Автоматизация программирования в СССР:

Часть 1. Программирующие программы (50-е годы)

Петр Советов
РТУ МИРЭА

План цикла докладов

Цель — напомнить о хорошо забытых идеях и алгоритмах, не вдаваясь в подробности биографий их авторов и детали устройства вычислительных машин.

Часть 1. Программирующие программы (50-е годы).

Часть 2. Трансляторы (60-70-е годы).

Часть 3. Метавычисления.

Часть 4. Синтез программ.

План первой части

Часть 1. Программирующие программы (50-е годы).

- **1.1. ПП-2.**
- **1.2. Схемы Канторовича.**
- **1.3. Нумерация значений.**
- **1.3. Число Ершова.**

Часть 2. Трансляторы (60-70-е годы).

Часть 3. Метавычисления.

Часть 4. Синтез программ.

Программирующая программа-2 (ПП-2) — первый оптимизирующий компилятор

Д. Кнут о первых языках и компиляторах [1]

Language	Principal author(s)	Year	Arith-metic	Implemen-tation	Read-ability	Control structures	Data structures	Machine inde-pendence	Impact	First
Plankalkül	Zuse	1945	X, S, F	F	D	A	A	B	C	Programming language, hierarchic data
Flow diagrams	Goldstine and von Neumann	1946	X, S	F	A	D	C	B	A	Accepted programming methodology
Composition	Curry	1948	X	F	D	C	D	C	F	Code generation algorithm
Short Code	Mauchly	1950	F	C	C	F	F	B	D	High-level language implemented
Intermediate PL	Burks	1950	?	F	A	D	C	A	F	Common subexpression notation
Klammerausdrücke	Rutishauser	1951	F	F	B	F	C	B	B	Simple code generation, loop expansion
Formules	Böhm	1951	X	F	B	D	C	B	D	Compiler in own language
AUTOCODE	Glennie	1952	X	C	C	C	C	D	D	Useful compiler
A-2	Hopper	1953	F	C	D	F	F	C	B	Macroexpander
Algebraic interpreter	Laning and Zierler	1953	F	B	A	D	C	A	B	Constants in formulas, manual for novices
AUTOCODE	Brooker	1954	X, F	A	B	D	C	A	C	Clean two-level storage
ПП-2	Kamynin and Lĭubimskiĭ	1954	F	B	C	D	C	B	D	Code optimization
ПП	Ershov	1955	F	B	B	C	C	B	C	Book about a compiler
BACAIC	Grems and Porter	1955	F	A	A	D	F	A	D	Use on two machines
Kompiler 2	Elsworth and Kuhn	1955	S	C	C	D	C	C	F	Scaling aids
ADES	Blum	1956	X, F	D	D	B	C	A	F	Declarative language
IT	Perlis	1956	X, F	A	B	C	C	A	B	Successful compiler
FORTRAN I	Backus	1956	X, F	A	A	C	C	A	A	I/O formats, global optimization
MATH-MATIC	Katz	1956	F	B	A	C	C	A	D	Heavy use of English
Patent 3,047,228	Bauer and Samelson	1957	F	D	B	D	C	B	C	Formula-controlled computer

[1] Knuth, Donald E., and Luis Trabb Pardo. "The early development of programming languages." *A history of computing in the twentieth century* (1980): 197-273.

Программирующая программа-2 (ПП-2)

Разработана в **1955** г.

Авторы ПП-2: С. С. Камынин, Э. З. Любимский, Э. С. Луховицкая, В. С. Штаркман.

Реализованные алгоритмы оптимизаций:

- **экономия общих подвыражений,**
- **экономия памяти для линейного участка,**
- **вычисление логических выражений по короткой схеме.** В духе упрощения && и || в C++.

ПРОБЛЕМЫ КИБЕРНЕТИКИ

ПОД РЕДАКЦИЕЙ
А. А. ЛЯПУНОВА

ВЫПУСК 1

81288.

ГОСУДАРСТВЕННОЕ ИЗДАТЕЛЬСТВО
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
МОСКВА 1958

СОДЕРЖАНИЕ

От редакции 4

I. ОБЩИЕ ВОПРОСЫ

А. А. Ляпунов. О некоторых общих вопросах кибернетики 5
М. Л. Цетлин. О непримитивных схемах 23

II. ПРОГРАММИРОВАНИЕ

А. А. Ляпунов. О логических схемах программ 46
Ю. И. Янов. О логических схемах алгоритмов 75
Р. И. Подловченко. Об основных понятиях программирования 128
С. С. Камынин, Э. З. Любимский, М. Р. Шура-Бура. Об автоматизации программирования при помощи программирующей программы 135
Э. С. Луховицкая. Блок обработки логических условий в ПП-2 . . . 172
Э. З. Любимский. Арифметический блок в ПП-2 178
С. С. Камынин. Блок переадресации в программе ПП-2 182
В. С. Штаркман. Блок экономии рабочих ячеек в ПП-2 185

III. ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ

Г. А. Михайлов, Б. Н. Шитиков, Н. А. Явлинский. Цифровая электронная машина ЦЭМ-1 190

IV. ВОПРОСЫ МАТЕМАТИЧЕСКОЙ ЛИНГВИСТИКИ

О. С. Кулагина. Об одном способе определения грамматических понятий на базе теории множеств 203
Т. Н. Молошная. Вопросы различения омонимов при машинном переводе с английского языка на русский 215
И. А. Мельчук. О машинном переводе с венгерского языка на русский 222

V. ХРОНИКА

Семинары по кибернетике в Московском университете 265
Научно-техническое совещание по кибернетике 266

Входной язык ПП-2: операторные схемы А.А. Ляпунова

Схема программы:

$$A_1 \prod_{i=1}^{50} (A_2\,(i)\; A_3 A_4 P_5 \overset{1}{\uparrow} A_6 \overset{2}{\uparrow} \overset{1}{\downarrow} A_7 \overset{2}{\downarrow} A_8 A_9\,(i))\; A_{10} \quad \text{СТОП}$$

Спецификация:

A_1 — ввод всех коэффициентов a_i, b_i, c_i $(i = 1, \ldots, 50)$.
A_2 — перенос коэффициентов a_i, b_i, c_i в ячейки a, b, c.
A_3 — вычисление дискриминанта: $t_1 = 2a$, $t_2 = 2c$, $дискр = b^2 - t_1 t_2$.
A_4 — вычисление заготовок для корней: $p = -b/t_1$, $q = \sqrt{|дискр|}/t_1$.
P_5 — проверка: $дискр < 0$?
A_6 — вычисление модуля и аргумента комплексной пары корней:
$t = p^2 + q^2$, $кор1 = \sqrt{t}$, $кор2 = \arcsin\,(q/кор1)$.
A_7 — вычисление пары действительных корней:
$кор1 = p + q$, $кор2 = p - q$.
A_8 — сохранение знака дискриминанта: $призн = \operatorname{sign}(дискр)$.
A_9 — перенос в массив решений пары корней i-го уравнения.
A_{10} — печать всех решений.

Ершов А. П. Введение в теоретическое программирование. – 1977.

Выходное представление ПП-2: код для ЭВМ “Стрела”

ЭВМ “Стрела” с точки зрения компиляторщика выглядит даже проще современных RISC-процессоров:

- **трехадресная система команд,**
- **архитектура память-память.**

“Машины БЭСМ-2 и "Стрела" не имели буквенно-цифрового ввода и вывода, и необходимость числовой кодировки буквенных текстов символических программ приводила опытных программистов к убеждению, что восьмеричное кодирование машинной программы по аккуратной блок-схеме не менее эффективно, чем числовое кодирование любой символической схемы программы” [1]

[1] Ершов А. П., Шура-Бура М. Р. Становление программирования в СССР //Кибернетика. Киев. – 1976. – №. 6. – С. 141-160.

