргументов тип записывается в виде

f :: X -> (Y -> Z)

Неформально, f принимает два аргумента типа X и Y. Применение f к значениям a и b происходит в два этапа:

Если a :: X, то f a :: Y -> Z.

Если b :: Y, то (f a) b :: Z.

Типы и каррирование в Haskell

Например, функцию для вычисления гипотенузы в прямоугольном треугольнике можно сделать каррированной, а можно некаррированной:

```
hyp :: Double -> Double -> Double
hyp x y = sqrt (square x + square y)

hyp' :: (Double, Double) -> Double
hyp' (x,y) = sqrt (square x + square y)
```

Полиморфизм в Haskell

```
f1 :: (Int,Int)->Int
f1 (x,y) = x + y

f2 :: (Int,Char)->[Char]
f2 (x,c)  | x == 0  = []
          | x > 0   = c:f2(x-1,c)

curry1 :: ((Int,Int)->Int)->Int->Int->Int
curry1 f x y = f (x,y)

curry2 :: ((Int,Char)->String)->Int->Char->String
curry2 f x y = f (x,y)

curryn :: ((a, b) -> c) -> a -> b -> c
curryn f x y = f (x,y)
```

Конструкция where

```
filter :: (a -> Bool) -> [a] -> [a]
```

Декларативный стиль

```
filter p [] = []
filter p (x:xs)
   | p x = x : rest
   | otherwise = rest
   where
     rest = filter p xs
```

Стиль выражений (expression)

```
filter =
   \p -> \ xs ->
      case xs of
         [] -> []
         (x:xs) ->
            let rest = filter p xs
            in  if p x
                  then x : rest
                  else rest
```

Ленивые и жадные вычисления

Ленивые (отложенные, lazy)

Что из того, что требуется для продуцирования результатов, может быть и, следовательно, должно быть вычислено?

Жадные

Что может быть и, следовательно, должно быть вычислено, используя наличествующие сейчас данные?

Жадные вычисления

В связи с мотивом оптимизации сформировался принцип организации вычислительного процесса, диаметрально противоположный принципу ленивых вычислений, — так называемые жадные вычисления. Этот принцип предписывает запускать вычисление всех данных, как только появляется возможность это сделать.

```
for(i=0;i<strlen(s);i++)
{
 <обработка строки s>
}
```



```
n = strlen(s);
for(i=0;i<n;i++)
{
 <обработка s>
}
```

Ленивые вычисления

Принцип организации вычислительного процесса, следуя которому любые запланированные программой действия откладываются до того момента, когда фактически возникает потребность, необходимость этих действий быть выполненными.

if(i<strlen(s) && s[i] != 0)

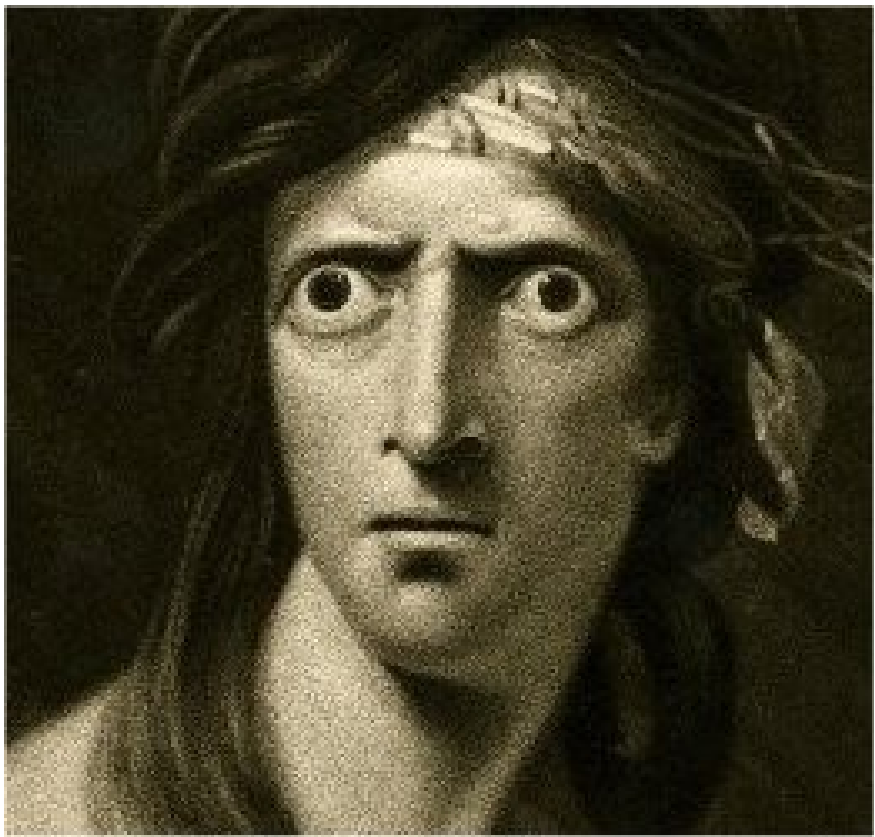
Ленивые вычисления в Haskell

```
ss i = i : i+1 : [i+2]

sss i = i : sss (i + 1)

ssss i = i : ssss j where j = i + 1
```

```
> ss 8
[8,9,10]
> take 3 [5,6,7,8,9]
[5,6,7]
> take 2 (ss 8)
[8,9]
> take 2 (sss 8)
[8,9]
> take 20 (sss 8)
[8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27]
```

