ИСП РАН

СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
SIBERIAN FEDERAL UNIVERSITY

Инструментальная поддержка создания и трансформации функционально-потоковых параллельных программ

Александр Легалов, Владимир Васильев, Иван Матковский, Мария Ушакова

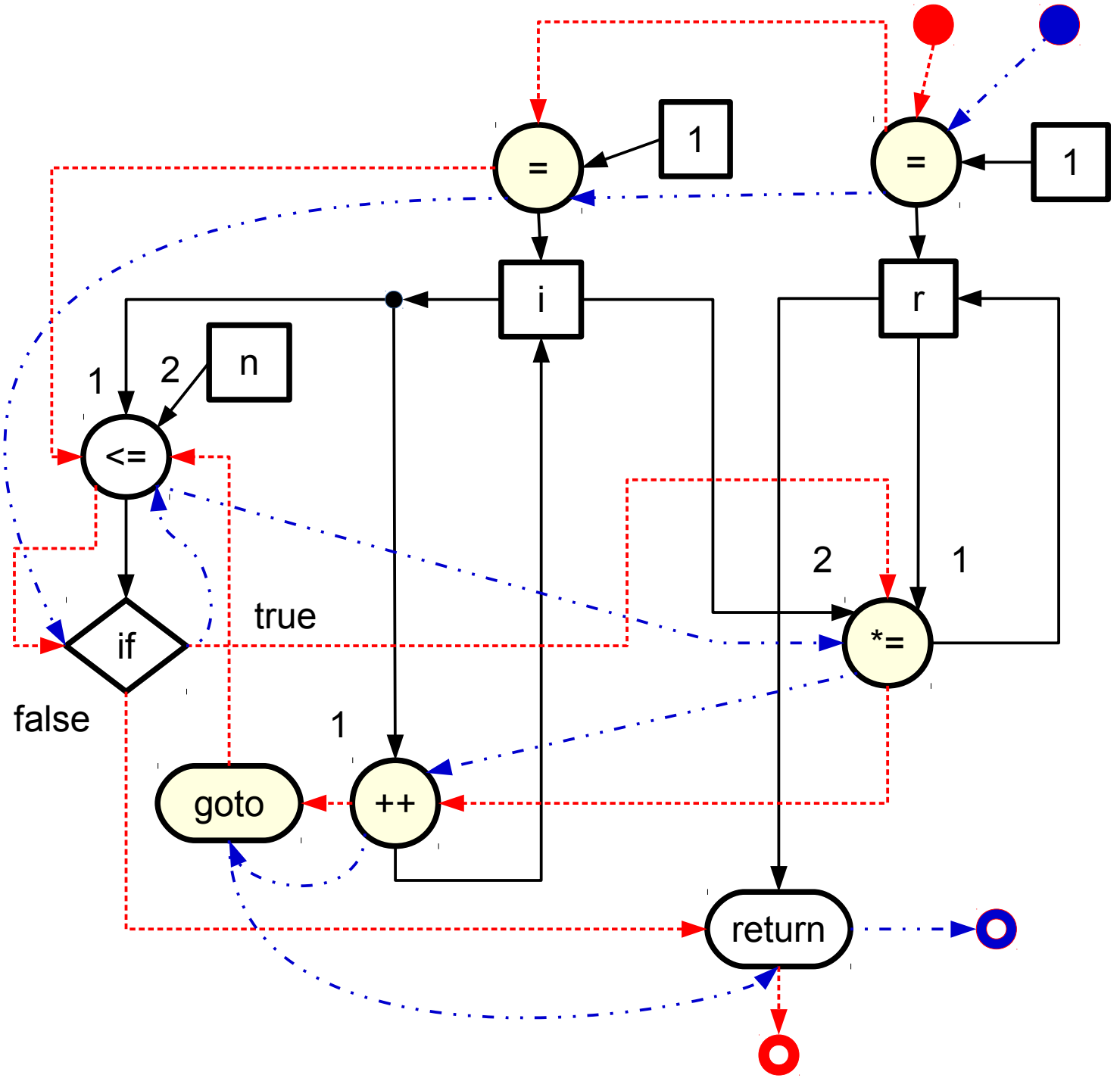
Сибирский федеральный университет

Исследование выполняется при финансовой поддержке РФФИ в рамках научного проекта № 17-07-00288

Общие идеи

- 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.

```
// Факториал от 1 до n
int fact1n(int n)
{
    int r = 1;
    int i = 1;
loop:
    if(i <= n) {
        r *= i;
        i++;
        goto loop;
    }
    return r;
}
```