Пример трансляции

В качестве примера рассмотрим формулу

$$Z = \frac{1 + a + b \ln x - \sqrt{(c + a + 1) \ln x} + d \cdot e^{(1 + a + b \ln x)}}{a + b \ln x}.$$

Согласно нашим правилам эту формулу следует записать следующим образом:

$$1 + a + (b \cdot \ln x) - \sqrt{((c + a + 1) \cdot \ln x)} + \\ + (d \cdot \exp(1 + a + (b \ln x))) : (a + (b \cdot \ln x)) = Z.$$

Результат трансляции:

$$1 + a = r_4, \qquad r_2 - r_3 = r_3,$$
$$\ln x = r_3, \qquad \exp r_2 = r_2,$$
$$b \cdot r_3 = r_1, \qquad d \cdot r_2 = r_2,$$
$$r_4 + r_1 = r_2, \qquad r_3 + r_2 = r_2,$$
$$c + r_4 = r_4, \qquad a + r_1 = r_1,$$
$$r_4 \cdot r_3 = r_3, \qquad r_2 : r_1 = Z.$$
$$\sqrt{r_3} = r_3,$$

“Проблемы кибернетики”. Выпуск 1.

Арифметический блок в ПП-2

Алгоритм арифметического блока **одновременно** выполняет:

- синтаксический разбор формулы,
- экономию общих подвыражений с учетом коммутативности,
- генерацию кода без экономии памяти.

Алгоритм арифметического блока (1)

1. **Найти простое выражение e в массиве.**

```
1 + a + ( b * ln x ) - sqrt ( ( c + a + 1 ) * ln x ) + ( d * exp ( 1 + a
+ ( b * ln x ) ) ) / ( a + ( b * ln x )) = Z
```

Алгоритм арифметического блока (2)

1. Найти простое выражение e в массиве.
2. **Сгенерировать приказ с записью e в новую рабочую ячейку t.**

```
1 + a + ( b * ln x ) - sqrt ( ( c + a + 1 ) * ln x ) + ( d * exp ( 1 + a
+ ( b * ln x ) ) ) / ( a + ( b * ln x )) = Z
```

```
t1 = 1 + a
```

Алгоритм арифметического блока (3)

1. Найти простое выражение e в массиве.
2. Сгенерировать приказ с записью e в новую рабочую ячейку t.
3. **Заменить все варианты вхождения e на t с учетом коммутативности.**

```
t1 + ( b * ln x ) - sqrt ( ( c + t1 ) * ln x ) + ( d * exp ( t1 + ( b *
ln x ) ) ) / ( a + ( b * ln x ) ) = Z
```

```
t1 = 1 + a
```

Алгоритм арифметического блока (4)

1. **Найти простое выражение e в массиве.**
2. Сгенерировать приказ с записью e в новую рабочую ячейку t.
3. Заменить все варианты вхождения e на t с учетом коммутативности.

```
t1 + ( b * ln x ) - sqrt ( ( c + t1 ) * ln x ) + ( d * exp ( t1 + ( b *
ln x ) ) ) / ( a + ( b * ln x ) ) = Z
```

```
t1 = 1 + a
```

Алгоритм арифметического блока (5)

1. Найти простое выражение e в массиве.
2. **Сгенерировать приказ с записью e в новую рабочую ячейку t.**
3. Заменить все варианты вхождения e на t с учетом коммутативности.

```
t1 + ( b * ln x ) - sqrt ( ( c + t1 ) * ln x ) + ( d * exp ( t1 + ( b *
ln x ) ) ) / ( a + ( b * ln x ) ) = Z
```

```
t1 = 1 + a
t2 = ln x
```

Алгоритм арифметического блока (6)

1. Найти простое выражение e в массиве.
2. Сгенерировать приказ с записью e в новую рабочую ячейку t.
3. **Заменить все варианты вхождения e на t с учетом коммутативности.**

```
t1 + ( b * t2 ) - sqrt ( ( c + t1 ) * t2 ) + ( d * exp ( t1 + ( b * t2 ) )
) ) / ( a + ( b * t2 ) ) = Z
```

```
t1 = 1 + a
t2 = ln x
```

Алгоритм арифметического блока (7)

1. **Найти простое выражение e в массиве.**
2. Сгенерировать приказ с записью e в новую рабочую ячейку t.
3. Заменить все варианты вхождения e на t с учетом коммутативности.

```
t1 + ( b * t2 ) - sqrt ( ( c + t1 ) * t2 ) + ( d * exp ( t1 + ( b * t2 )
) ) / ( a + ( b * t2 ) ) = Z
```

```
t1 = 1 + a
t2 = ln x
```

Алгоритм арифметического блока (8)

1. Найти простое выражение e в массиве.
2. **Сгенерировать приказ с записью e в новую рабочую ячейку t.**
3. Заменить все варианты вхождения e на t с учетом коммутативности.

```
t1 + ( b * t2 ) - sqrt ( ( c + t1 ) * t2 ) + ( d * exp ( t1 + ( b * t2 )
) ) / ( a + ( b * t2 ) ) = Z
```

```
t1 = 1 + a
t2 = ln x
t3 = b * t2
```

Алгоритм арифметического блока (9)

1. Найти простое выражение e в массиве.
2. Сгенерировать приказ с записью e в новую рабочую ячейку t.
3. **Заменить все варианты вхождения e на t с учетом коммутативности.**

```
t1 + ( t3 ) - sqrt ( ( c + t1 ) * t2 ) + ( d * exp ( t1 + ( t3 ) ) ) / (
a + ( t3 ) ) = Z
```

```
t1 = 1 + a
t2 = ln x
t3 = b * t2
```

Алгоритм арифметического блока (10)

1. Найти простое выражение e в массиве.
2. Сгенерировать приказ с записью e в новую рабочую ячейку t.
3. Заменить все варианты вхождения e на t с учетом коммутативности.
4. **Убрать лишние скобки.**

```
t1 + t3 - sqrt ( ( c + t1 ) * t2 ) + ( d * exp ( t1 + t3 ) ) / ( a + t3 )
= Z
```

```
t1 = 1 + a
t2 = ln x
t3 = b * t2
```

Алгоритм арифметического блока (11)

Пока массив длиннее 3 элементов:

1. Найти простое выражение e в массиве.
2. Сгенерировать приказ с записью e в новую рабочую ячейку t.
3. Заменить все варианты вхождения e на t с учетом коммутативности.
4. Убрать лишние скобки.

```
t11 / t12 = Z
```

```
t1 = 1 + a
t2 = ln x
t3 = b * t2
t4 = t1 + t3
t5 = c + t1
```

```
t6 = t5 * t2
t7 = sqrt t6
t8 = t4 - t7
t9 = exp t4
t10 = d * t9
```

```
t11 = t8 + t10
t12 = a + t3
Z = t11 / t12
```

Без экономии памяти!

Блок экономии рабочих ячеек в ПП-2

Алгоритм экономии рабочих ячеек (алгоритм Штаркмана) решает задачу **локального распределения регистров** (local register allocation).

Выгрузки регистров в память (spilling) нет (архитектура — память-память!).