Монады

Монада — это просто моноид в категории эндофункторов



Группы

Полугруппа — упорядоченная пара (S,*)
S — непустое множество объектов
* — ассоциативная бинарная операция над объектами s1*(s2*s3)=(s1*s2)*s3
*::S->S->S

Идемпотент e ∈ S, если e * e = e

Единица e*s=s*e=s

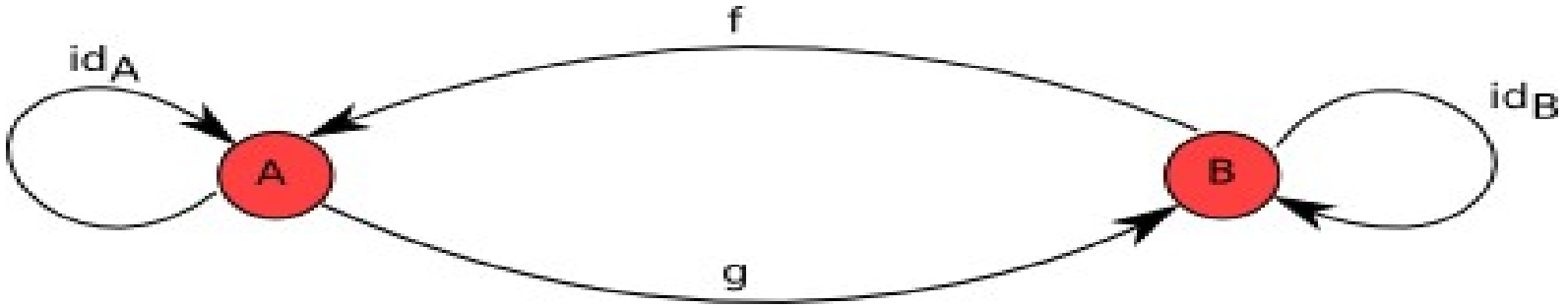
Ноль e*s=s*e= 

Моноид — полугруппа с единицей

Группа — моноид с обратным элементом

Абелева группа: s1*s2=s2*s1

Категории



Для любых двух объектов A и B определён класс морфизмов. Для любых двух морфизмов f: A -> B и g: B -> C определена их композиция – морфизм h = g ∘ f, h : A -> C.

Композиция ассоциативна: (h∘g) ∘ f = h ∘ (g ∘ f)

Для любого объекта A имеется «единичный» морфизм: idB ∘ f = f = f ∘ idA.

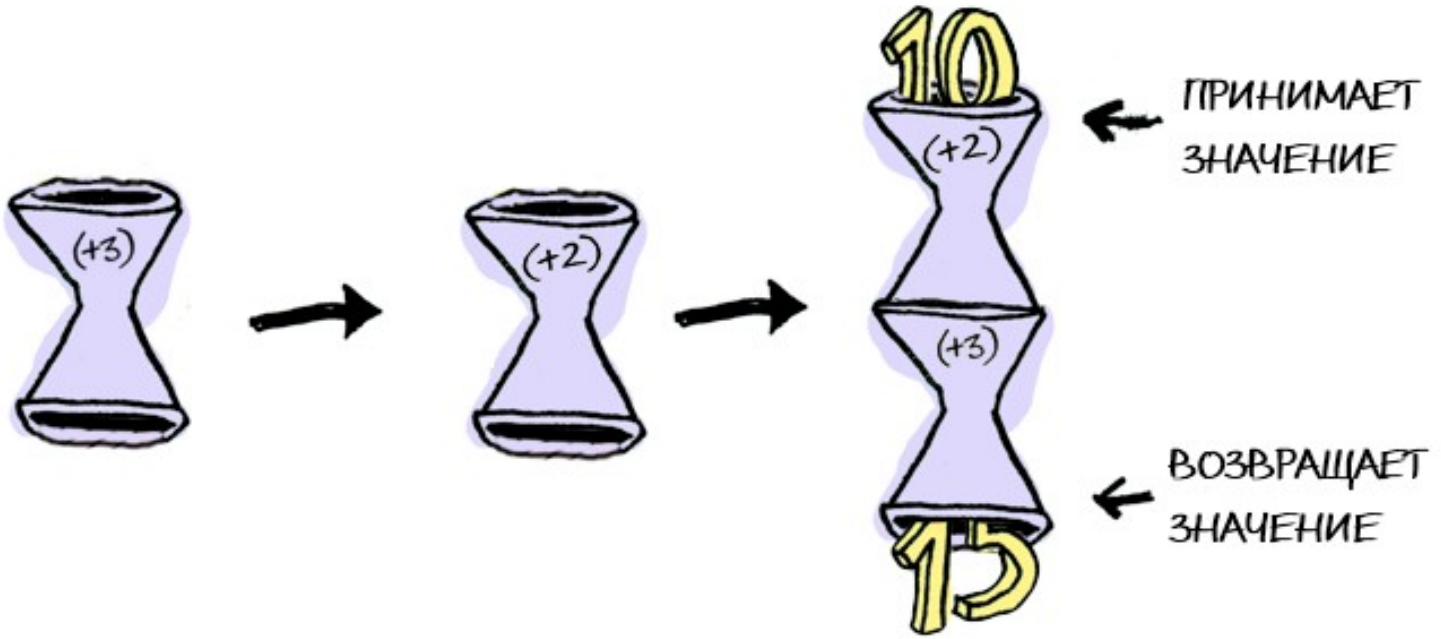
Функторы

Функтор F из категории C в категорию D является функцией, которая отображает каждый объект a категории C в некоторый объект F(a) категории D,

