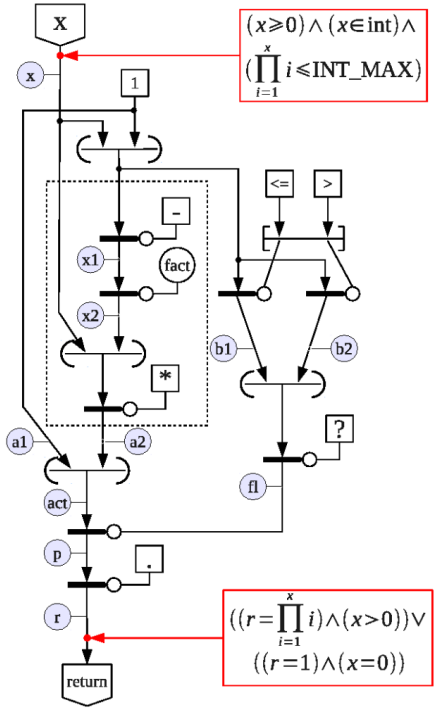
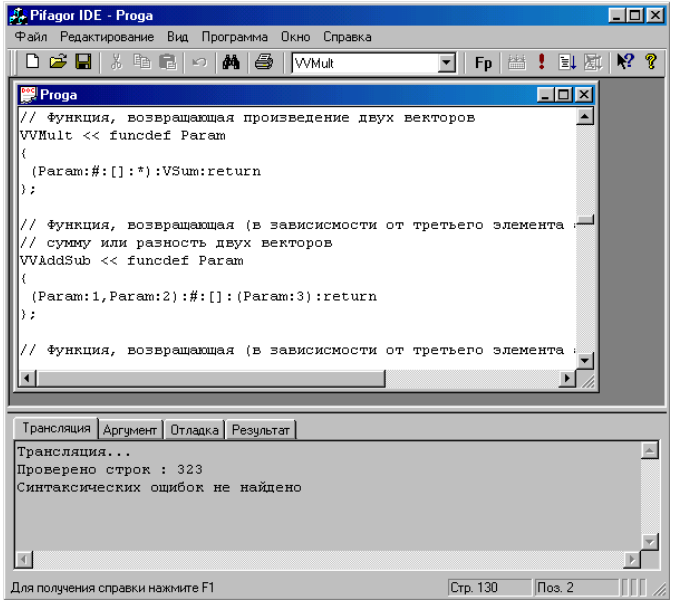
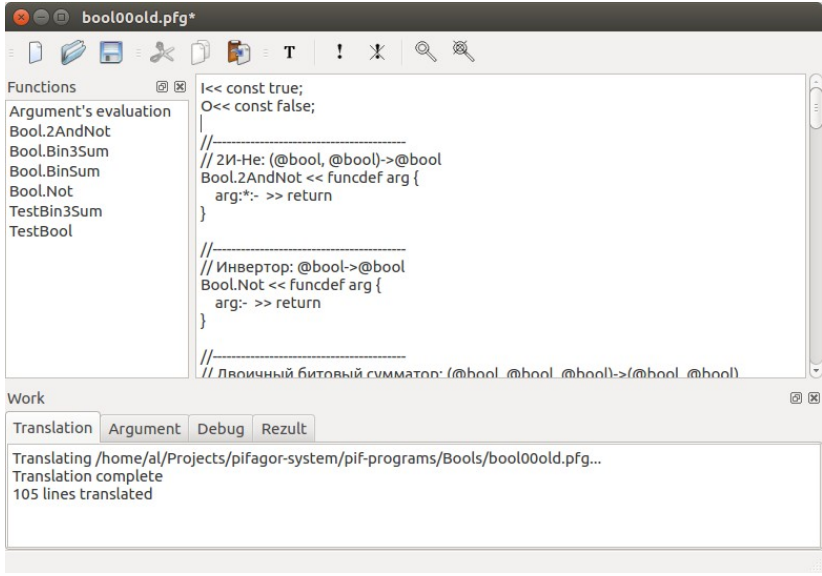
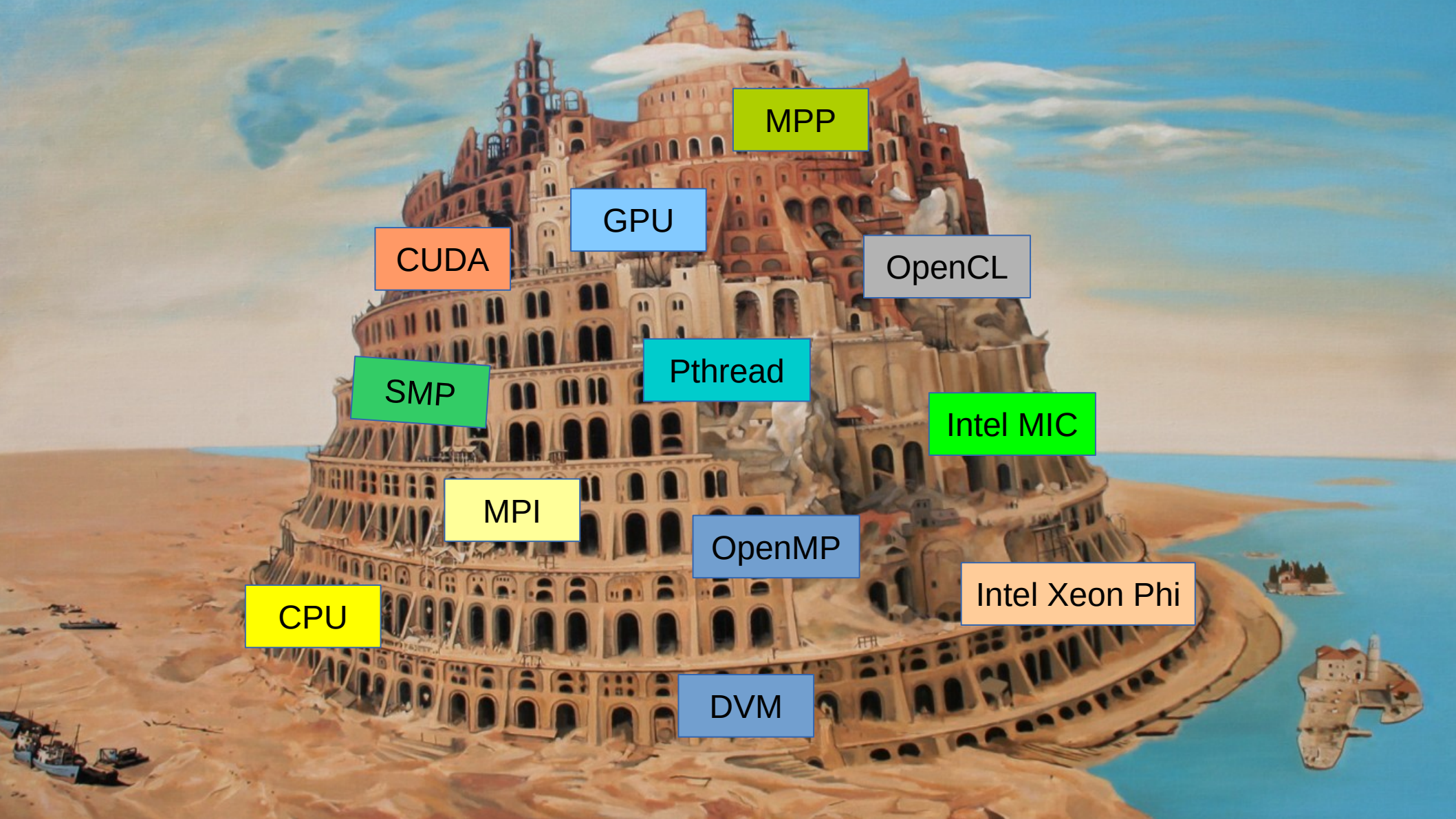
Архитектурно-независимое параллельное программирование