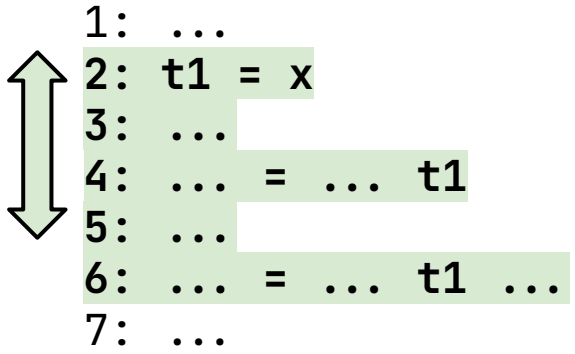
Алгоритм работает за один проход **от последнего приказа к первому**.

Схожие алгоритмы используются в некоторых современных JIT-компиляторах [1].

[1] Gal, Andreas, Christian W. Probst, and Michael Franz. "HotpathVM: An effective JIT compiler for resource-constrained devices." *Proceedings of the 2nd international conference on Virtual execution environments*. 2006.

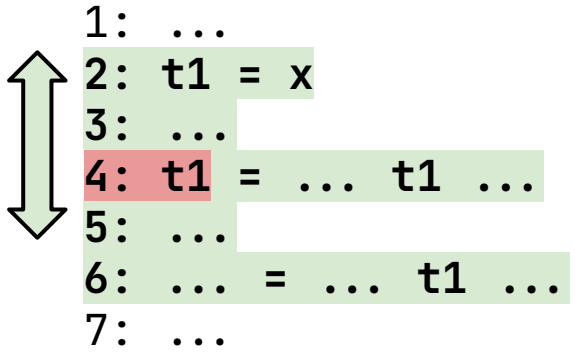
Область существования промежуточного результата (1)

Область от приказа, породившего результат и до последнего приказа (исключая его), где этот результат используется.



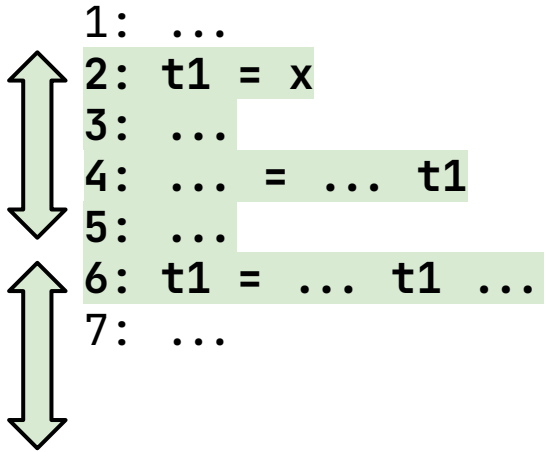
Область существования промежуточного результата (2)

Область от приказа, породившего результат и до последнего приказа (исключая его), где этот результат используется.



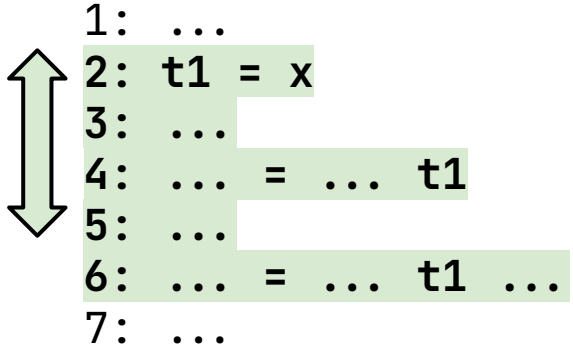
Область существования промежуточного результата (3)

Область от приказа, породившего результат и до последнего приказа (исключая его), где этот результат используется.



Область существования промежуточного результата (4)

Как найти эту область? Двигаясь от конца к началу, найти первый приказ, где результат используется. Здесь регистр, хранящий результат, уже свободен. Запись в результат значения является началом области существования.



Если области существования результатов пересекаются – им нельзя назначить один и тот же регистр!

Алгоритм экономии рабочих ячеек (1)

Пройти по массиву команд в обратном порядке:


```
...   ...

 8:  t8 = t4 - t7

 9:  t9 = exp t4

10:  t10 = d * t9

11:  t11 = t8 + t10

12:  t12 = a + t3

13:  Z = t11 / t12
```

Если аргумент отсутствует в таблице, записать его в таблицу

r1	r2	r3	r4
t11			

Алгоритм экономии рабочих ячеек (2)

Пройти по массиву команд в обратном порядке:


```
...   ...

 8:  t8 = t4 - t7

 9:  t9 = exp t4

10:  t10 = d * t9

11:  t11 = t8 + t10

12:  t12 = a + t3

13:  Z = r1 / t12
```

Если аргумент отсутствует в таблице, записать его в таблицу, **заменить аргумент в приказе номером заполненной ячейки.**

r1	r2	r3	r4
t11			

Алгоритм экономии рабочих ячеек (3)

Пройти по массиву команд в обратном порядке:

```
...   ...
 8:  t8 = t4 - t7
 9:  t9 = exp t4
10:  t10 = d * t9
11:  t11 = t8 + t10
12:  t12 = a + t3
13:  Z = r1 / t12
```

Если аргумент отсутствует в таблице, записать его в таблицу, заменить аргумент в приказе номером заполненной ячейки.

r1	r2	r3	r4
t11	**t12**		

Алгоритм экономии рабочих ячеек (4)

Пройти по массиву команд в обратном порядке:


```
...   ...

 8:  t8 = t4 - t7

 9:  t9 = exp t4

10:  t10 = d * t9

11:  t11 = t8 + t10

12:  t12 = a + t3

13:  Z = r1 / r2
```

Если аргумент отсутствует в таблице, записать его в таблицу, **заменить аргумент в приказе номером заполненной ячейки.**

r1	r2	r3	r4
t11	**t12**		

Алгоритм экономии рабочих ячеек (5)


```
...   ...

 7:  t7 = sqrt t6

 8:  t8 = t4 - t7

 9:  t9 = exp t4

10:  t10 = d * t9

11:  t11 = t8 + t10

12:  t12 = a + t3
```

Пройти по массиву команд в обратном порядке:

Записать номер ячейки результата в приказ.

Если аргумент отсутствует в таблице, записать его в таблицу, заменить аргумент в приказе номером заполненной ячейки.

r1	r2	r3	r4
t11	**t12**		

Алгоритм экономии рабочих ячеек (6)


```
...   ...

 7:  t7 = sqrt t6

 8:  t8 = t4 - t7

 9:  t9 = exp t4

10:  t10 = d * t9

11:  t11 = t8 + t10

12:  r2 = a + t3
```

Пройти по массиву команд в обратном порядке:

1. Записать номер ячейки результата в приказ.
2. **Очистить ячейку результата приказа в таблице.**
3. Если аргумент отсутствует в таблице, записать его в таблицу, заменить аргумент в приказе номером заполненной ячейки.

r1	r2	r3	r4
t11			

Алгоритм экономии рабочих ячеек (7)


```
...   ...

 7:  t7 = sqrt t6

 8:  t8 = t4 - t7

 9:  t9 = exp t4

10:  t10 = d * t9

11:  t11 = t8 + t10

12:  r2 = a + t3
```

Пройти по массиву команд в обратном порядке:

1. Записать номер ячейки результата в приказ.
2. Очистить ячейку результата приказа в таблице.
3. **Если аргумент отсутствует в таблице, записать его в таблицу**, заменить аргумент в приказе номером заполненной ячейки.