Функторы

а каждый морфизм g в категории C из объекта a в объект b в некоторый морфизм F(g) в категории D из объекта F(a) в объект F(b).

g::a->b --->> F(g)::F(a)->F(b)



Эндофунктор — это функтор категории в саму себя.

Монады в Haskell

- IO — последовательные вычисления
- Maybe — вычисления с обработкой отсутствующих значений
- List — вычисления с несколькими результатами
- Пользовательские

Монада IO

```
main :: IO ()
main = do
  line <- getLine
  if line /= "quit"
    then putStrLn line >> main
    else return ()

main2 :: IO ()
main2 = do
  let x = 10
  line <- getLine
  let y = read line :: Int
  let z = x + y
  if line /= "666"
    then print z >> main2
    else return ()
```

Монада Maybe

```
import Control.Monad

f::Int -> Maybe Int
f 0 = Nothing
f x = Just x

isqrt :: Integer -> Maybe Integer
isqrt x = isqrt' x (0,0)
  where
    isqrt' x (s,r)
      | s > x     = Nothing
      | s == x    = Just r
      | otherwise = isqrt' x (s + 2*r + 1, r+1)

i4throot :: Integer -> Maybe Integer
i4throot x = case isqrt x of
               Nothing -> Nothing
               Just y  -> isqrt y
```

Алгорифмы Маркова

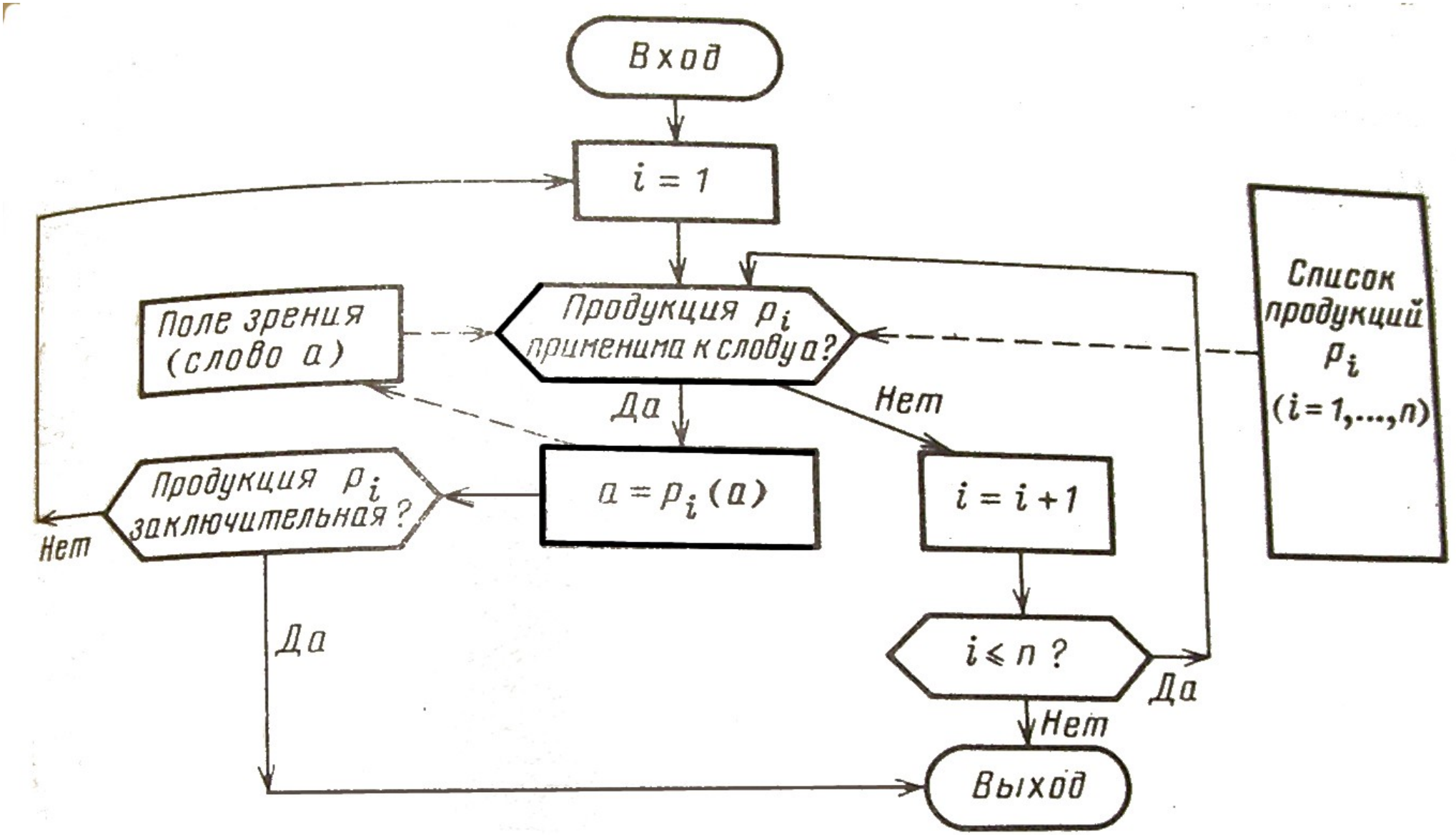
А. Марков, вторая половина 1940х.