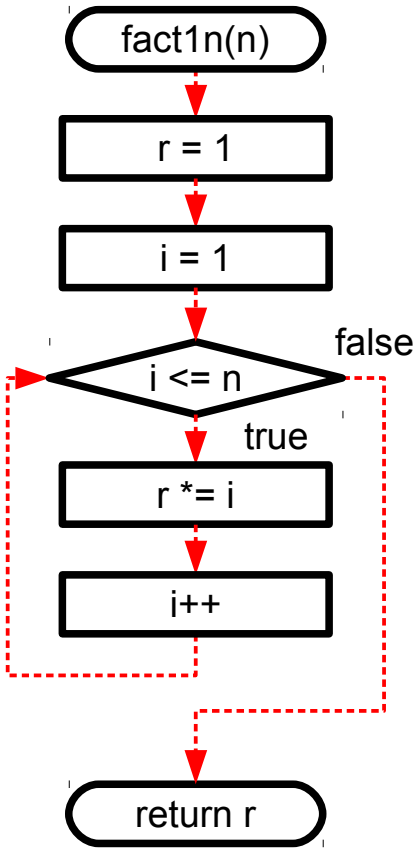


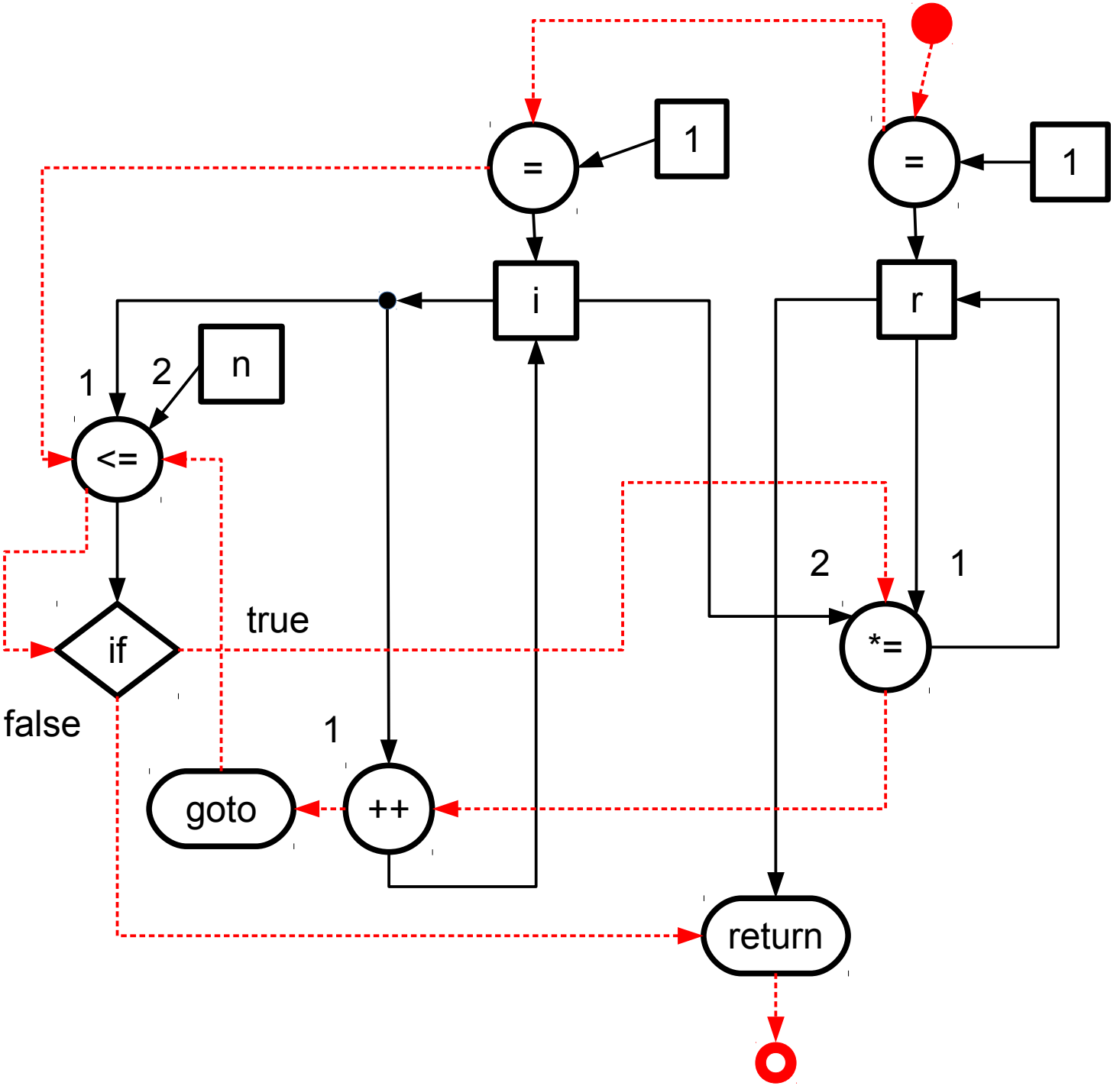
Отношение расположения

Отношение между данными

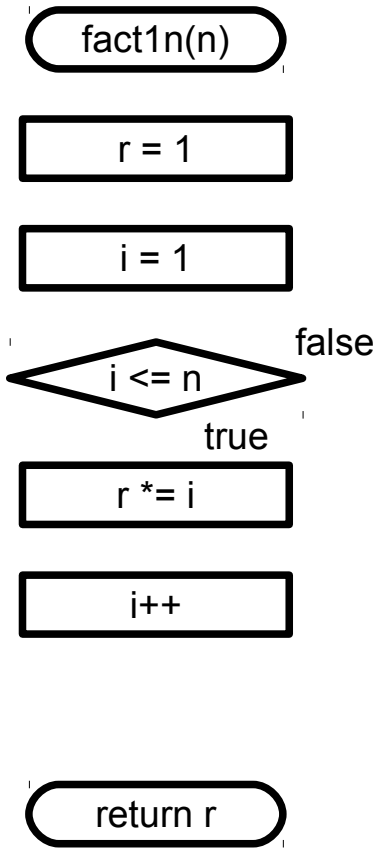
Отношение управления

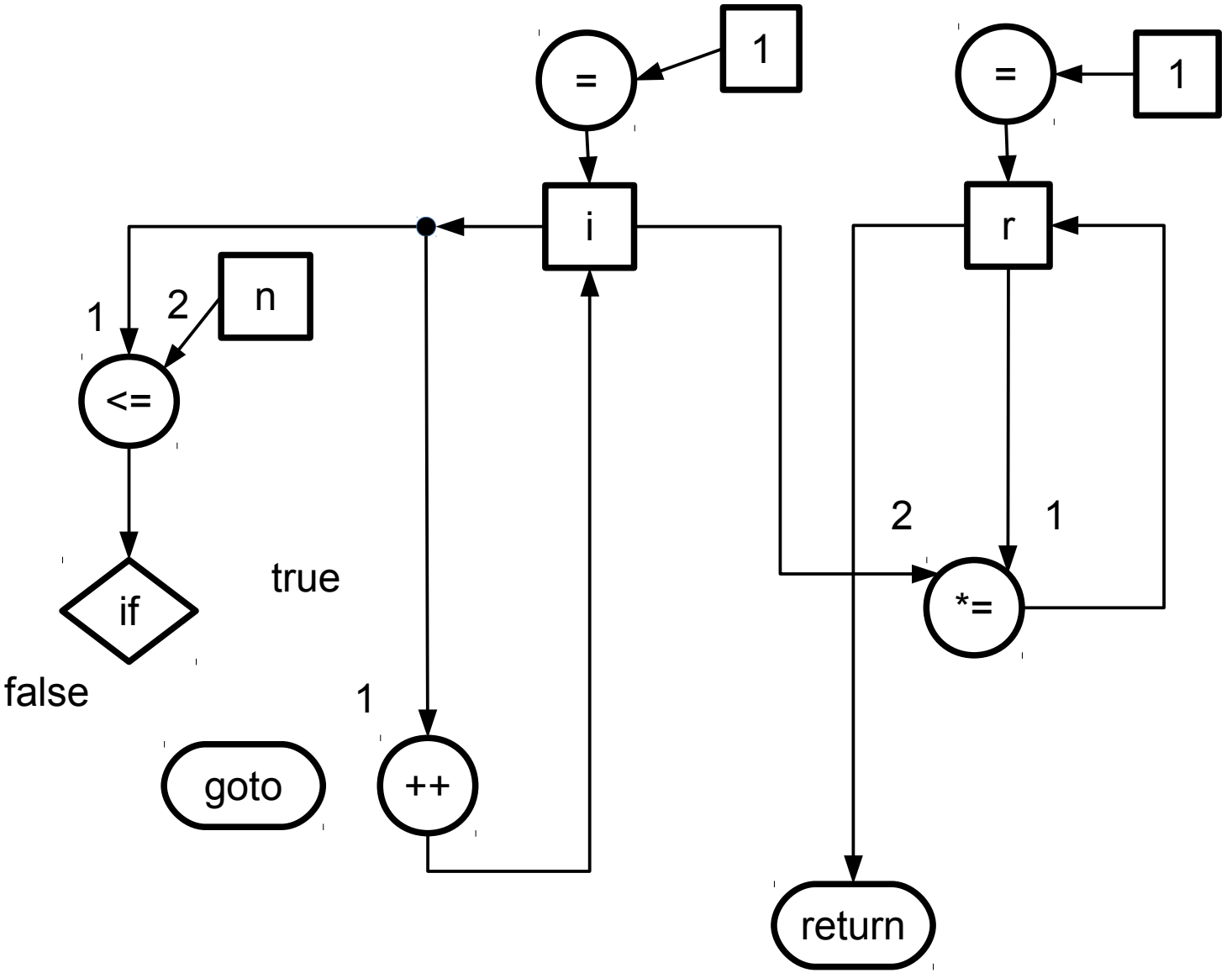
```c
// Факториал от 1 до n
int fact1n(int n)
{
    int r = 1;
    int i = 1;
loop:
    if(i <= n) {
        r *= i;
        i++;
        goto loop;
    }
    return r;
}
```