r1	r2	r3	r4
t11	**t3**		

Алгоритм экономии рабочих ячеек (8)


```
...   ...

 7:  t7 = sqrt t6

 8:  t8 = t4 - t7

 9:  t9 = exp t4

10:  t10 = d * t9

11:  t11 = t8 + t10

12:  r2 = a + r2
```

Пройти по массиву команд в обратном порядке:

1. Записать номер ячейки результата в приказ.
2. Очистить ячейку результата приказа в таблице.
3. Если аргумент отсутствует в таблице, записать его в таблицу, **заменить аргумент в приказе номером заполненной ячейки.**

r1	r2	r3	r4
t11	**t3**		

Алгоритм экономии рабочих ячеек (9)


```
...   ...

 6:  t6 = t5 * t2

 7:  t7 = sqrt t6

 8:  t8 = t4 - t7

 9:  t9 = exp t4

10:  t10 = d * t9

11:  t11 = t8 + t10
```

Пройти по массиву команд в обратном порядке:

1. **Записать номер ячейки результата в приказ.**
2. Очистить ячейку результата приказа в таблице.
3. Если аргумент отсутствует в таблице, записать его в таблицу, заменить аргумент в приказе номером заполненной ячейки.

r1	r2	r3	r4
t11	t3		

Алгоритм экономии рабочих ячеек (10)


```
...   ...

 6:  t6 = t5 * t2

 7:  t7 = sqrt t6

 8:  t8 = t4 - t7

 9:  t9 = exp t4

10:  t10 = d * t9

11:  r1 = t8 + t10
```

Пройти по массиву команд в обратном порядке:

1. Записать номер ячейки результата в приказ.
2. **Очистить ячейку результата приказа в таблице.**
3. Если аргумент отсутствует в таблице, записать его в таблицу, заменить аргумент в приказе номером заполненной ячейки.

r1	r2	r3	r4
	t3		

Алгоритм экономии рабочих ячеек (11)


```
...   ...

 6:  t6 = t5 * t2

 7:  t7 = sqrt t6

 8:  t8 = t4 - t7

 9:  t9 = exp t4

10:  t10 = d * t9

11:  r1 = t8 + t10
```

Пройти по массиву команд в обратном порядке:

1. Записать номер ячейки результата в приказ.
2. Очистить ячейку результата приказа в таблице.
3. **Если аргумент отсутствует в таблице, записать его в таблицу**, заменить аргумент в приказе номером заполненной ячейки.

r1	r2	r3	r4
t8	t3		

Алгоритм экономии рабочих ячеек (12)


```
...   ...

 6:  t6 = t5 * t2

 7:  t7 = sqrt t6

 8:  t8 = t4 - t7

 9:  t9 = exp t4

10:  t10 = d * t9

11:  r1 = r1 + t10
```

Пройти по массиву команд в обратном порядке:

1. Записать номер ячейки результата в приказ.
2. Очистить ячейку результата приказа в таблице.
3. Если аргумент отсутствует в таблице, записать его в таблицу, **заменить аргумент в приказе номером заполненной ячейки.**

r1	r2	r3	r4
t8	t3		

Алгоритм экономии рабочих ячеек (13)


```
...   ...

 6:  t6 = t5 * t2

 7:  t7 = sqrt t6

 8:  t8 = t4 - t7

 9:  t9 = exp t4

10:  t10 = d * t9

11:  r1 = r1 + t10
```

Пройти по массиву команд в обратном порядке:

1. Записать номер ячейки результата в приказ.
2. Очистить ячейку результата приказа в таблице.
3. **Если аргумент отсутствует в таблице, записать его в таблицу**, заменить аргумент в приказе номером заполненной ячейки.

r1	r2	r3	r4
t8	t3	**t10**	

Алгоритм экономии рабочих ячеек (14)


```
...   ...

 6:  t6 = t5 * t2

 7:  t7 = sqrt t6

 8:  t8 = t4 - t7

 9:  t9 = exp t4

10:  t10 = d * t9

11:  r1 = r1 + r3
```

Пройти по массиву команд в обратном порядке:

1. Записать номер ячейки результата в приказ.
2. Очистить ячейку результата приказа в таблице.
3. Если аргумент отсутствует в таблице, записать его в таблицу, **заменить аргумент в приказе номером заполненной ячейки.**

r1	r2	r3	r4
t8	t3	**t10**	

Алгоритм экономии рабочих ячеек (15)


```
...   ...

 5:  t5 = c + t1

 6:  t6 = t5 * t2

 7:  t7 = sqrt t6

 8:  t8 = t4 - t7

 9:  t9 = exp t4

10:  t10 = d * t9
```

Пройти по массиву команд в обратном порядке:

1. **Записать номер ячейки результата в приказ.**
2. Очистить ячейку результата приказа в таблице.
3. Если аргумент отсутствует в таблице, записать его в таблицу, заменить аргумент в приказе номером заполненной ячейки.

r1	r2	r3	r4
t8	t3	**t10**	

Алгоритм экономии рабочих ячеек (16)


```
...   ...

 5:  t5 = c + t1

 6:  t6 = t5 * t2

 7:  t7 = sqrt t6

 8:  t8 = t4 - t7

 9:  t9 = exp t4

10:  r3 = d * t9
```

Пройти по массиву команд в обратном порядке:

1. Записать номер ячейки результата в приказ.
2. **Очистить ячейку результата приказа в таблице.**
3. Если аргумент отсутствует в таблице, записать его в таблицу, заменить аргумент в приказе номером заполненной ячейки.

r1	r2	r3	r4
t8	t3		

Алгоритм экономии рабочих ячеек (17)


```
...   ...

 5:  t5 = c + t1

 6:  t6 = t5 * t2

 7:  t7 = sqrt t6

 8:  t8 = t4 - t7

 9:  t9 = exp t4

10:  r3 = d * t9
```

Пройти по массиву команд в обратном порядке:

1. Записать номер ячейки результата в приказ.
2. Очистить ячейку результата приказа в таблице.
3. **Если аргумент отсутствует в таблице, записать его в таблицу**, заменить аргумент в приказе номером заполненной ячейки.

r1	r2	r3	r4
t8	t3	**t9**	

Алгоритм экономии рабочих ячеек (18)


```
...   ...

 5:  t5 = c + t1

 6:  t6 = t5 * t2

 7:  t7 = sqrt t6

 8:  t8 = t4 - t7

 9:  t9 = exp t4

10:  r3 = d * r3
```

Пройти по массиву команд в обратном порядке:

1. Записать номер ячейки результата в приказ.
2. Очистить ячейку результата приказа в таблице.
3. Если аргумент отсутствует в таблице, записать его в таблицу, **заменить аргумент в приказе номером заполненной ячейки.**

r1	r2	r3	r4
t8	t3	**t9**	

Алгоритм экономии рабочих ячеек (19)


```
...   ...

 4:  t4 = t1 + t3

 5:  t5 = c + t1

 6:  t6 = t5 * t2

 7:  t7 = sqrt t6

 8:  t8 = t4 - t7

 9:  t9 = exp t4
```

Пройти по массиву команд в обратном порядке:

1. **Записать номер ячейки результата в приказ.**
2. Очистить ячейку результата приказа в таблице.
3. Если аргумент отсутствует в таблице, записать его в таблицу, заменить аргумент в приказе номером заполненной ячейки.

r1	r2	r3	r4
t8	t3	**t9**	

Алгоритм экономии рабочих ячеек (20)


```
...   ...

 4:  t4 = t1 + t3

 5:  t5 = c + t1

 6:  t6 = t5 * t2

 7:  t7 = sqrt t6

 8:  t8 = t4 - t7

 9:  r3 = exp t4
```