Не фон-неймоновский способ записи алгоритмов.

Как и машина Тьюринга работает со всеми алгоритмически-разрешимыми задачами.

Не составление последовательности действий для выполнения функции, а рекурсивное описание.

Блок-схема алгорифма Маркова



Примеры продукций

ПРИВЕТ — ПОКА

ОВЕ → ОКА
РИ → О

ЗАМЕНА

a → b

СЛОЖЕНИЕ УНАР. Ч.

\+ →
| · →

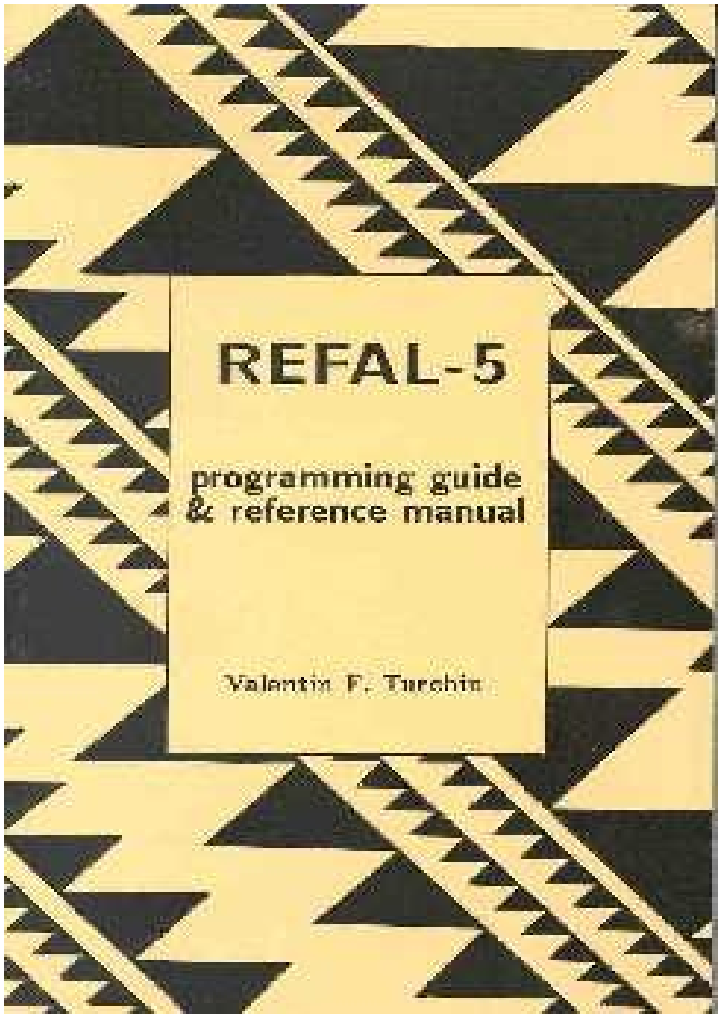
СОРТИРОВКА

ba → ab

Рефал

REcursive **F**unctions **A**lgorithmic **L**anguage
Programming language

By Prof. Valentin F. Turchin
The City College of New York



Структура программы Рефал

программа:
функция
функция
……

функция F:
F {
продукция;
продукция;
……
}

продукция:
O = Q

O: образец (P-выражение без <F P>)
Q: P-выражение

P-выражение:
1. строка в кавычках
2. переменная-символ s.i
3. переменная-слово e.i
4. вызов функции <F P>
5. P P

Программа Рефал замены a на b

```
$ENTRY Go { = <Prout <Pal 'abbabbba'>> }

Pal {
    'a' e.1 = 'b' <Pal e.1>;
    s.1 e.2 = s.1 <Pal e.2>;
    'a' = 'b';
    =;
  }
```

```
$ ./refc ab
Refal-5 Compiler. Version PZ Nov 20 2013
Copyright: Refal Systems Inc.
$ ./refgo ab
bbbbbbbb
```

Программа Рефал удаление комментариев

```
$ENTRY Go { = <Prout <Pal 'code one
/* comment one */ code two /* comment two */
d'>> }

Pal {
    e.1 '/*' e.2 '*/' e.3 = e.1 <Pal e.3>;
    e.1 = e.1 ;
   }
```

```
$ ./refc coms
Refal-5 Compiler. Version PZ Nov 20 2013
Copyright: Refal Systems Inc.
$ ./refgo coms
code one  code two  d
bbbbbbbb
```