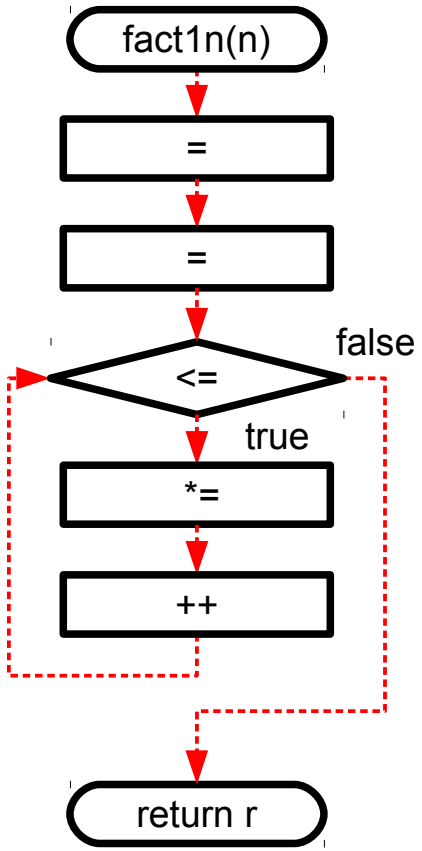


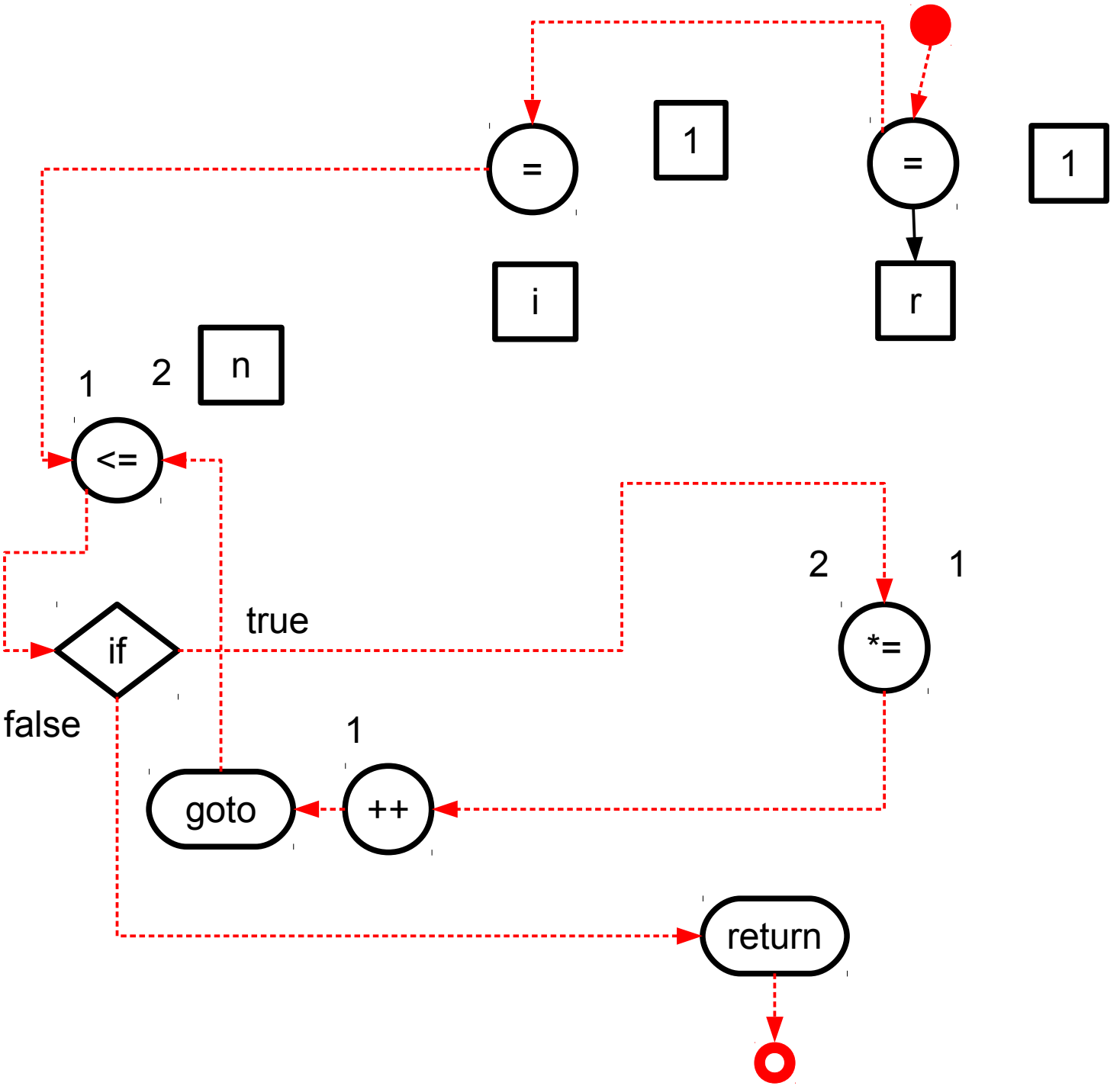
```
// Факториал от 1 до n
int fact1n(int n)
{
    int r = 1;
    int i = 1;
loop:
    if(i <= n) {
        r *= i;
        i++;
        goto loop;
    }
    return r;
}
```



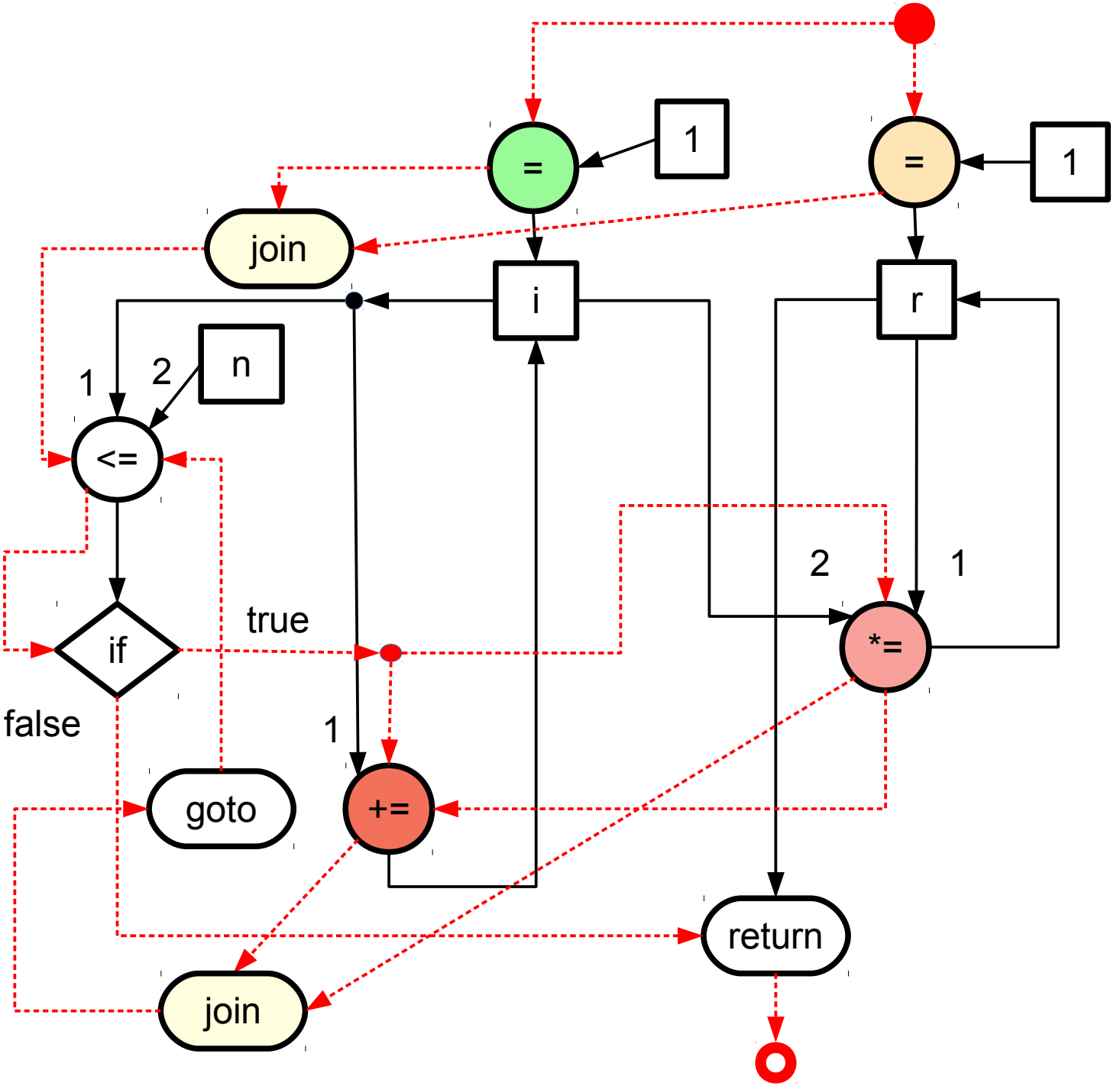


```
// Факториал от 1 до n
int fact1n(int n)
{
    int r = 1;
    int i = 1;
loop:
    if(i <= n) {
        r *= i;
        i++;
        goto loop;
    }
    return r;
}
```





```
// Факториал от 1 до n
int fact1n(int n)
{
  fork
    int r = 1;
    int i = 1;
  join
loop:
    if(i <= n) {
      fork
        r *= i;
        i++;
      join
      goto loop;
    }
    return r;
}
```



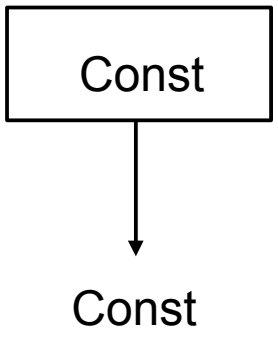
Модель функционально-потоковых параллельных вычислений

- 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 (**проблеменая ориентация на параллелизм**).
- Параллелизм на уровне **элементарных операций**.
- Программа – информационный граф (**data flow**).
- Выбор операций и аксиом, определяющих примитивы языка, ориентирован на наглядное **текстовое** представление информационного графа программы.

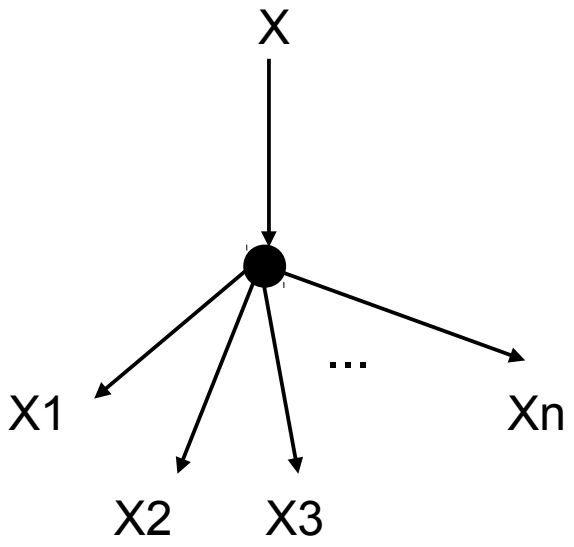
Backus J. Can programming be liberated from von Neuman style? A functional stile and its algebra of programs // CACM. 1978. Vol. 21, N 8. P. 613–641.