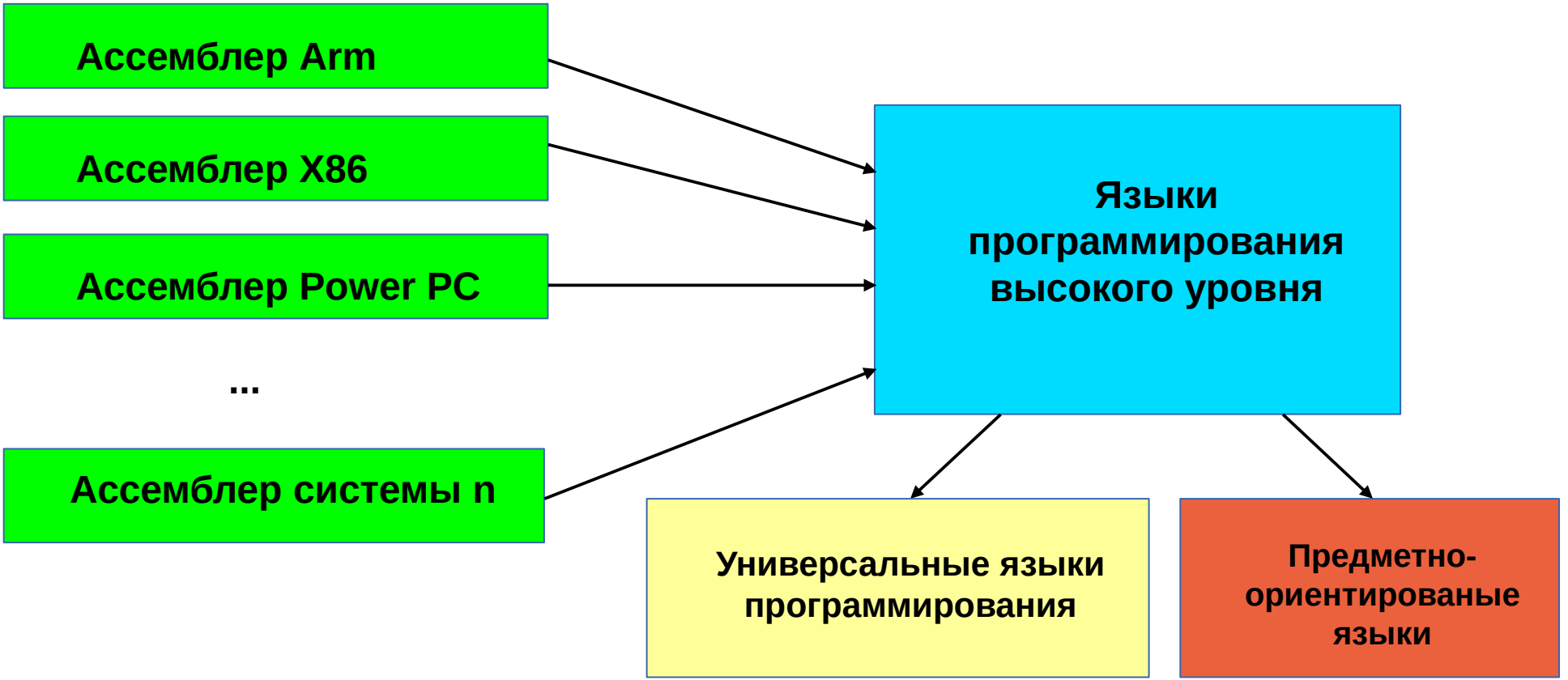




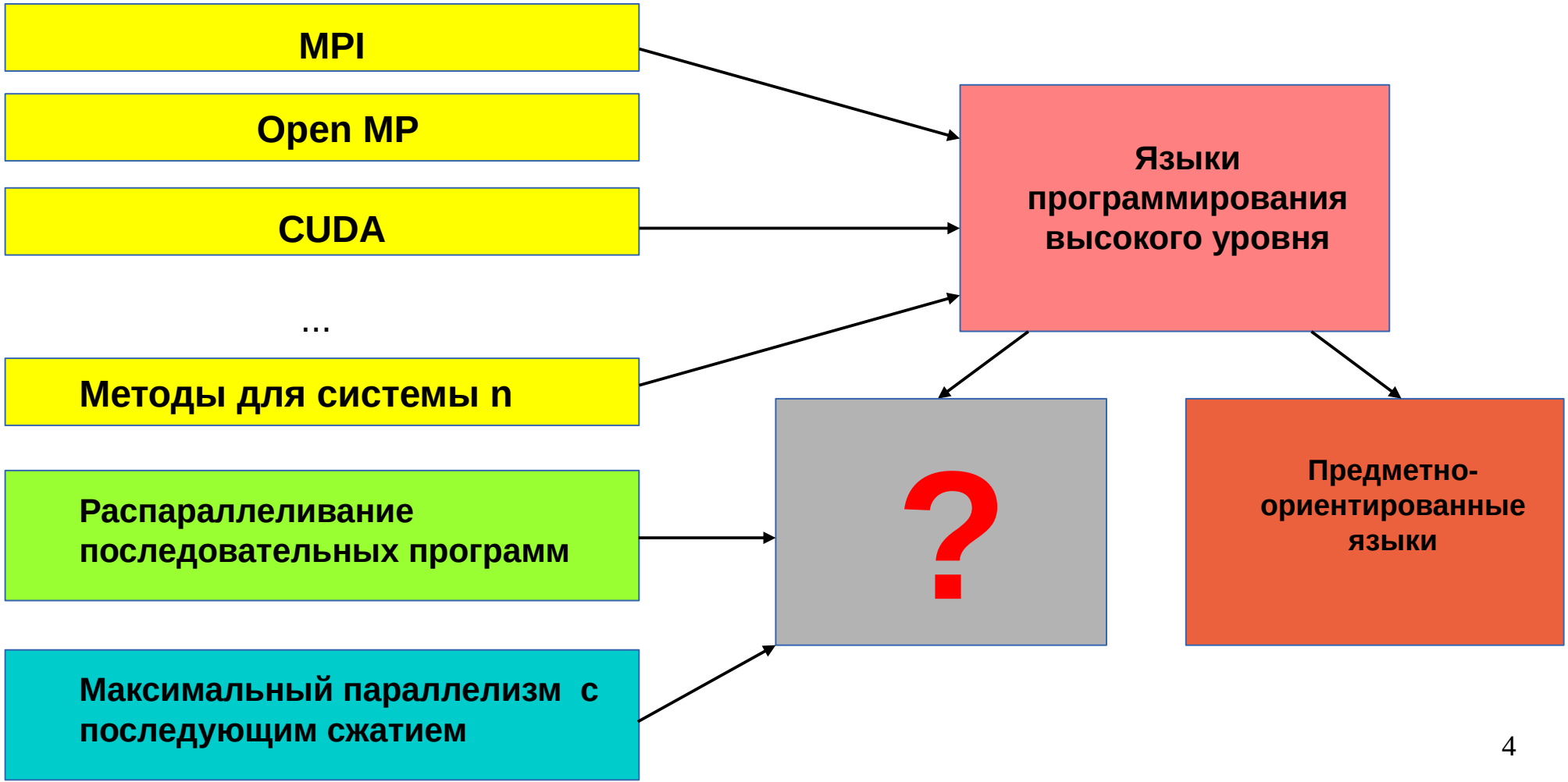



MPP
GPU
CUDA
OpenCL
Pthread
SMP
Intel MIC
MPI
OpenMP
Intel Xeon Phi
CPU
DVM

Архитектуры последовательных ВС



Архитектуры параллельных ВС



Общие идеи

http://softcraft.ru/fppp/

- Вместо распараллеливания последовательных программ – сжатие параллелизма программы, написанной в максимально параллельном стиле.
- Использование максимально параллельной программы для анализа, оптимизации, тестирования, верификации, отладки.
- Последующее сжатие параллелизма с применением формальных методов в соответствии с используемыми вычислительными ресурсами.

Стратегии управления в языках программирования

Множество условий готовности:

$$X = (Information, Resource, Acknowledgement)$$

Множество способов управления, обеспечивающих соблюдение условий готовности:

Automatic + Empty = Unevident - неявный

$$Z = \{Subjective, Unevident\}$$

S_1=*{Information×Z} – множество методов управления данными;*
S_2=*{Resource×Z} – множество методов управления ресурсами;*
S_3=*{Acknowledge×Z} – множество методов управления подтверждениями.*

Множество стратегий управления в ВС:

$$S=\{S_1 \times S_2 \times S_3\} = 8$$

Архитектурно-независимое параллельное программирование

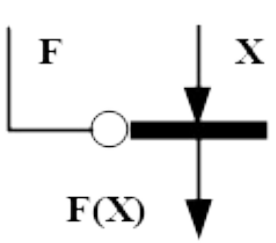
Общая стратегия управления:
Unevident Information, Unevident Resource, Unevident Acknowlement

(Empty Information, Empty Resource, Empty Acknowledge)-машина

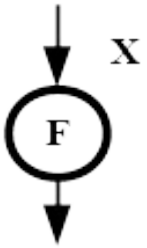
Модель функционально-потоковых параллельных вычислений

- Ориентация на неограниченные вычислительные ресурсы (принцип единственного использования вычислительных ресурсов).
- Запуск операции по готовности данных. Определяется в соответствии с аксиоматикой языка и его алгеброй преобразований (неявное управление вычислениями).
- Программа не имеет циклов, а значит и механизмов описания повторного использования ресурсов (все итеративные вычисления задаются через рекурсию).
- Для повышения эффективности при описании параллелизма используются специальные программо-формирующие структуры данных, определяемые как списки различного вида.
- Параллелизм на уровне элементарных операций.
- Программа – информационный граф.
- Выбор операций и аксиом, определяющих примитивы языка, ориентирован на наглядное текстовое представление информационного графа программы.