Программа Рефал палиндром

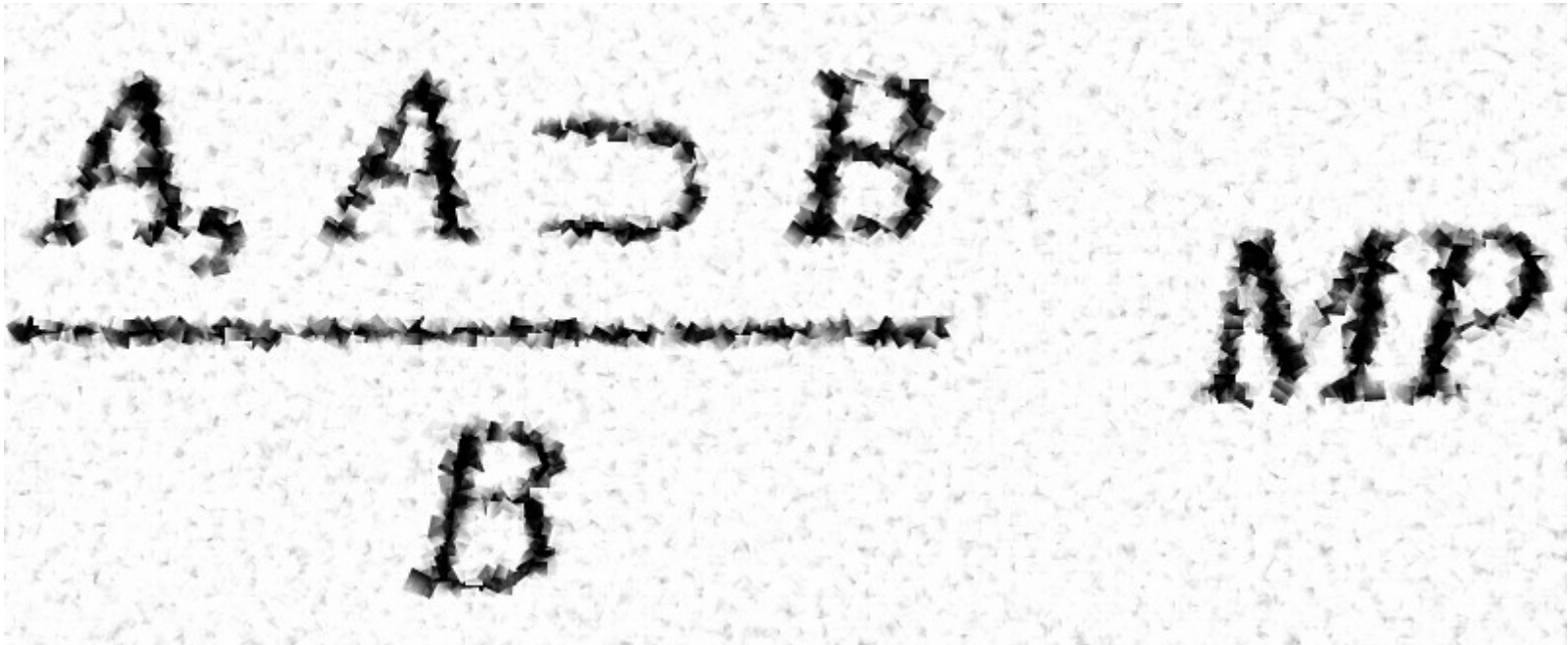
```
$ENTRY Go { = <Prout <Pal 'revolver'>> }

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