Кутепов, В. П. Модель асинхронных вычислений значений функций в языке функциональных схем. / В. П. Кутепов, В. Н. Фальк // Программирование. – 1978. – № 3. – С. 3-15.

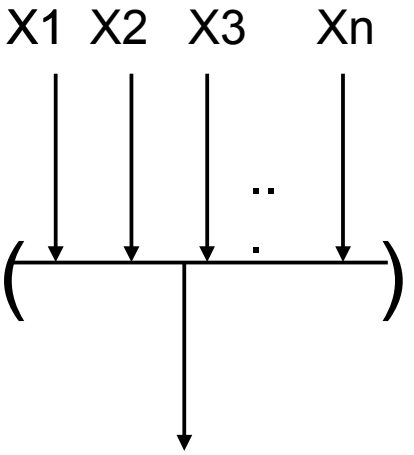
Основные операторы модели вычислений



Константный оператор

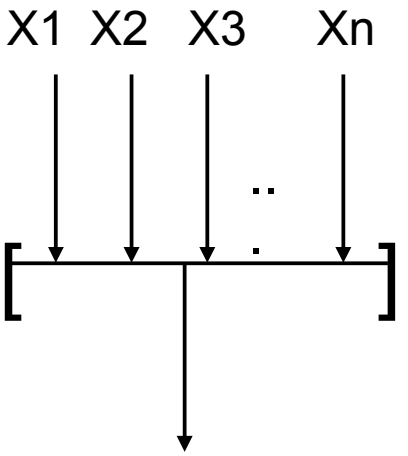


Копирование данных



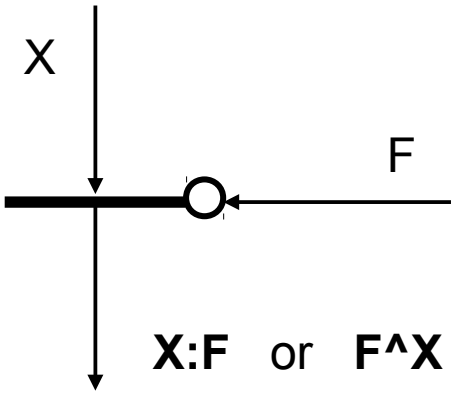
(X1, X2, X3, … , Xn)

Список данных



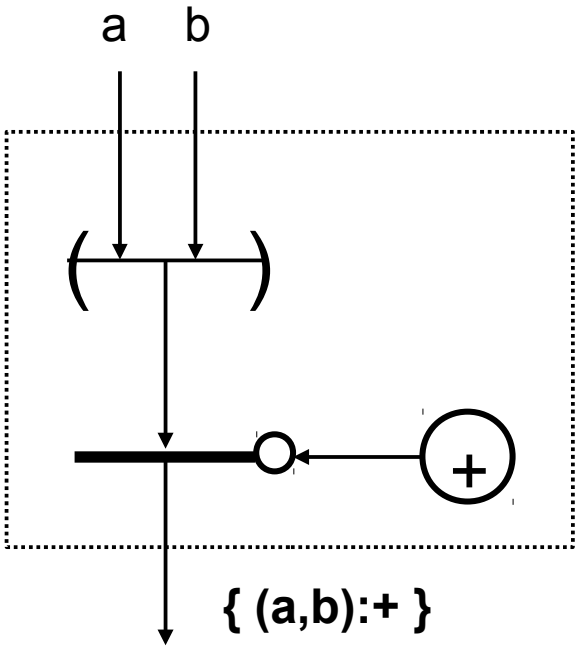
[X1, X2, X3, … , Xn]