Операторы



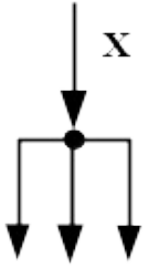
a)



b)

X:F or **F^X** ≡ **F(X)**

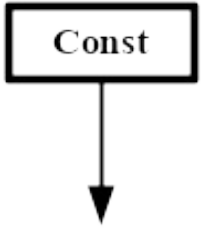
Оператор интерпретации




```
value >> name, or
  name << value
```

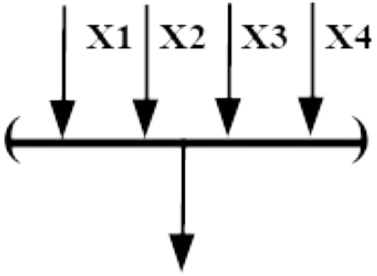
a)

Копирование



b)

Константа

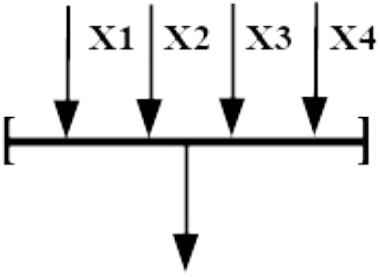


(X1, X2, X3, X4)

```
(a,b):+ ≡ a+b
```

c)

Список данных

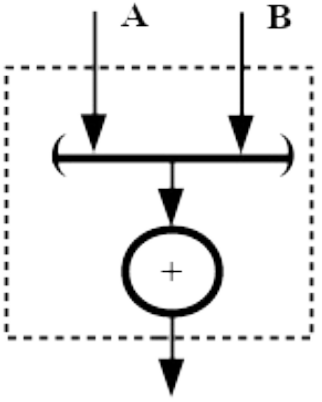


[X1, X2, X3, X4]

```
[x,y]:sin ≡
[x:sin,y:sin]
```

d)

Параллельный список



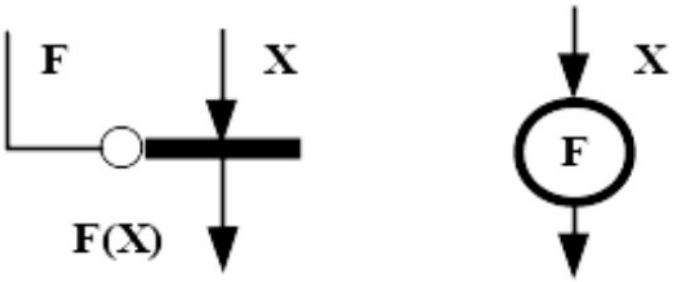
{ (A, B):+ }

e)

Задержанный список

Оператор интерпретации

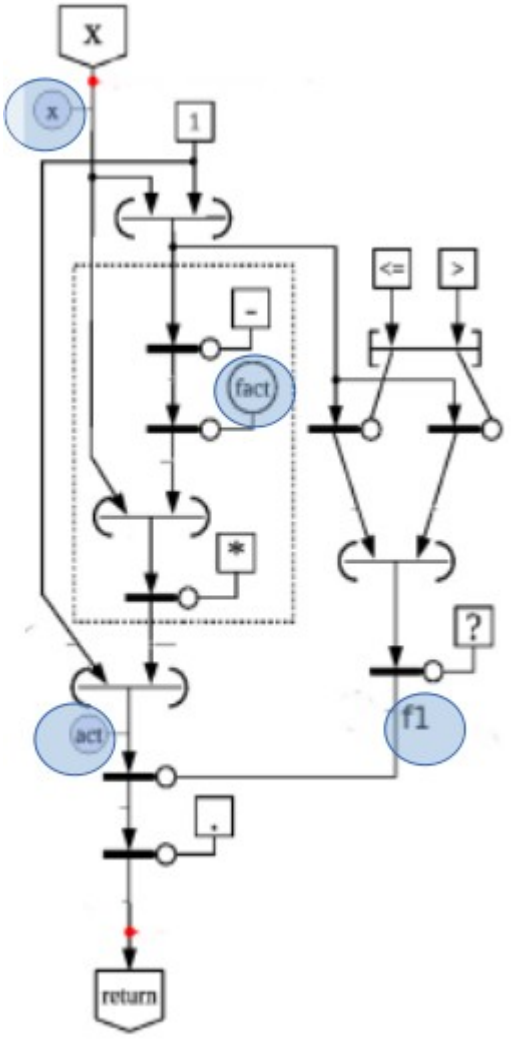
Имеется только одна функция: оператор интерпретации
Обеспечивает преобразование входного набора данных X, выступающего в качестве аргумента, в выходной набор Y, который является результатом, используя при этом входной набор F в качестве функции, определяющей алгоритм преобразования.



Постфиксная запись: **X:F ⇒ Y**
Префиксная запись: **F^X ⇒ Y**

(a,b):+ *или* **+^(a,b)** ≡ **a+b**
x: sin *или* **sin ^ x** ≡ **sin(x)**
((b,b):*,((4, a):*,c):*):- ≡ **b * b - 4 * a * c**

```
// Factorial definition
fact << funcdef x {
  // selector of variant
  fl << ((x,1):[<=,>]):?;
  // two possible variants
  act << (
    1,
    { (x, (x,1):-:fact ):* }
  );
  return << act:fl:.;
}
```



Преобразования параллельных списков

[(x1, x2), (x3, x4),(x5,x6)]:+
=> [(x1, x2):+, (x3, x4):+,(x5,x6):+]