```
$ ./refgo pal
False
```

Метод резолюций

Аксиоматическая теория первого порядка, которая использует доказательство от противного.

Теорию изобрел Дж.Робинсон в 1965 году. В теории используется язык **предикатов**.



Предикаты

Предикат — логическая функция, аргументы которой могут принимать значения из некоторой предметной области, а сама функция принимать значения **истина** или **ложь**.

Предикаты бывают одноместные P(x), двухместные P(x,y) и т.д.

Ноль-местный предикат – это высказывание (его значение не зависит от аргумента).

x>0 — предикат

5<0 — высказывание

Алгоритм метода резолюций

1. Формализация задачи, получаем некоторые записи в форме предикатов.

2. Получение предваренной нормальной форме.

3. Сколемизация.

4. Получении дизъюнктов.

5. Добавлении в систему отрицания выводимой формулы.

6. Выполнении резолюций:

- Унификация (приведение предикатов к единой форме),
- Получение резольвенты.

Формализация задачи

А1) Кто ходит в гости по утрам, тот поступает мудро.

А2) У кого есть воздушный шарик, тот ходит в гости по утрам.

А3) Воздушный шарик есть у Пятачка.

Г(х) – х ходит в гости по утрам

М(х) – х поступает мудро

Ш(х) – х владеет воздушным шариком

A1 $\forall x Г(x) \rightarrow М(x)$

A2 $\forall x Ш(x) \rightarrow Г(x)$

A3 $Ш(пятачок)$

B $\exists x М(x)$

ПНФ

Все кванторы должны находиться слева, знаки отрицания должны стоять непосредственно у самих предикатов.

Знаки между предикатами должны быть только & или \/.

$$\forall x P(x) \wedge \exists x R(x) = \forall x P(x) \wedge \exists y R(y) = \forall x \exists y (P(x) \wedge R(y))$$

$$\neg \forall x (\forall y P(x,y) \vee \neg \exists z R(z) \rightarrow Q(x) \wedge \forall y M(x,y)) =$$
$$= \neg \forall x (\neg (\forall y P(x,y) \vee \neg \exists z R(z)) \vee Q(x) \wedge \forall y M(x,y)) =$$
$$= \exists x ((\forall y P(x,y) \vee \neg \exists z R(z)) \wedge (\neg Q(x) \vee \neg \forall y M(x,y))) =$$
$$= \exists x ((\forall y P(x,y) \vee \forall z \neg R(z)) \wedge (\neg Q(x) \vee \exists t \neg M(x,t))) =$$
$$= \exists x \forall y \forall z \exists t ((P(x,y) \vee \neg R(z)) \wedge (\neg Q(x) \vee \neg M(x,t)))$$

A1 $\forall x \neg Г(x) \vee М(x)$

A2 $\forall x \neg Ш(x) \vee Г(x)$

A3 $Ш(пятачок)$

B $\exists x М(x)$

Сколемизация

Отбросить все кванторы:

Кванторы **всеобщности** просто отбрасываются

переменные, которые были связаны с квантором **существования**, заменяются:

• на константу, если левее не было квантора всеобщности,

• иначе на функцию от всех переменных, стоящих слева в кванторе всеобщности.

Примеры сколемизации

$$\exists x \forall y \exists z P(x, y, z) = P(a^c, y, f^c(y));$$

$$\forall x \forall y \exists z P(x, y, z) = P(x, y, f^c(x, y))$$

Получение дизъюнктов

$$P(x) \vee R(x) \wedge G(x) = (P(x) \vee R(x)) \wedge (P(x) \vee G(x))$$

Д1 $(P(x) \vee R(x))$

Д2 $(P(x) \vee G(x))$

Д1 $\neg Г(x) \vee М(x)$

Д2 $\neg Ш(x) \vee Г(x)$

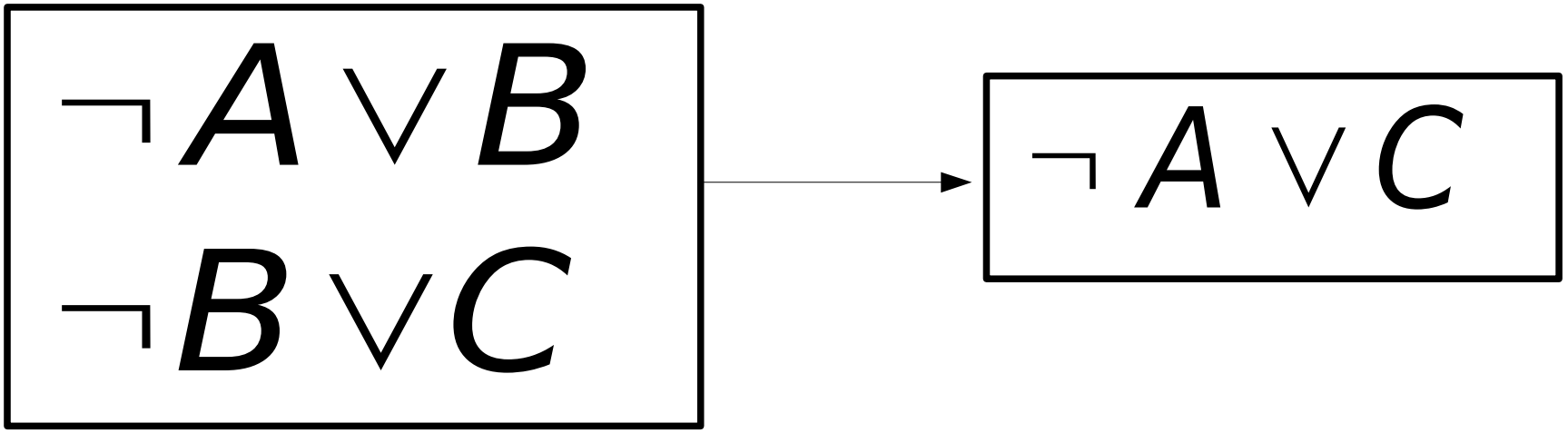
Д3 $Ш(пятачок)$

Д4 $\neg М(x) \vee Отв(x)$

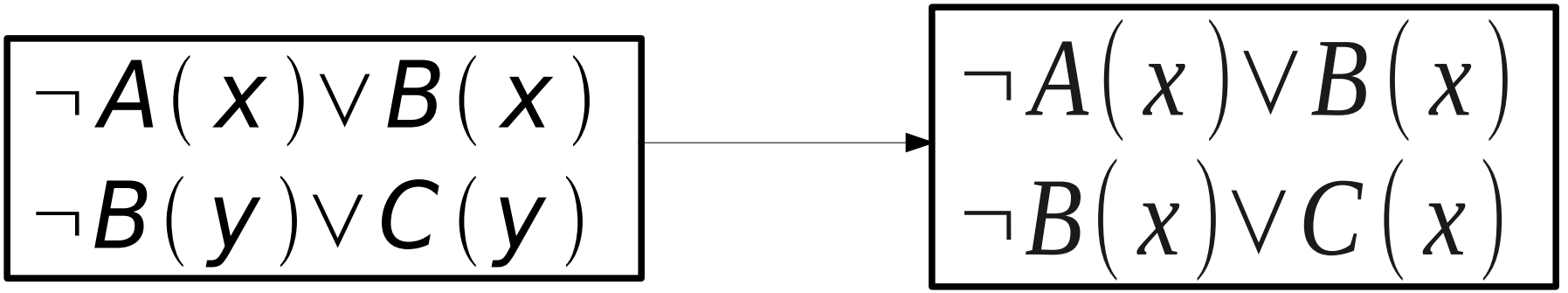
$\neg \exists x M(x) = \forall x \neg M(x)$

$\neg M(x)$

Получение резольвенты



Унификация



$$\neg A(x) \vee C(x)$$

Д1-Д2: Д5 $\neg Ш(x) \vee М(x)$

{x/пятачок}

Д3-Д5: Д6 $М(пятачок)$

Д4-Д6:  v Отв(пятачок)

Хорновская логическая программа

Хорновская логическая программа состоит из фактов и формул.

1) факты имеют вид:

<предикат>(<константы>)

Например: Ш(Пятачок)

2) формулы должны быть вида:

$$A_1 \wedge A_2 \wedge \ldots \wedge A_n \rightarrow A_0$$

$$\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_n \vee A_0$$

Пролог

Programma**log**ique

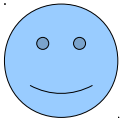
- Автор Алэн Колмероэ (фр.)
- 1972
- GNU Prolog, SWI-Prolog,
- Turbo Prolog, Visual Prolog

Метод резолюций!

Аксиомы в Прологе

Произвольный предикат

$$A_1 \wedge \neg A_2 \vee A_4 \rightarrow A_0$$



Дизъюнкт Хорна

$$A_1 \wedge A_2 \wedge A_4 \rightarrow A_0$$

Пролог

$$A_0 :- A_1, A_2, A_4$$

Структура программы Пролог

Факты

<предикат>(<термы>).

Формулы

<предикат>(<термы>):−<предикаты>.

Вопрос (запрос)

?- <предикат>(<термы>).

Пример программы на Прологе

$$\forall x Г(x) \rightarrow М(x)$$

$$\forall x Ш(x) \rightarrow Г(x)$$

$$Ш(пятачок)$$

$$\exists x М(x)$$