Параллельный список



X:F or **F^X** equal **F(X)**

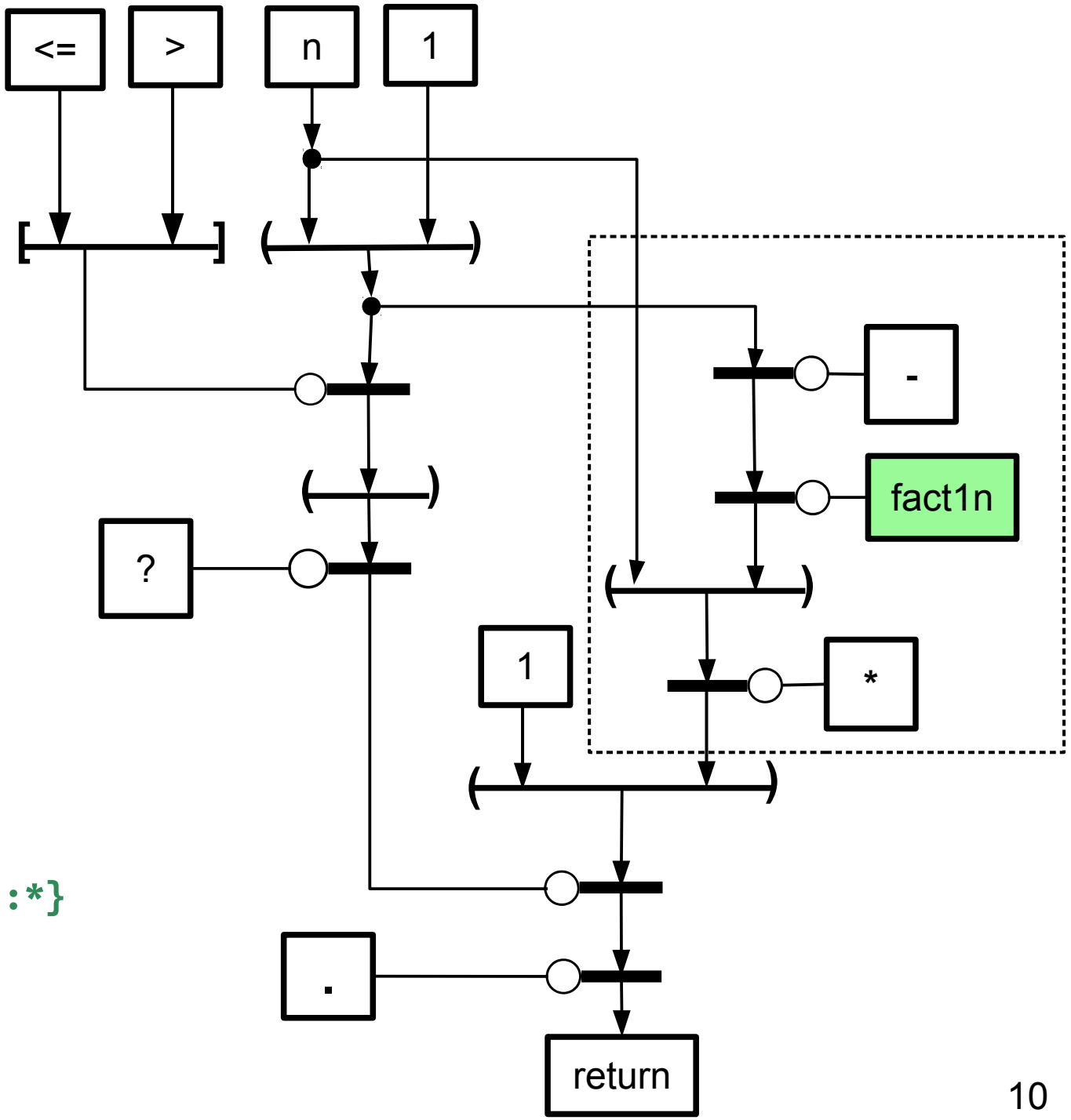
Оператор интерпретации



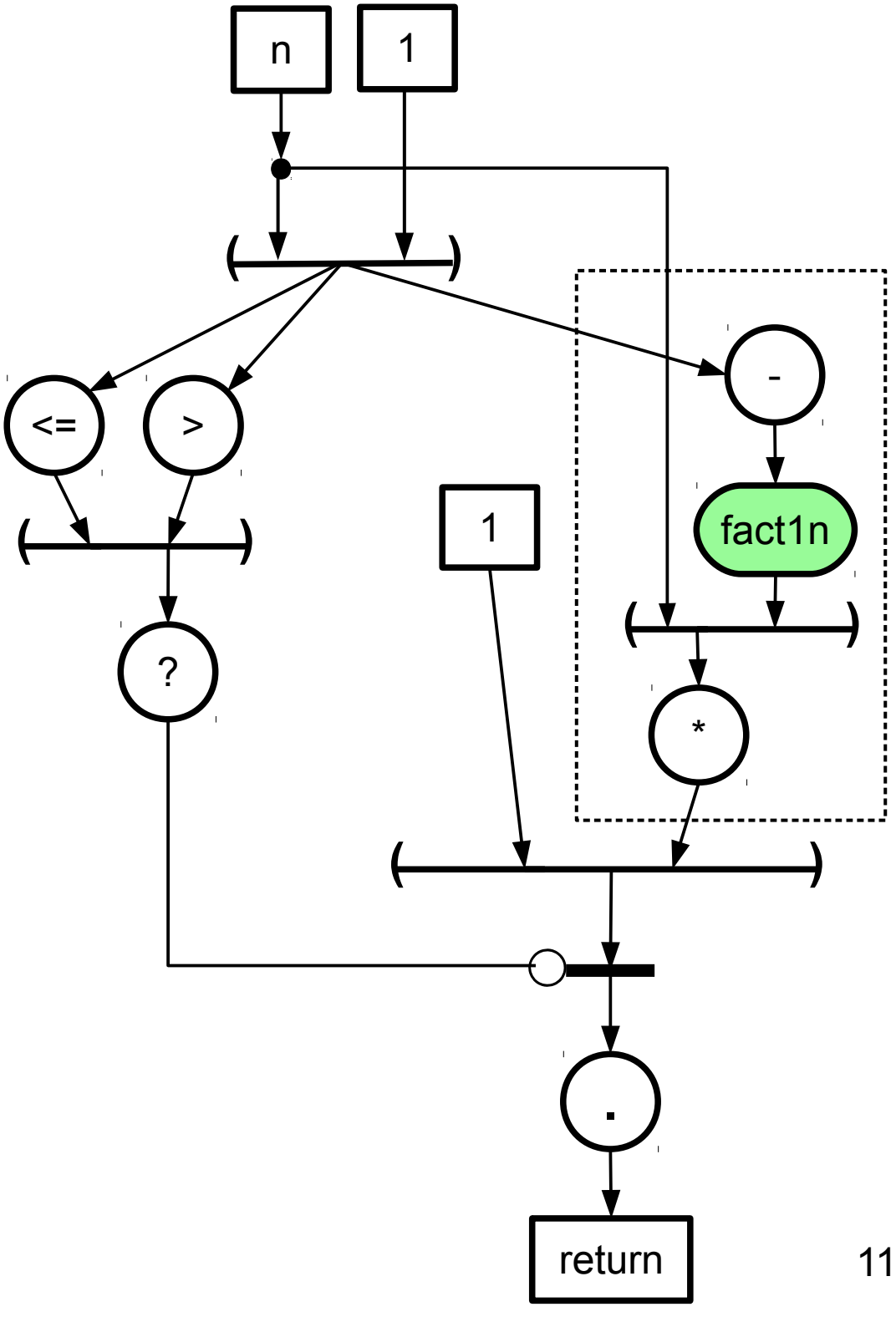
{ (a,b):+ }

Задержанный список

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

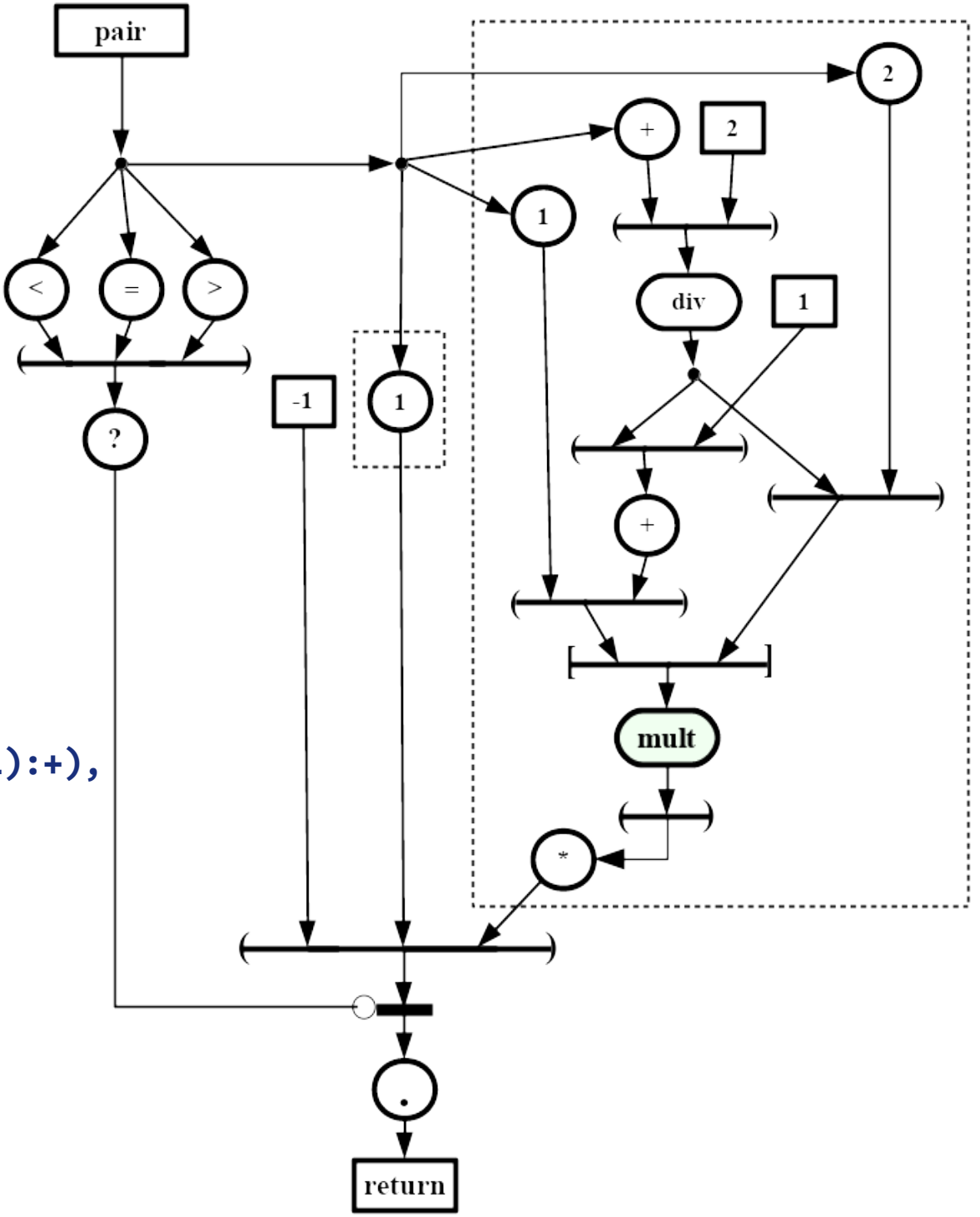


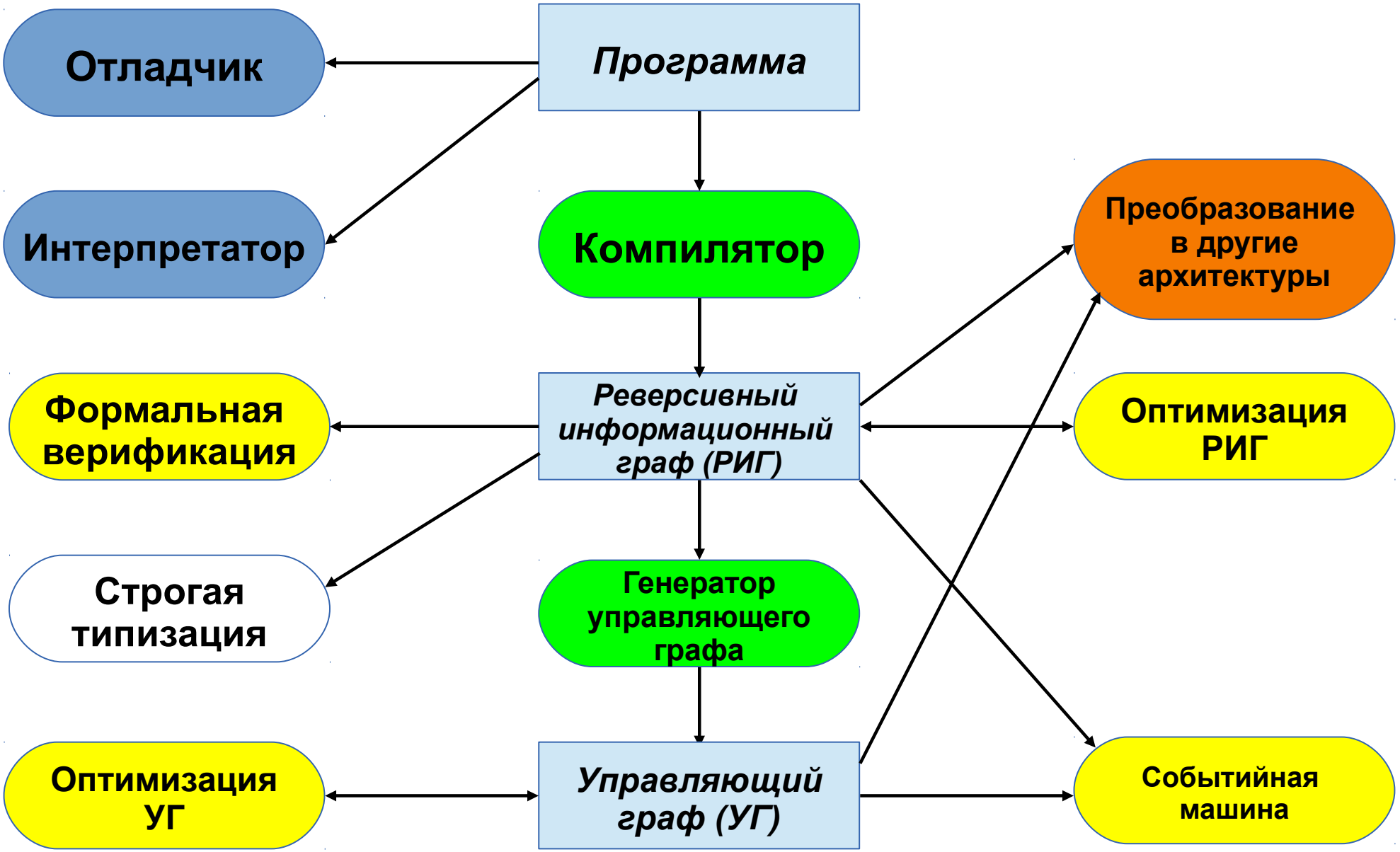
```
fact1n << funcdef n {
    n1<< (n,1);
    [(n1:[<=,>]):?]^
    (
        1,
        {(n, n1:-:fact1n):*}
    ):.
>>return
}
```



```
parfact<< funcdef n {
  selector<< (n,0):(<,=,>):?;
  selector^ (-1, 1, {(n,1):mult}):.
      >>return
}
```

```
mult<< funcdef pair {
  selector<< pair:(<,=,>):?;
  selector^ (
    -1,
    {pair:1},
    {([(pair:1,((pair:+,2):div>>half,1):+),
       (half, pair:2)
    ]:mult):*
    }
  ):. >>return
}
```




Отладчик
Программа
Интерпретатор
Компилятор
Преобразование в другие архитектуры
Формальная верификация
Реверсивный информационный граф (РИГ)
Оптимизация РИГ