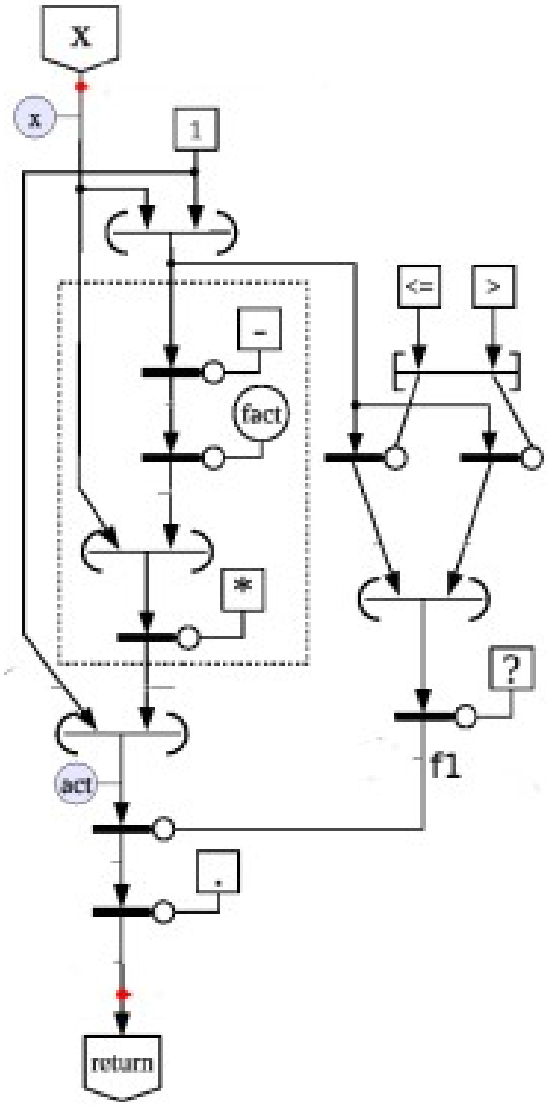
(x1,x2):[+,-,*,/]
=> [(x1,x2):+,(x1,x2):-,(x1,x2):*,(x1,x2):/]

Общий случай:

[x,y,z]:[f,g,h]
=> [x:[f,g,h],y:[f,g,h],z:[f,g,h]]
=> [[x:f,x:g,x:h],[y:f,y:g,y:h],[z:f,z:g,z:h]]

В программе:

(x,1):[<=,>]
=> [(x,1):<=,(x,1):>]



Ветвления

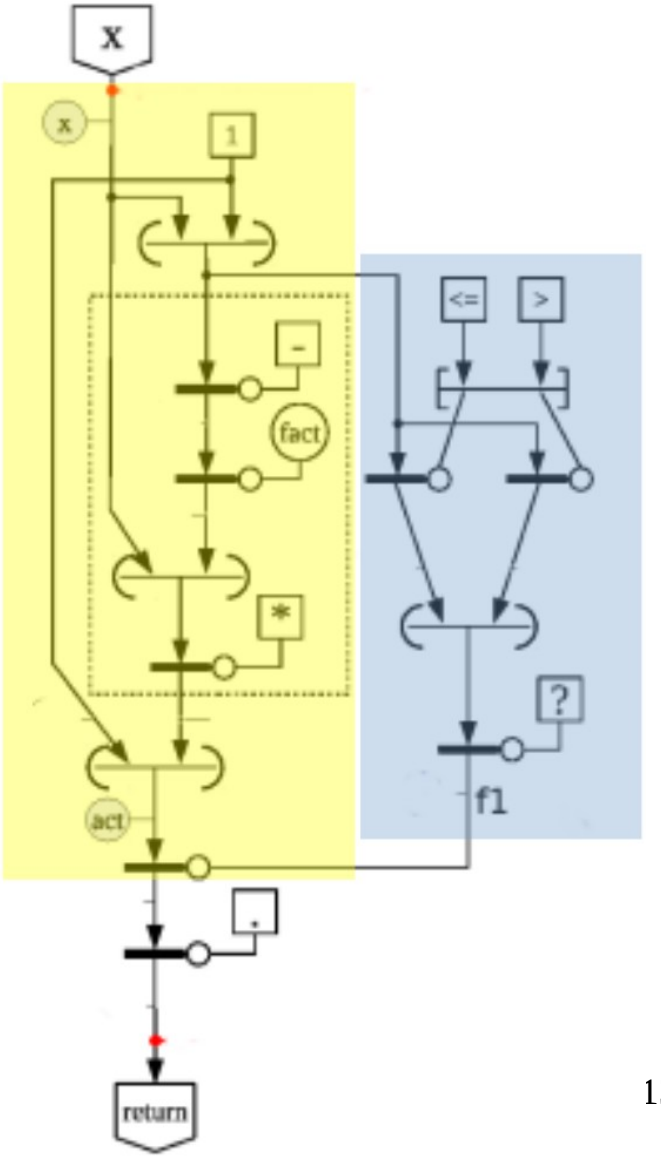
Селектор:

(3,7,2,6):3 => 2
(x,1):[<=,>]
=> [true, false] or [false, true]
(true,true,false,false,true,false):?
==> [1,2,5]

В программе:

((x,1):[<=,>]):?
=> ([true, false]):? or ([false, true])?
=> (true, false):? or (false, true):?
=> [1] or [2] == 1 or 2 >> f1

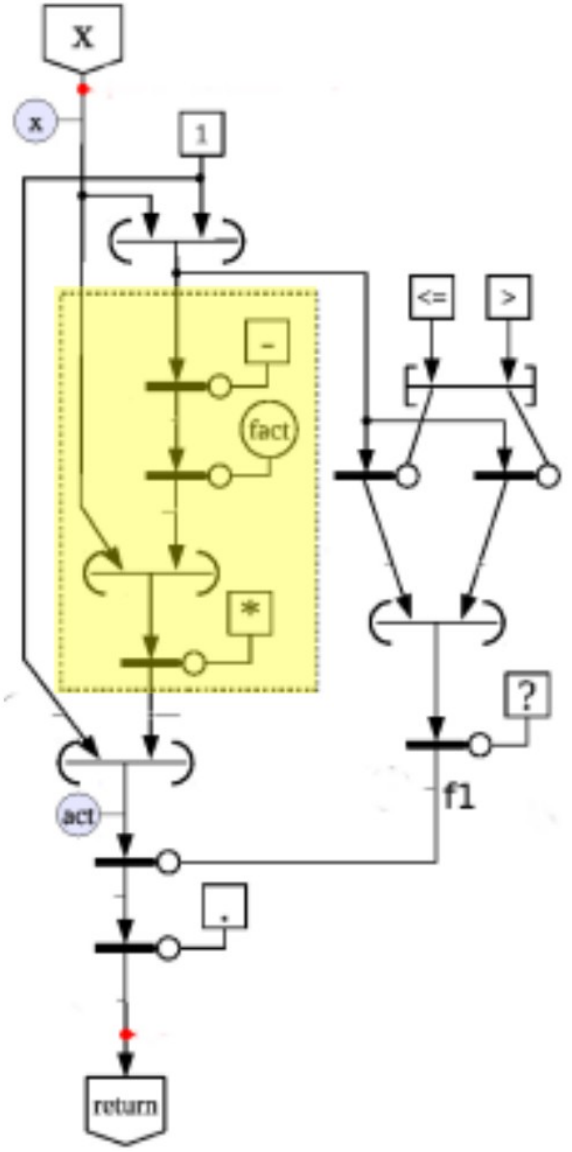
(1, { (x, (x,1):-:fact):* }) : 1 => 1 // 0! or 1!
(1, { (x, (x,1):-:fact):* }) : 2 => { (x, (x,1):-:fact):* } // x*(x-1)!