```
m(X) :- g(X).
g(X) :- s(X).
s('Pyatak').
?- m(X).
```

Недостатки Пролога

- Ответ на вопрос может быть не найден (ловушка бесконечной ветви)
- Добавлены примитивы, управляющие стратегией обхода дерева (оператор отсечения)
- Проблемы использования памяти и быстродействия требуют отдельного решения

GNU Prolog

Особенности:

- ISO Prolog
- Программирование в ограничениях (constraint programming)
- Нативный код и классический интерпретатор
- Отладчик
- Warren Abstract Machine

Использование gprolog

1. Программа с фактами и формулами оформляются в отдельном файле
2. Из интерпретатора файл программы подключается предикатом *consult*
3. В интерпретаторе задаются вопросы

```
$ gprolog
| ?- consult(shar).
yes
| ?- m(X).
X = 'Pyatak'
yes
| ?- g('Pyatak').
yes
```

Обработка списков на Прологе

Типы элементов могут быть разные

```
| ?- X=[2,3,4].
X = [2,3,4]
yes
| ?- X=[2,3,4,'a'].
X = [2,3,4,a]
yes
| ?- X=[2,3,4,['a',3]].
X = [2,3,4,[a,3]]
yes
```

Структура списка в Прологе

Голова отделяется от хвоста символом |

```
golova([X|_],X).

hvost([_|X],X).
```

```
| ?- golova([2,3,4,'a'],X).
X = 2
yes

| ?- hvost([2,3,4,'a'],X).
X = [3,4,a]
yes
```

Пример обработки списков на Прологе