Пройти по массиву команд в обратном порядке:

1. Записать номер ячейки результата в приказ.
2. **Очистить ячейку результата приказа в таблице.**
3. Если аргумент отсутствует в таблице, записать его в таблицу, заменить аргумент в приказе номером заполненной ячейки.

r1	r2	r3	r4
t8	t3		

Алгоритм экономии рабочих ячеек (21)


```
...   ...

 4:  t4 = t1 + t3

 5:  t5 = c + t1

 6:  t6 = t5 * t2

 7:  t7 = sqrt t6

 8:  t8 = t4 - t7

 9:  r3 = exp t4
```

Пройти по массиву команд в обратном порядке:

1. Записать номер ячейки результата в приказ.
2. Очистить ячейку результата приказа в таблице.
3. **Если аргумент отсутствует в таблице, записать его в таблицу**, заменить аргумент в приказе номером заполненной ячейки.

r1	r2	r3	r4
t8	t3	**t4**	

Алгоритм экономии рабочих ячеек (22)


```
...   ...

 4:  t4 = t1 + t3

 5:  t5 = c + t1

 6:  t6 = t5 * t2

 7:  t7 = sqrt t6

 8:  t8 = t4 - t7

 9:  r3 = exp r3
```

Пройти по массиву команд в обратном порядке:

1. Записать номер ячейки результата в приказ.
2. Очистить ячейку результата приказа в таблице.
3. Если аргумент отсутствует в таблице, записать его в таблицу, **заменить аргумент в приказе номером заполненной ячейки.**

r1	r2	r3	r4
t8	t3	**t4**	

Алгоритм экономии рабочих ячеек (23)


```
...   ...

 3:  t3 = b * t2

 4:  t4 = t1 + t3

 5:  t5 = c + t1

 6:  t6 = t5 * t2

 7:  t7 = sqrt t6

 8:  t8 = t4 - t7
```

Пройти по массиву команд в обратном порядке:

1. **Записать номер ячейки результата в приказ.**
2. Очистить ячейку результата приказа в таблице.
3. Если аргумент отсутствует в таблице, записать его в таблицу, заменить аргумент в приказе номером заполненной ячейки.

r1	r2	r3	r4
t8	t3	t4	

Алгоритм экономии рабочих ячеек (24)


```
...   ...

 3:  t3 = b * t2

 4:  t4 = t1 + t3

 5:  t5 = c + t1

 6:  t6 = t5 * t2

 7:  t7 = sqrt t6

 8:  r1 = t4 - t7
```

Пройти по массиву команд в обратном порядке:

1. Записать номер ячейки результата в приказ.
2. **Очистить ячейку результата приказа в таблице.**
3. Если аргумент отсутствует в таблице, записать его в таблицу, заменить аргумент в приказе номером заполненной ячейки.

r1	r2	r3	r4
	t3	t4	

Алгоритм экономии рабочих ячеек (25)


```
...   ...

 3:  t3 = b * t2

 4:  t4 = t1 + t3

 5:  t5 = c + t1

 6:  t6 = t5 * t2

 7:  t7 = sqrt t6

 8:  r1 = t4 - t7
```

Пройти по массиву команд в обратном порядке:

1. Записать номер ячейки результата в приказ.
2. Очистить ячейку результата приказа в таблице.
3. Если аргумент отсутствует в таблице, записать его в таблицу, заменить аргумент в приказе номером заполненной ячейки.
4. **Если аргумент находится в таблице, заменить аргумент в приказе номером найденной ячейки.**

r1	r2	r3	r4
	t3	**t4**	

Алгоритм экономии рабочих ячеек (26)


```
1:  r1 = 1 + a

2:  r4 = ln x

3:  r2 = b * r4

4:  r3 = r1 + r2

5:  r1 = c + r1

6:  r1 = r1 * r4

7:  r1 = sqrt r1
```

Пройти по массиву команд в обратном порядке:

1. Записать номер ячейки результата в приказ.
2. Очистить ячейку результата приказа в таблице.
3. Если аргумент отсутствует в таблице, записать его в таблицу, заменить аргумент в приказе номером заполненной ячейки.
4. Если аргумент находится в таблице, заменить аргумент в приказе номером найденной ячейки.

Для программы достаточно 4 регистров!

r1	r2	r3	r4
t1			

Схемы Канторовича для преобразования программ в графовом представлении

Схемы Канторовича

Предложены Л.В. Канторовичем, исследовались им и его коллегами с **1953** года [1].

Синтаксически напоминают промежуточное представление — “четверки”.

Однократное присваивание в духе функционального программирования.

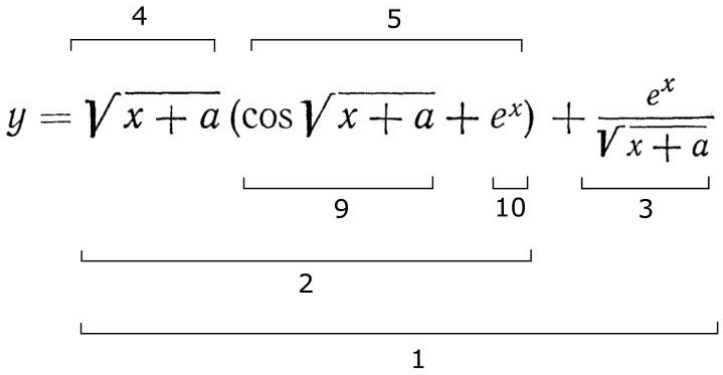
Операции над сложными типами данных: векторами, матрицами и даже самими схемами — в духе языка Лисп.

“Прорабы” (программа работ) — DSL-компиляторы и интерпретаторы для различных видов схем.

[1] Л. В. Канторович, Мой путь в науке (предполагавшийся доклад в Московском математическом обществе), УМН, 1987, том 42, выпуск 2(254), 183–213

Пример схемы Канторовича

Исходная формула:



Дерево логической подчиненности:

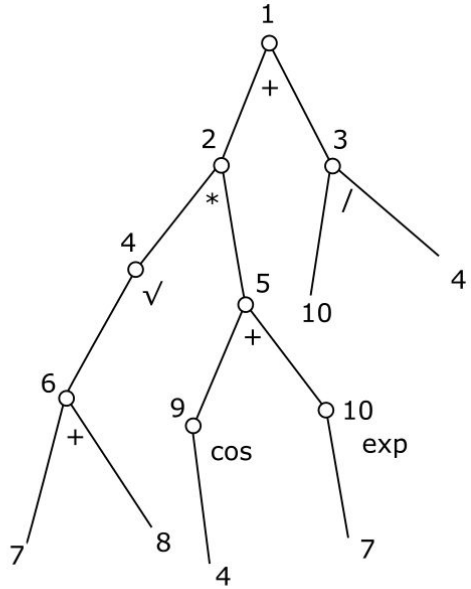
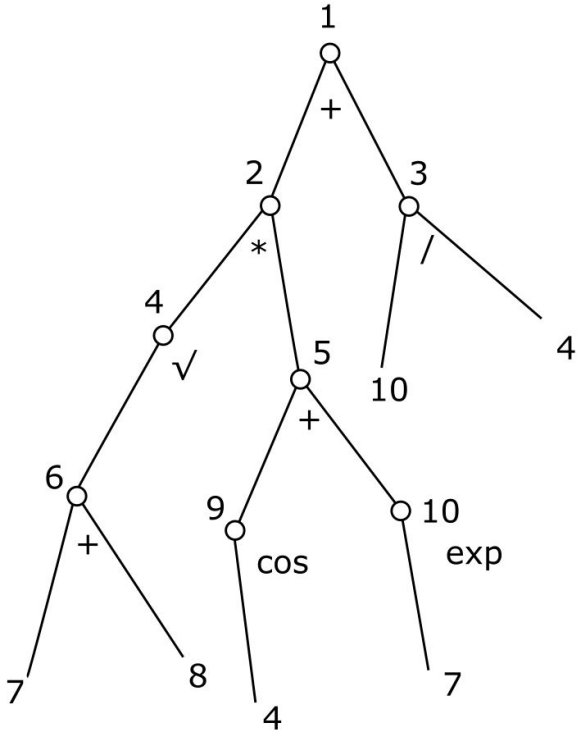