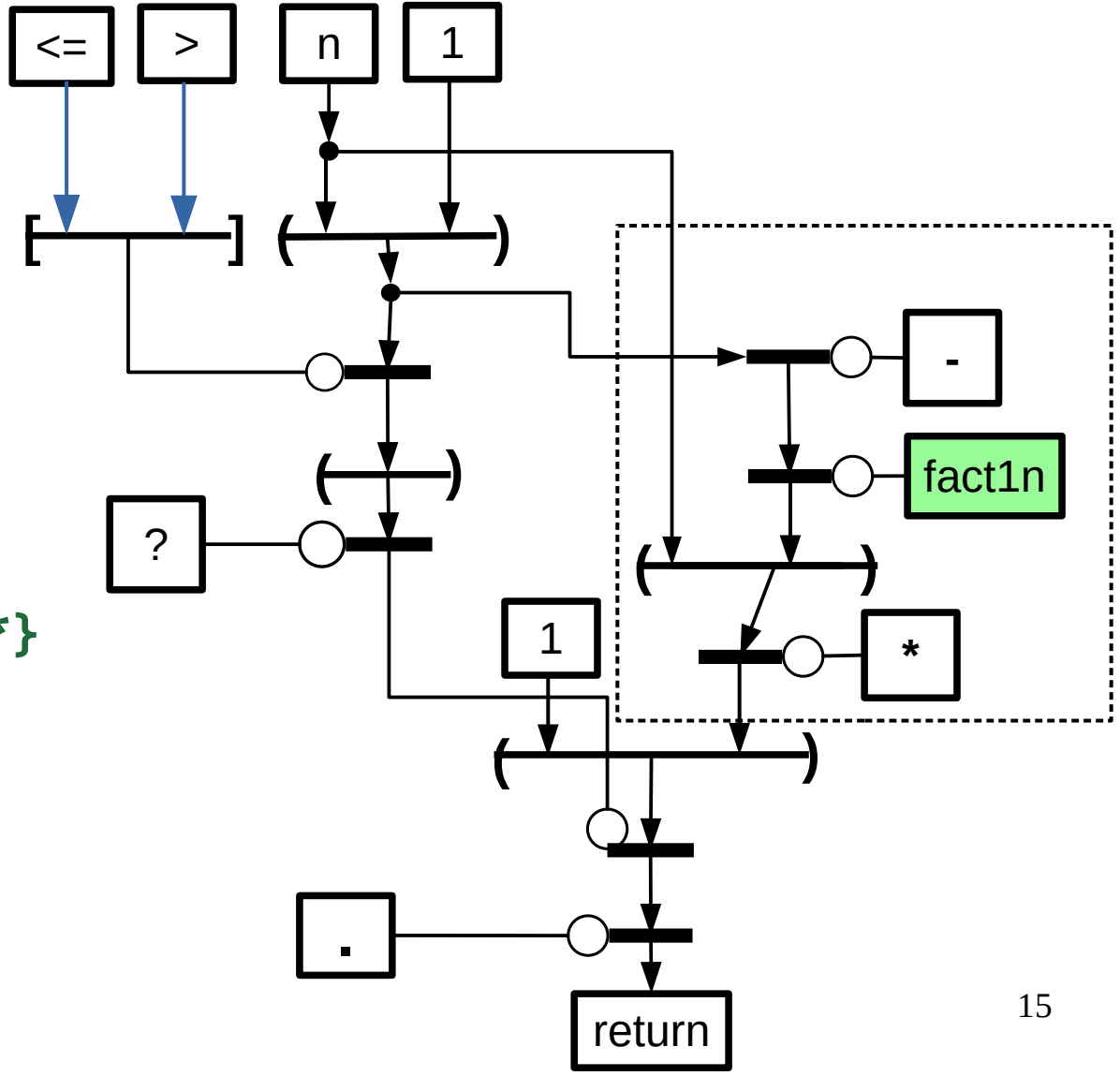
Задержанный список

В программе:

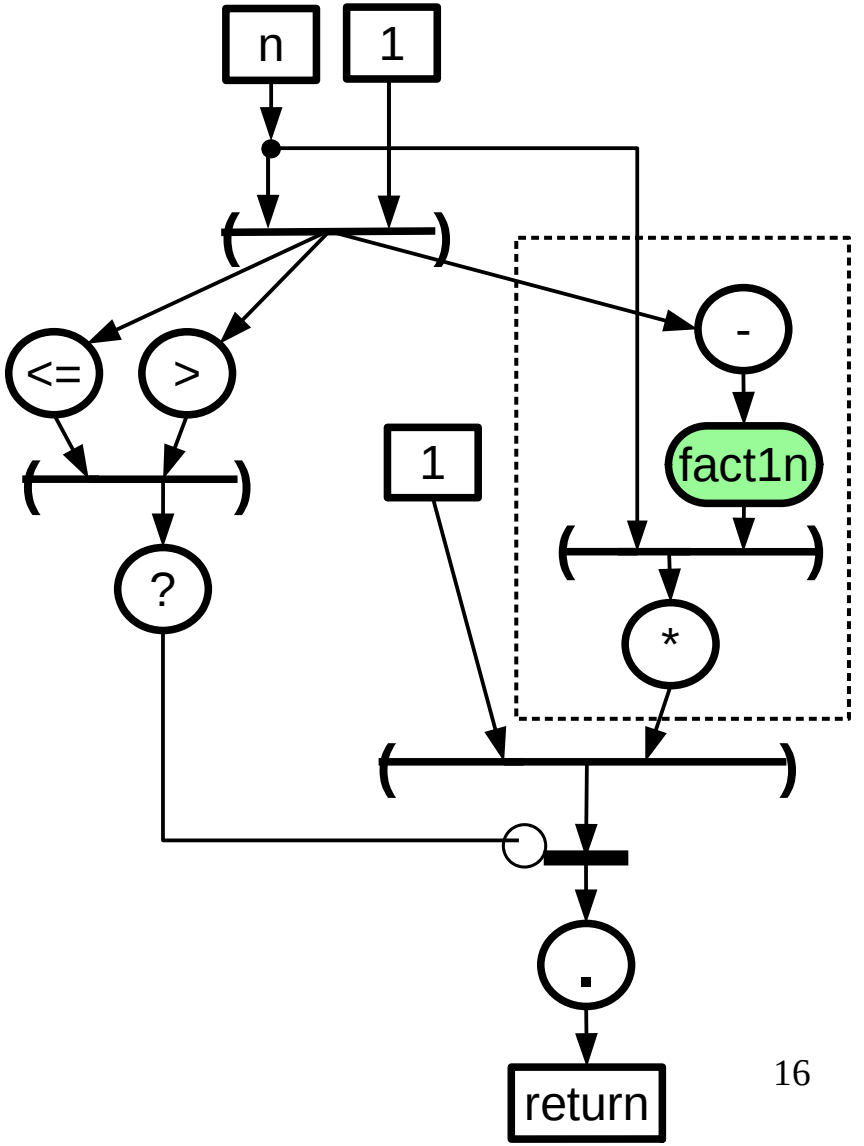
```
{ (x, (x,1):-:fact ):* }:.
    => [ (x, (x,1):-:fact ):* ]:.
    => … == x*(x-1)!
```