```
app([],L,L).
app([X|L1], L2, [X|L3]) :-app(L1,L2,L3).
```

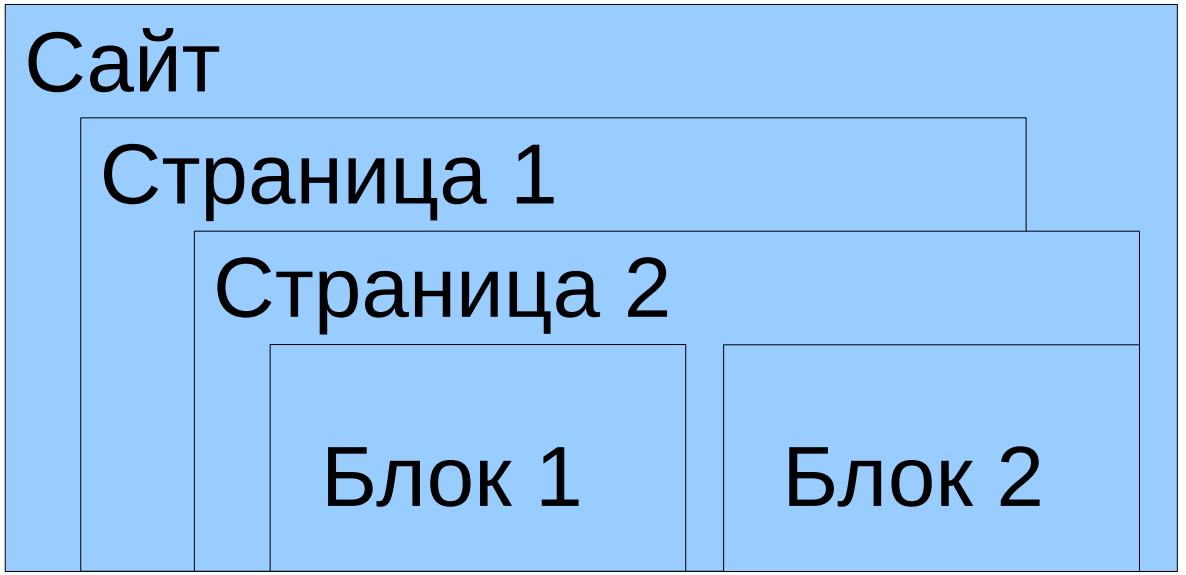
```
| ?- app([1,2],[6,4],X).
X = [1,2,6,4]
yes
| ?- app([1,2],X,[1,2,8,7]).
X = [8,7]
yes
| ?- app([1,2],X,[2,8,7]).
no
```

Базы данных на Прологе

Для организации древовидных БД в Прологе Используются функторы.

Функтор в Прологе - не то же самое, что функция и других языках программирования; это просто имя, которое определяет вид составного объекта данных и объединяет вместе его аргументы.

Пример Функтора




```
site('s1',pg('/main/',[bl('news','str'),bl('cnt','txt')]),
    [ pg('/news/',[bl('news','str')]),
      pg('/about/',[bl('hist','txt'),bl('sch','txt')])
    ]).
site('s2',pg('/about/',[bl('hist','str'),bl('cnt','txt')]),
    [ pg('/news/',[bl('news','str')])
    ]).
```

Запросы к БД Пролога

```
mainp(X,S):-site(S,X,_).
innerp(X,S):-site(S,_,Y),inthe(X,Y).

inthe( X, [X | L ]).
inthe( X, [Y | L ]):-inthe( X, L).

existp(X,S):-mainp(X,S).
existp(X,S):-innerp(X,S).
```

Формальные грамматики на Прологе

```
str(X):-es([],X2),fields(X2,X3),ef(X3,X).
es(X,Y):-app(X,['s'],Y).
ef(X,Y):-app(X,['f'],Y).
fields(X,Y):-iks(X,X1),igr(X1,Y).
igr(X,Y):-app(X,[],Y).
igr(X,Y):-iks(X,X1),igr(X1,Y).

iks(X,Y):-app(X,['x'],Y).

app([],L,L).
app([X|L1], L2, [X|L3]) :- app(L1,L2,L3).
```

$$S \rightarrow s\ F\ f$$
$$F \rightarrow x\ Y$$
$$Y \rightarrow x\ Y$$
$$Y \rightarrow \varepsilon$$

Вариант грамматики с отсечениями на Прологе

```
str(X):-
es(X1),iks(X2),app(X1,X2,L1),!,ef(X3),app(L1,
X3,L2),!,iks(X4),app(L2,X4,X).
es( ['s'] ).
iksx( ['x'] ).
ef( ['f'] ).
iks(X):-iksx(X1),igr(X2),app(X1,X2,X).
igr([]).
igr(X):-iksx(X1),igr(X2),app(X1,X2,X).
app([],L,L).
app([X|L1], L2, [X|L3]) :- app(L1,L2,L3).
```

Программирование с ограничениями

$$\begin{array}{r} \texttt{SEND} \\ + \texttt{MORE} \\ \hline \texttt{MONEY} \end{array}$$

Программирование с ограничениями

$$\begin{array}{r} \texttt{SEND} \\ + \texttt{MORE} \\ \hline \texttt{MONEY} \end{array}$$

1000*S + 100*E + 10*N + D +
1000*M + 100*O + 10*R + E
= 10000*M + 1000*O + 100*N + 10*E + Y

constrain prolog

```
1000*S + 100*E + 10*N + D +
1000*M + 100*O + 10*R + E
= 10000*M + 1000*O + 100*N + 10*E + Y
```

```
send(LD):-
        LD=[S,E,N,D,M,O,R,Y],
        fd_all_different(LD),
        fd_domain(LD,0,9),
        fd_domain([S,M],1,9),

        1000*S + 100*E + 10*N + D +
        1000*M + 100*O + 10*R + E
        #= 10000*M + 1000*O + 100*N + 10*E + Y,

        fd_labelingff(LD).
```