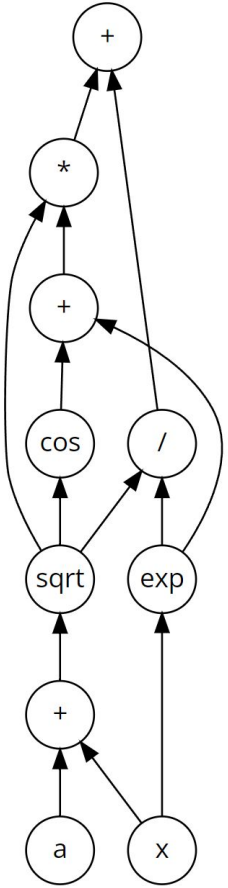
Схема Канторовича:

```
 1 = (+, 2, 3)
 3 = (/, 10, 4)
 2 = (*, 4, 5)
 5 = (+, 9, 10)
10 = (exp, 7)
 9 = (cos, 4)
 4 = (sqrt, 6)
 6 = (+, 7, 8)
 8 = a
 7 = x
```

Дерево логической подчиненности – НАГ



=



Упрощения и преобразования схем Канторовича

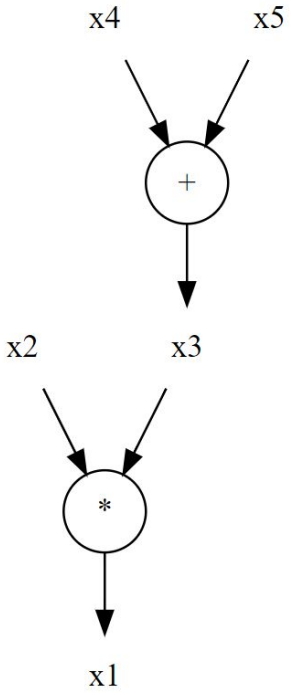
- Свертка констант.
- Упорядочение схемы с помощью топологической сортировки.
- **Замена по типовым подсхемам (rewriting), в том числе с учетом некоторого предиката (conditional rewriting).**
- Компоновка преобразований в более сложные преобразования (rewriting strategies).

Петрова Л. Т. О проведении аналитических выкладок на машинах с программным управлением //Известия высших учебных заведений. Математика. – 1958. – №. 5. – С. 95-104.

Прикладные задачи для схем Канторовича

- Оптимизации в компиляторе, основанные на правилах.
- Символьная математика (символьное дифференцирование).
- Схожий подход сегодня используется в MLIR (**DAG-to-DAG Rewriting**) и DSL-компиляторах для нейросетевых вычислений.

Пример правила замены по типовой подсхеме

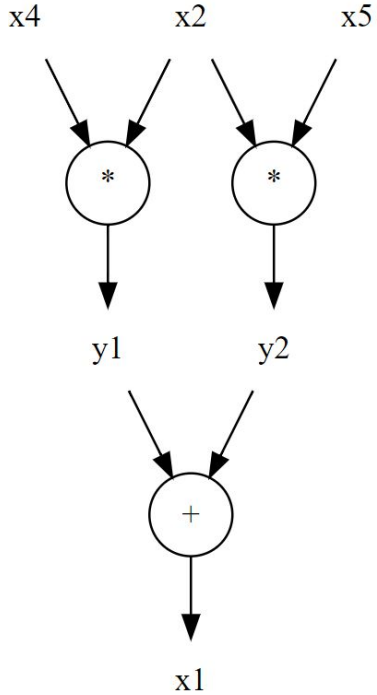


$$a(b + c) \rightarrow ab + ac$$

A:
```
x1 = (*, x2, x3)
x3 = (+, x4, x5)
```

→

B:
```
x1 = (+, y1, y2)
y1 = (*, x2, x4)
y2 = (*, x2, x5)
```



Метапеременные $x_1,\ldots,x_n$ получают значения при сопоставлении со строками схемы.

Метапеременные $y_1,\ldots,y_n$ получают уникальные номера в заменяемой схеме.

Алгоритм замены по типовой подсхеме (1)


```
S:
1 = (*, 2, 3)
3 = (+, 4, 5)
5 = c
4 = b
2 = a
```

```
A:
x1 = (*, x2, x3)
x3 = (+, x4, x5)
```

```
B:
x1 = (+, y1, y2)
y1 = (*, x2, x4)
y2 = (*, x2, x5)
```

```
mvars:
{}
```

S’:

Найти в S строку, которой можно сопоставить строку из A.

Алгоритм замены по типовой подсхеме (2)


```
S:
1 = (*, 2, 3)
3 = (+, 4, 5)
5 = c
4 = b
2 = a
```

```
A:
x1 = (*, x2, x3)
x3 = (+, x4, x5)
```

```
B:
x1 = (+, y1, y2)
y1 = (*, x2, x4)
y2 = (*, x2, x5)
```

```
mvars:
{x1: 1, x2: 2, x3: 3}
```

```
S':
```

Найти в S строку, которой можно сопоставить строку из A.

Занести значения x в mvars.

Алгоритм замены по типовой подсхеме (3)


```
S:
1 = (*, 2, 3)
3 = (+, 4, 5)
5 = c
4 = b
2 = a
```

```
A:
x1 = (*, x2, x3)
x3 = (+, x4, x5)
```

```
B:
x1 = (+, y1, y2)
y1 = (*, x2, x4)
y2 = (*, x2, x5)
```

```
mvars:
{x1: 1, x2: 2, x3: 3}
```

```
S':
```

Найти в S строку, которой можно сопоставить строку из A.

Занести значения x в mvars.

Алгоритм замены по типовой подсхеме (4)


```
S:
1 = (*, 2, 3)
3 = (+, 4, 5)
5 = c
4 = b
2 = a
```

```
A:
x1 = (*, x2, x3)
x3 = (+, x4, x5)
```

```
B:
x1 = (+, y1, y2)
y1 = (*, x2, x4)
y2 = (*, x2, x5)
```

```
mvars:
{x1: 1, x2: 2, x3: 3, x4: 4, x5: 5}
```

```
S':
```

Найти в S строку, которой можно сопоставить строку из A.

Занести значения x в mvars.