Строгая типизация
Генератор управляющего графа
Оптимизация УГ
Управляющий граф (УГ)
Событийная машина

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

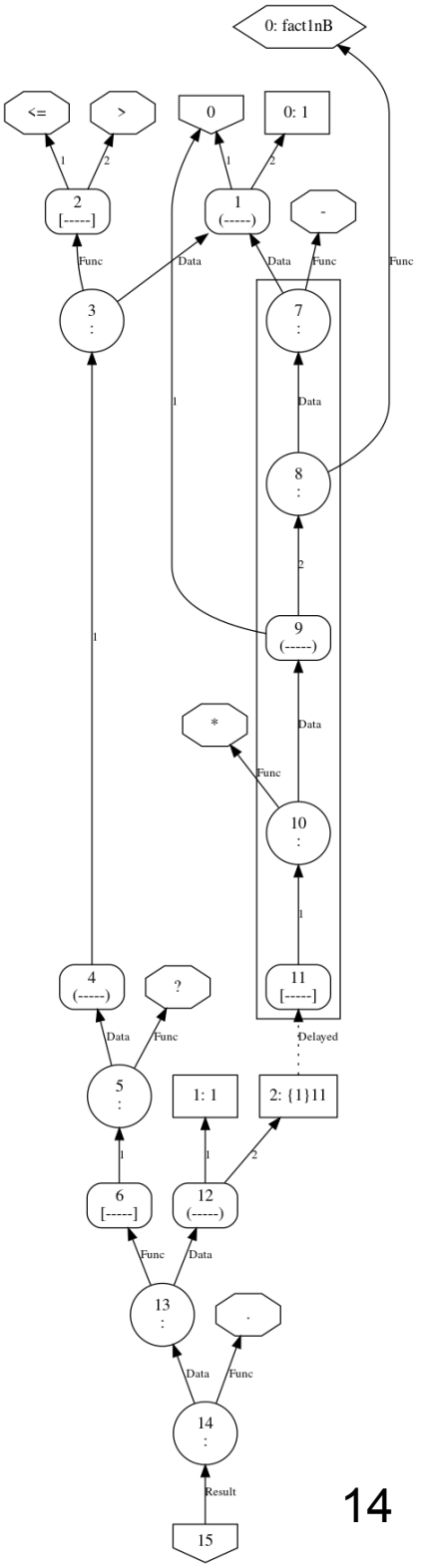
External

0	fact1nB

Local

0	1
1	1
2	{1}11

id	delay	operation	links
0	0	arg	
1	0	(---)	0 loc:0
2	0	[---]	<= >
3	0	:	1 2
4	0	(---)	3
5	0	:	4 ?
6	0	[---]	5
7	1	:	1 -
8	1	:	7 ext:0
9	1	(---)	0 8
10	1	:	9 *
11	1	[---]	10
12	0	(---)	loc:1 loc:2
13	0	:	12 6
14	0	:	13 .
15	0	return	14



fact1nB

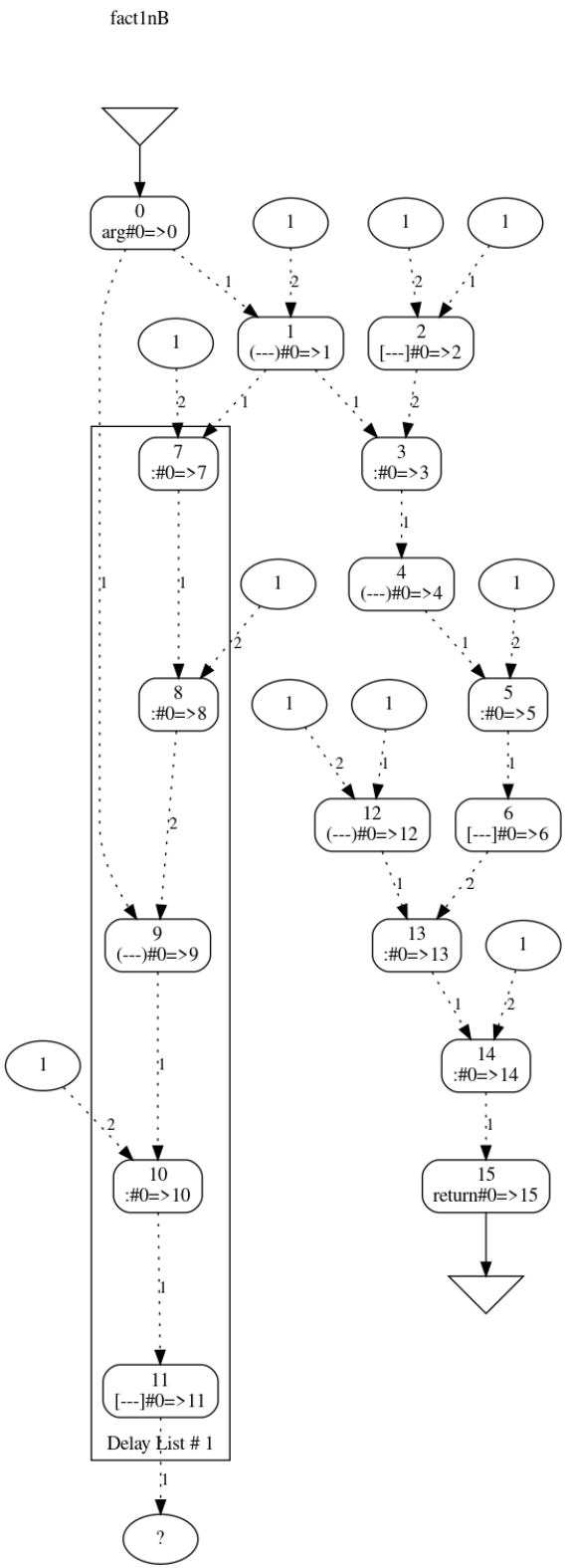
id	delay	automat	inode	links
0	0	arg,0	0	links:1,1;9,1;
1	0	(---),0	1	links:3,1;7,1;
2	0	[---],0	2	links:3,2;
3	0	:,0	3	links:4,1;
4	0	(---),0	4	links:5,1;
5	0	:,0	5	links:6,1;
6	0	[---],0	6	links:13,2;
7	1	:,0	7	links:8,1;
8	1	:,0	8	links:9,2;
9	1	(---),0	9	links:10,1;
10	1	:,0	10	links:11,1;
11	1	[---],0	11	
12	0	(---),0	12	links:13,1;
13	0	:,0	13	links:14,1;
14	0	:,0	14	links:15,1;
15	0	return,0	15	