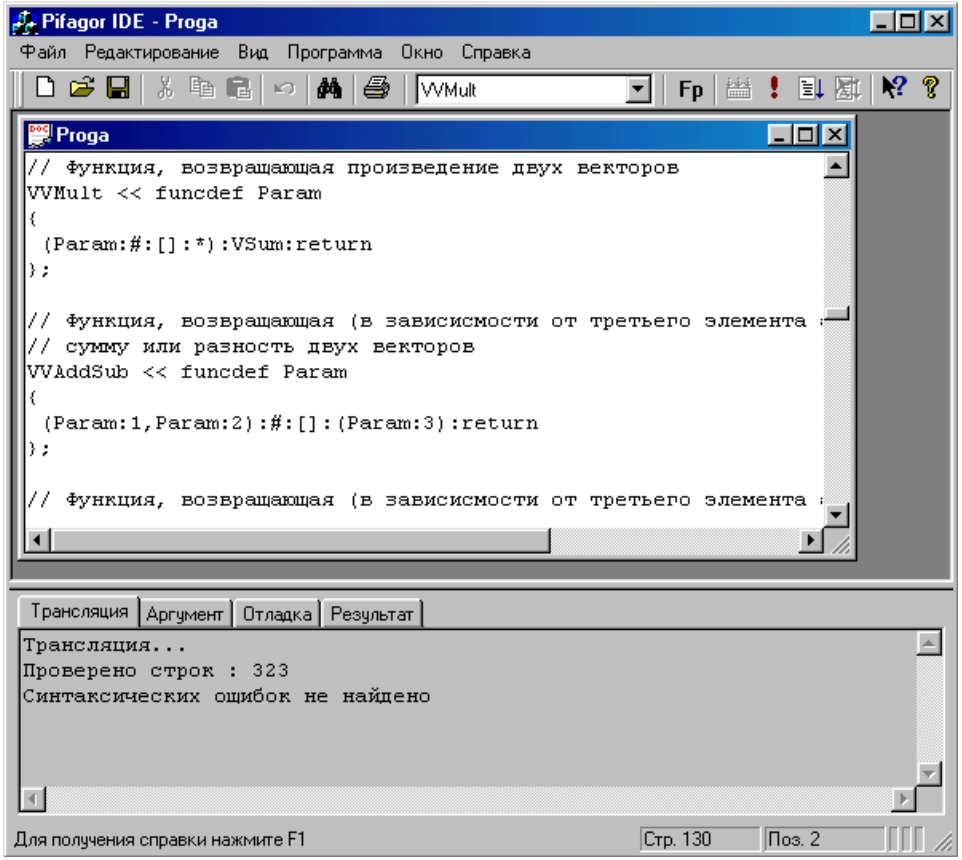


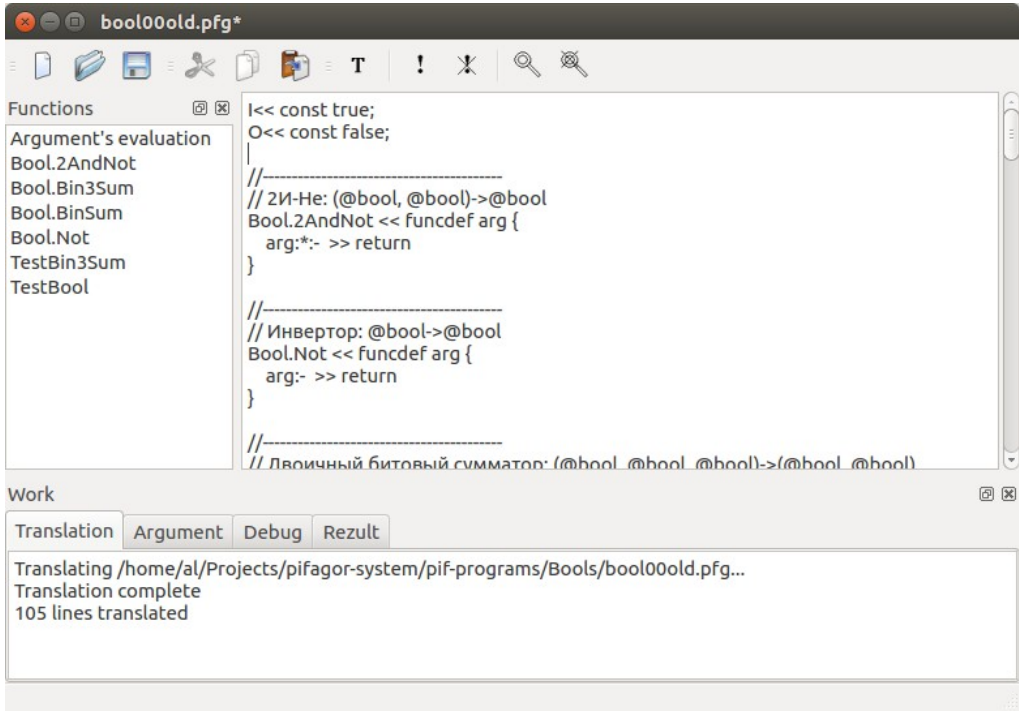
```
fact1n << funcdef n {
    n1<< (n,1);
    [(n1:[<=,>]):?]^
    (
        1,
        {(n, n1:-:fact1n):*}
    ):.
  >>return
}
```



```
fact1n << funcdef n {
    n1<< (n,1);
    [(n1:[<=,>]):?]^
    (
        1,
        {(n, n1:-:fact1n):*}
    ):.
  >>return
}
```




Pifagor IDE - Proga
Файл Редактирование Вид Программа Окно Справка
VVMult
Fp
Proga
// Функция, возвращающая произведение двух векторов
VVMult << funcdef Param
{
(Param:#:[]:*):VSum:return
};
// Функция, возвращающая (в зависисмости от третьего элемента
// сумму или разность двух векторов
VVAddSub << funcdef Param
{
(Param:1,Param:2):#:[]:(Param:3):return
};
// Функция, возвращающая (в зависисмости от третьего элемента
Трансляция
Аргумент
Отладка
Результат
Трансляция...
Проверено строк : 323
Синтаксических ошибок не найдено
Для получения справки нажмите F1
Стр. 130
Поз. 2


bool00old.pfg*
Functions
Argument's evaluation
Bool.2AndNot
Bool.Bin3Sum
Bool.BinSum
Bool.Not
TestBin3Sum
TestBool
I<< const true;
O<< const false;
//----------
// 2И-Не: (@bool, @bool)->@bool
Bool.2AndNot << funcdef arg {
arg:*:- >> return
}
//----------
// Инвертор: @bool->@bool
Bool.Not << funcdef arg {
arg:- >> return
}
//----------
Work
Translation
Argument
Debug
Rezult
Translating /home/al/Projects/pifagor-system/pif-programs/Bools/bool00old.pfg...
Translation complete
105 lines translated

Описание комбинационных схем

```
// Одноразрядный двоичный сумматор
// Аргумент = (0|1, 0|1, 0|1) - (1-е слагаемое, 2-е слагаемое, перенос)
Add1 << funcdef x1x2p {
    x1 << x1x2p:1;   // первое слагаемое
    x2 << x1x2p:2;   // второе слагаемое
    p  << x1x2p:3;   // заем из предыдущего разряда

    // Формирование переноса
    //C = (P && X1) || (((!P && X1) || (P && !X1)) && X2);
    c << ((p,x1):And2,(((p:Not,x1):And2,(p,x1:Not):*):Or2,x2):And2):Or2;

    // Формирование результата
    //Y = (!P && ((X1 && !X2) || (!X1 && X2))) ||
          (P && ((X1 && X2) || (!X1 && !X2)));
    y << ((p:Not,((x1,x2:Not):And2,(x1:Not,x2):And2):Or2):And2,
         (p,((x1,x2):And2,(x1:Not,x2:Not):And2):Or2):And2):Or2;
    (c,y) >>return;
}
```

Описание комбинационных схем

```
// Двухразрядный двоичный сумматор,
// построенный на основе одноразрядных
// Аргумент = (1-е слагаемое, 2-е слагаемое)
// Результат = (перенос, старший разряд, младший
разряд)
Sum2 << funcdef s1s2 {
    s1<< s1s2:1;
    s2<< s1s2:2;
    y0<< (s1:2,s2:2,0):Add1;
    y1<< (s1:1,s2:1,y0:1):Add1;
    (y1:1,y1:2,y0:2) >>return
}
```

Описание комбинационных схем

```
SumMxM << funcdef s1s2 {
    s1<< s1s2:1;
    s2<< s1s2:2;
    count<< s1s2:3;
    sum<< s1s2:4;
    // Проверка равенства длин слагаемых
    [(count,0):(<,=,>):?]^
    (
        0,    // некорректная длина
        sum,  // сумма накоплена
        {     // правая рекурсия с накоплением
            (s1,s2,(count,1):-,
            (([s1,s2]:count,sum:1):Add1:[],sum:-1:[])):SumMxM
        }
    ):. >>return
}
```

Используемые источники

1. Легалов А. И. Стратегии управления в вычислительных системах и языках программирования - 2002 г. Электронный ресурс: http://softcraft.ru/parallel/strat/

2. Функционально-потоковое параллельное программирование - Электронный ресурс: http://softcraft.ru/fppp/

3. А. И. Легалов, Ф. А. Казаков, Д. А. Кузьмин, Д. В. Привалихин. Функциональная модель параллельных вычислений и язык программирования "Пифагор" - 2003 г. Электронный ресурс: http://softcraft.ru/parallel/fpp/

4. Легалов А. И. Построение алгоритмов за счет ограничений, накладываемых на максимальный параллелизм - Электронный ресурс: http://softcraft.ru/parallel/maxsort/