Алгоритм замены по типовой подсхеме (5)


```
S:
1 = (*, 2, 3)
3 = (+, 4, 5)
5 = c
4 = b
2 = a
```

```
A:
x1 = (*, x2, x3)
x3 = (+, x4, x5)
```

```
B:
x1 = (+, y1, y2)
y1 = (*, x2, x4)
y2 = (*, x2, x5)
```

```
mvars:
{x1: 1, x2: 2, x3: 3, x4: 4, x5: 5}
```

```
S':
1 = (+, y1, y2)
y1 = (*, 2, 4)
y2 = (*, 2, 5)
```

1. Пока не все строки из A сопоставлены:
    1.1. Найти в S строку, которой можно сопоставить строку из A.
    1.2. Занести значения x в mvars.
2. **Записать в S' строки из B с заменой x их значениями.**

Алгоритм замены по типовой подсхеме (6)


```
S:
1 = (*, 2, 3)
3 = (+, 4, 5)
5 = c
4 = b
2 = a
```

```
A:
x1 = (*, x2, x3)
x3 = (+, x4, x5)
```

```
B:
x1 = (+, y1, y2)
y1 = (*, x2, x4)
y2 = (*, x2, x5)
```

```
mvars:
{x1: 1, x2: 2, x3: 3, x4: 4, x5: 5}
```

```
S':
1 = (+, 6, 7)
6 = (*, 2, 4)
7 = (*, 2, 5)
```

1. Пока не все строки из A сопоставлены:
    1.1. Найти в S строку, которой можно сопоставить строку из A.
    1.2. Занести значения x в mvars.
2. Записать в S' строки из B с заменой x их значениями.
3. **Задать для y уникальные номера.**

Алгоритм замены по типовой подсхеме (7)


```
S:
1 = (*, 2, 3)
3 = (+, 4, 5)
5 = c
4 = b
2 = a
```

```
A:
x1 = (*, x2, x3)
x3 = (+, x4, x5)
```

```
B:
x1 = (+, y1, y2)
y1 = (*, x2, x4)
y2 = (*, x2, x5)
```

```
mvars:
{x1: 1, x2: 2, x3: 3, x4: 4, x5: 5}
```

```
S':
1 = (+, 6, 7)
6 = (*, 2, 4)
7 = (*, 2, 5)
```

1. Пока не все строки из A сопоставлены:
    1.1. Найти в S строку, которой можно сопоставить строку из A.
    1.2. Занести значения x в mvars.
2. Записать в S' строки из B с заменой x их значениями.
3. Задать для y уникальные номера.
4. **Удалить из S перевычисленные в S' строки.**

Алгоритм замены по типовой подсхеме (8)


```
S:
3 = (+, 4, 5)
5 = c
4 = b
2 = a
1 = (+, 6, 7)
6 = (*, 2, 4)
7 = (*, 2, 5)
```

```
A:
x1 = (*, x2, x3)
x3 = (+, x4, x5)
```

```
B:
x1 = (+, y1, y2)
y1 = (*, x2, x4)
y2 = (*, x2, x5)
```

```
mvars:
{x1: 1, x2: 2, x3: 3, x4: 4, x5: 5}
```

```
S':
1 = (+, 6, 7)
6 = (*, 2, 4)
7 = (*, 2, 5)
```

1. Пока не все строки из A сопоставлены:
    1.1. Найти в S строку, которой можно сопоставить строку из A.
    1.2. Занести значения x в mvars.
2. Записать в S' строки из B с заменой x их значениями.
3. Задать для y уникальные номера.
4. Удалить из S перевычисленные в S' строки.
5. **Добавить к S строки S' и удалить мертвый код.**

Алгоритм замены по типовой подсхеме (9)


```
S:
5 = c
4 = b
2 = a
1 = (+, 6, 7)
6 = (*, 2, 4)
7 = (*, 2, 5)
```

```
A:
x1 = (*, x2, x3)
x3 = (+, x4, x5)
```

```
B:
x1 = (+, y1, y2)
y1 = (*, x2, x4)
y2 = (*, x2, x5)
```

```
mvars:
{x1: 1, x2: 2, x3: 3, x4: 4, x5: 5}
```

```
S':
1 = (+, 6, 7)
6 = (*, 2, 4)
7 = (*, 2, 5)
```

Можно ли ускорить этот алгоритм?

1. Пока не все строки из A сопоставлены:
    1.1. Найти в S строку, которой можно сопоставить строку из A.
    1.2. Занести значения x в mvars.
2. Записать в S' строки из B с заменой x их значениями.
3. Задать для y уникальные номера.
4. Удалить из S перевычисленные в S' строки.
5. Добавить к S строки S' и удалить мертвый код.

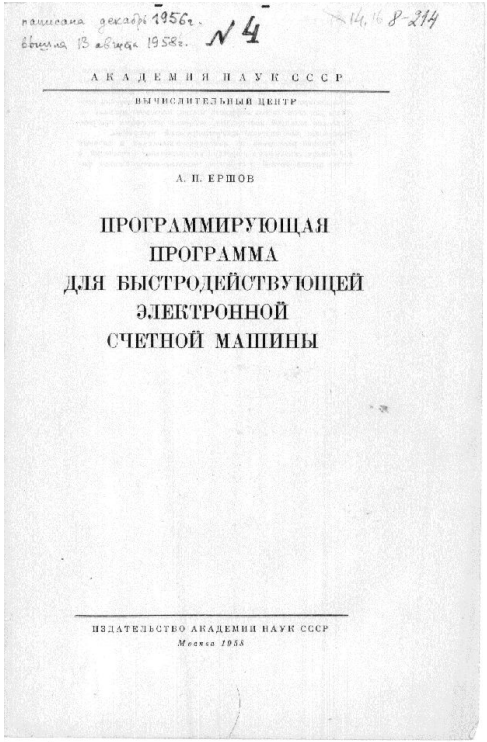
Первая книга о построении компиляторов

Ершов А. П. *Программирующая программа для быстродействующей электронной счетной машины.* – 1958. 116 с.

Перевод: Ershov A. P. *Programming programme for the BESM computer.* – Pergamon Press, 1959. 158 p.

Полное описание реализации программирующей программы для БЭСМ (ПП-БЭСМ).

В 2020-м году французский аспирант Xavier Denis создал реализацию ПП-БЭСМ (включая симулятор) на Haskell: https://github.com/xldenis/besm



О программировании арифметических операторов

Ершов А. П. *О программировании арифметических операторов* //Доклады Академии наук. – Российская академия наук, **1958**. – Т. 118. – №. 3. – С. 427-430.

В статье на 4 страницах впервые предложены:

1. **Нумерация значений** (value numbering).
2. **Число Ершова** (Ershov number).

В оригинальном варианте нумерация значений и число Ершова работают **совместно**.

О программировании арифметических операторов (перевод)

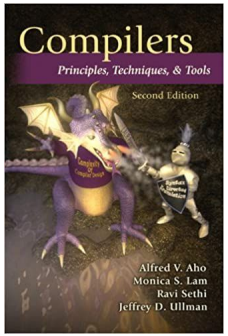
Английский перевод статьи – **самая известная в мире** советская работа по теории компиляции.

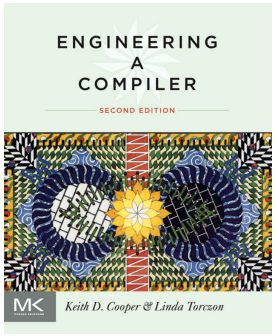
[PDF] On **programming** of **arithmetic operations** [PDF] acm.org

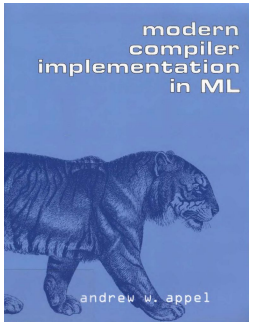
AP Ershov - Communications of the ACM, 1958 - dl.acm.org

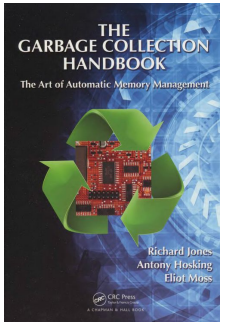
… **Programming** algorithms of **arithmetic operations** (AO) consist of three parts. The first part A1 successively generates the commands of the AO program. The second part A2 …

☆ Сохранить 99 Цитировать Цитируется: 158 Похожие статьи Все версии статьи (4)









Нумерация значений — одна из важнейших техник в современных компиляторах

О нумерации значений

Позволяет решать множество задач в компиляторе:

- Экономия общих подвыражений.
- Свертка и распространение констант.
- Упрощение выражений.
- Удаление мертвого кода.
- Построение графового IR.
- Построение формы SSA.
- …

} Эти задачи решаются одновременно

Особенности нумерации значений

В процессе нумерации значений каждое подвыражение программы заносится в **хеш-таблицу** и получает свой номер (уникальный, если ключ в таблице не найден).