Signals: (Number, Node, Input)

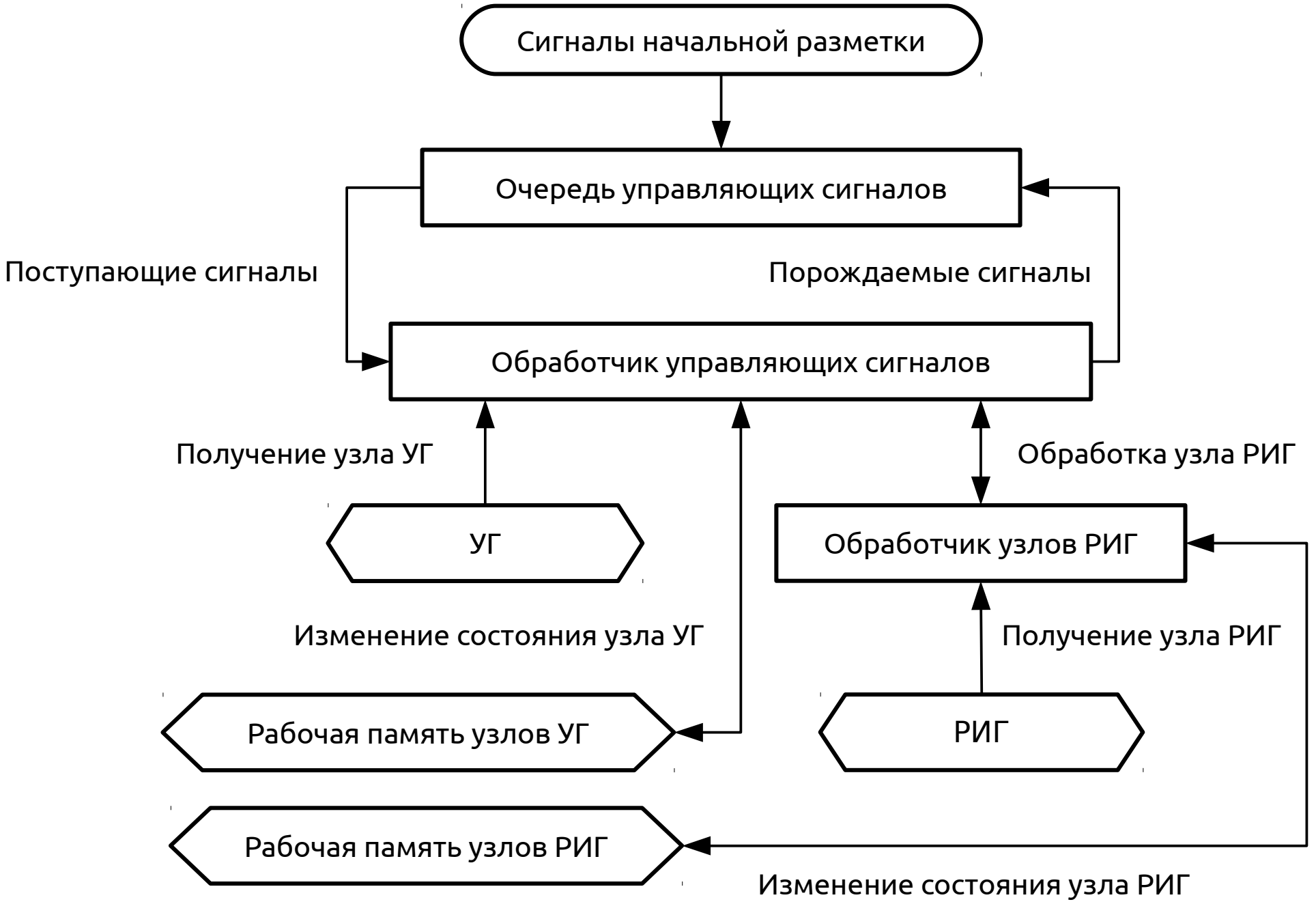
0	1	2
1	2	1
2	2	2
3	5	2
4	7	2
5	8	2
6	10	2
7	12	1
8	12	2
9	14	2

Dynamic links: (Number, Delay list, Node)

0	1	11



Обобщённая структура событийного процессора



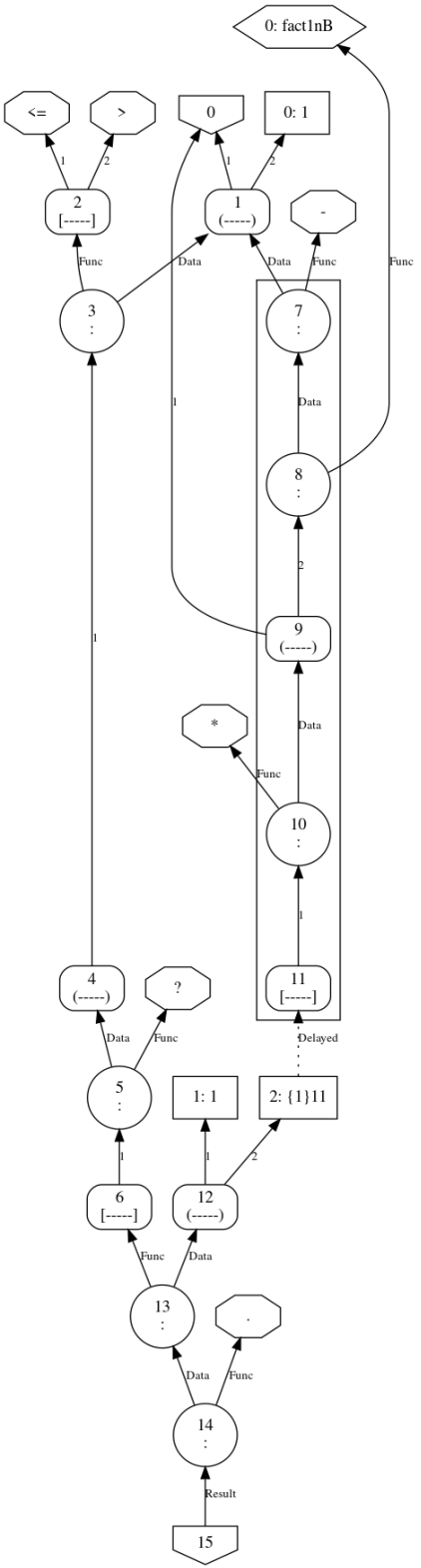
External

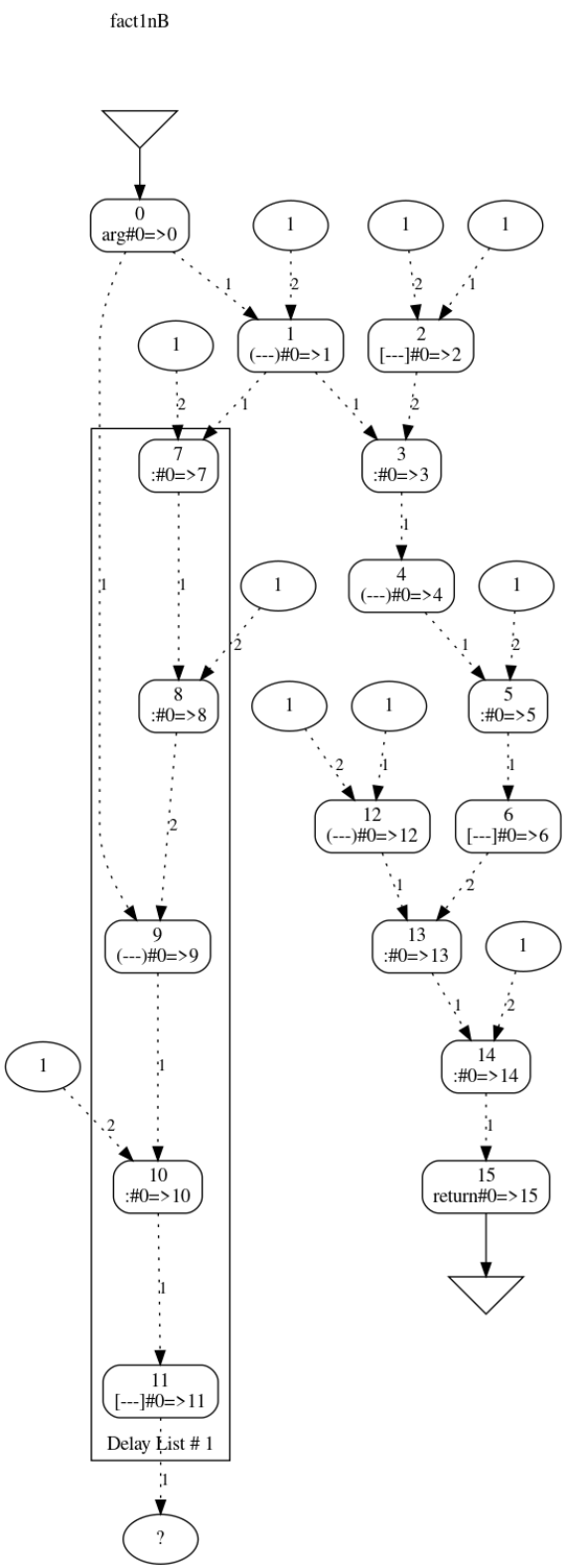
0	fact1nB

Local

0	1
1	1
2	{1}11

id	delay	operation	links
0	0	arg	
1	0	(---)	0 loc:0
2	0	[---]	<= >
3	0	:	1 2
4	0	(---)	3
5	0	:	4 ?
6	0	[---]	5
7	1	:	1 -
8	1	:	7 ext:0
9	1	(---)	0 8
10	1	:	9 *
11	1	[---]	10
12	0	(---)	loc:1 loc:2
13	0	:	12 6
14	0	:	13 .
15	0	return	14





Оптимизация функционально-потоковых параллельных программ

Удаление неиспользуемого кода

Оптимизация в повторяющихся вычислениях

Непосредственная (inline) подстановка простых функций

Удаление повторяющегося кода

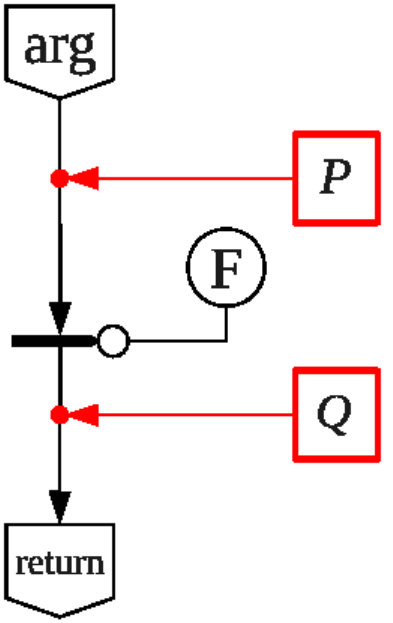
Оптимизация на основе эквивалентных преобразований

Удаление избыточных управляющих зависимостей

Порождение управляющего графа, реализующего последовательный обход вершин РИГ

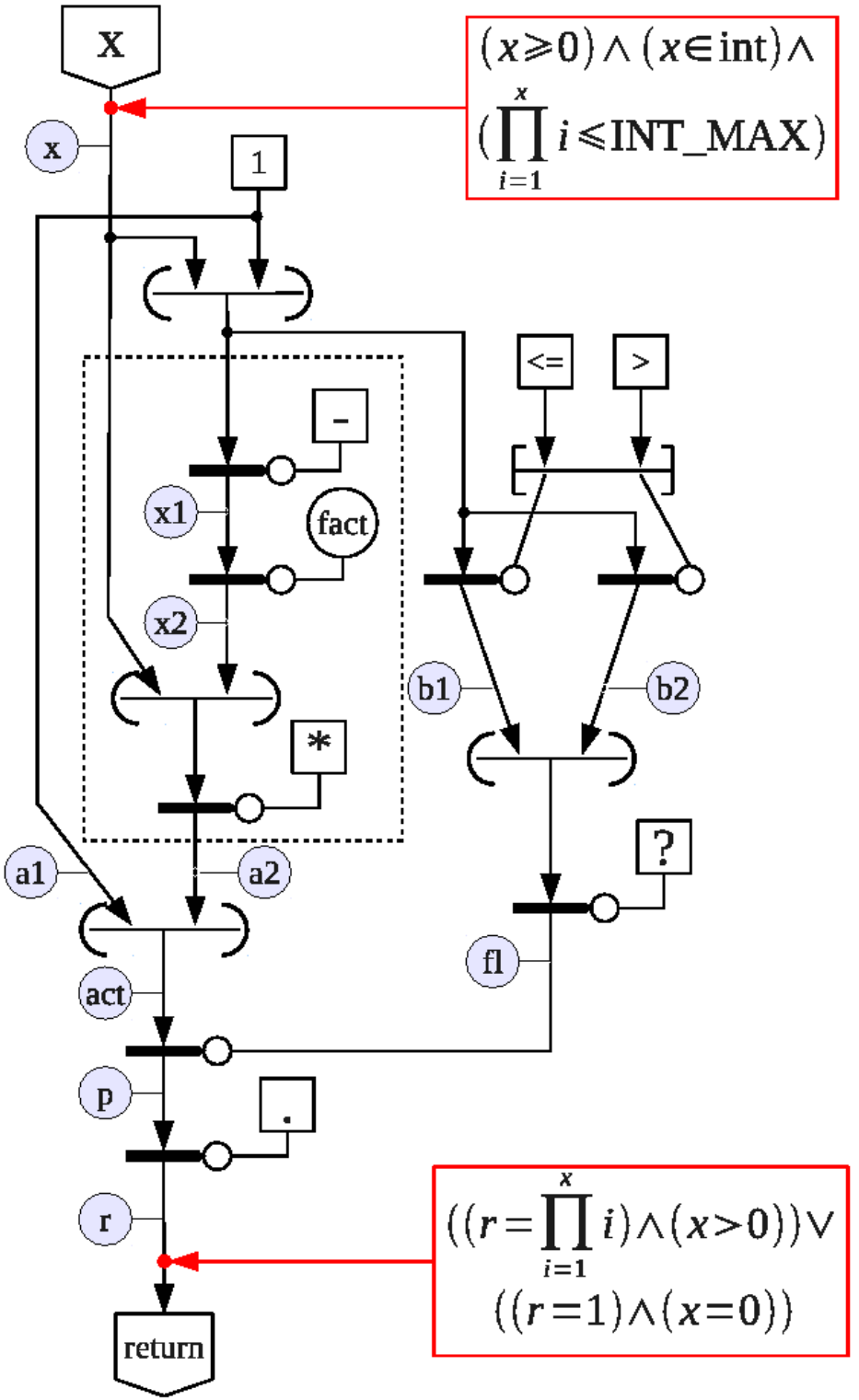
```
Fun << funcdef arg {
     arg:F >> return
}
```

$$\boxed{P(arg)}\ \texttt{arg:F}\rightarrow r\ \boxed{Q(r)}$$



```
fact << funcdef x {
  fl << ((x,1):[<=,>]):?;
  Act << (
    1,
    { (x, (x,1):-:fact ):* }
  );
  return << act:fl:.;
}
```

$$(x \in int) \wedge (x \geqslant 0) \wedge \prod_{i=1}^{x} i \leqslant \texttt{INT\_MAX} \quad \texttt{x:fact} \rightarrow r \quad ((r = \prod_{i=1}^{x} i) \wedge (x > 0)) \vee ((r = 1) \wedge (x = 0))$$


X
$(x \geqslant 0) \wedge (x \in \text{int}) \wedge$
$(\prod_{i=1}^{x} i \leqslant \text{INT\_MAX})$
x
1
<=
>
-
x1
fact
x2
*
b1
b2
?
a1
a2
fl
act
p
.
r
return
$((r = \prod_{i=1}^{x} i) \wedge (x > 0)) \vee$
$((r = 1) \wedge (x = 0))$

Благодарю за внимание!

https://bitbucket.org/alpha900i/pifagor

http://www.softcraft.ru/fppp/