Поле результата в команде не требуется, подвыражения связаны с помощью номеров в хеш-таблице.

Нумерация значений позволяет проверить наличие общего подвыражения за O(1), что ускоряет процесс экономии общих подвыражений.

Алгоритм нумерации значений (1)


```
a = b + c
b = a - d
c = b + c
e = a
d = e - d
```

```
dag = {}

vars = {}
```

Алгоритм нумерации значений (2)


```
a = b + c
b = a - d
c = b + c
e = a
d = e - d
```

```
dag = {b: 0}

vars = {}
```

b

Для каждого аргумента a получить VN(a).

VN(x):

dag[x] = уникальный номер.
Вернуть dag[x].

Алгоритм нумерации значений (3)


```
a = b + c
b = a - d
c = b + c
e = a
d = e - d
```

```
dag = {b: 0, c: 1}

vars = {}
```

b c

Для каждого аргумента a получить VN(a).

VN(x):

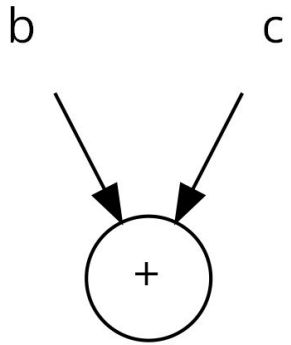
dag[x] = уникальный номер.
Вернуть dag[x].

Алгоритм нумерации значений (4)


```
a = b + c
b = a - d
c = b + c
e = a
d = e - d
```

```
dag = {b: 0, c: 1, (+, 0, 1): 2}

vars = {}
```



Для каждого аргумента a получить VN(a).

Для e = [оп, VN(a_1), …, VN(a_n)] получить VN(e).

VN(x):

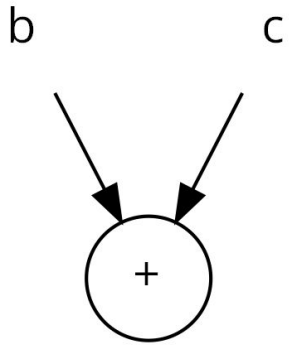
dag[x] = уникальный номер.
Вернуть dag[x].

Алгоритм нумерации значений (5)


```
a = b + c
b = a - d
c = b + c
e = a
d = e - d
```

```
dag = {b: 0, c: 1, (+, 0, 1): 2}

vars = {a: 2}
```



Для каждого аргумента a получить VN(a).
Для e = [оп, VN(a_1), …, VN(a_n)] получить VN(e).
vars[r] = VN(e), где r – имя результата команды.

VN(x):

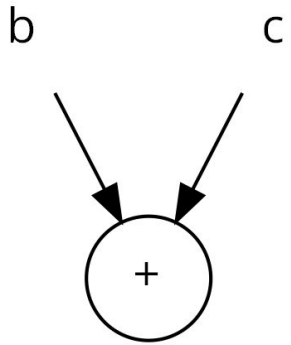
dag[x] = уникальный номер.
Вернуть dag[x].

Алгоритм нумерации значений (6)


```
a = b + c
b = a - d
c = b + c
e = a
d = e - d
```

```
dag = {b: 0, c: 1, (+, 0, 1): 2}

vars = {a: 2}
```



Для каждой команды:

1. **Для каждого аргумента a получить VN(a).**
2. Для e = [оп, VN(a_1), …, VN(a_n)] получить VN(e).
3. vars[r] = VN(e), где r — имя результата команды.

VN(x):

Если x в vars, вернуть vars[x].

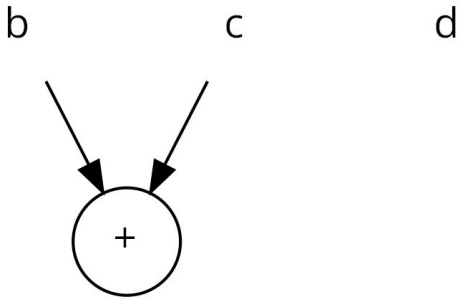
dag[x] = уникальный номер.
Вернуть dag[x].

Алгоритм нумерации значений (7)


```
a = b + c
b = a - d
c = b + c
e = a
d = e - d
```

```
dag = {b: 0, c: 1, (+, 0, 1): 2,
       d: 3}

vars = {a: 2}
```



Для каждой команды:

1. **Для каждого аргумента a получить VN(a).**
2. Для e = [оп, VN(a_1), …, VN(a_n)] получить VN(e).
3. vars[r] = VN(e), где r — имя результата команды.

VN(x):

Если x в vars, вернуть vars[x].

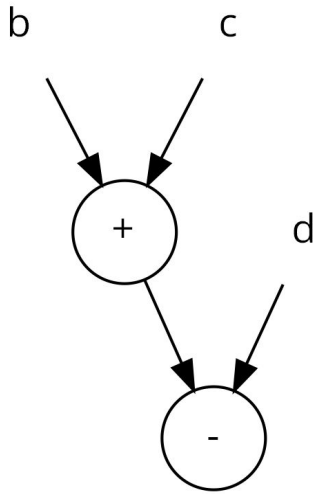
dag[x] = уникальный номер.
Вернуть dag[x].

Алгоритм нумерации значений (8)


```
a = b + c
b = a - d
c = b + c
e = a
d = e - d
```

```
dag = {b: 0, c: 1, (+, 0, 1): 2, d: 3,
       (-, 2, 3): 4}

vars = {a: 2}
```



Для каждой команды:

1. Для каждого аргумента a получить VN(a).
2. **Для e = [оп, VN(a_1), …, VN(a_n)] получить VN(e).**
3. vars[r] = VN(e), где r – имя результата команды.

VN(x):

Если x в vars, вернуть vars[x].

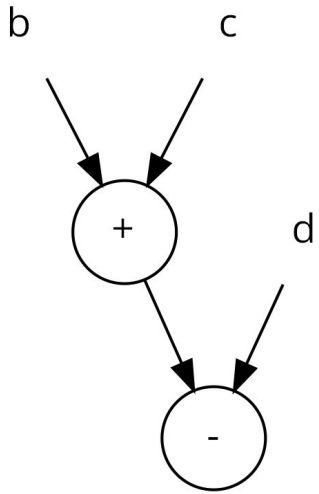
dag[x] = уникальный номер.
Вернуть dag[x].

Алгоритм нумерации значений (9)


```
a = b + c
b = a - d
c = b + c
e = a
d = e - d
```

```
dag = {b: 0, c: 1, (+, 0, 1): 2, d: 3,
       (-, 2, 3): 4}

vars = {a: 2, b: 4}
```



Для каждой команды:

1. Для каждого аргумента a получить VN(a).
2. Для e = [оп, VN(a_1), …, VN(a_n)] получить VN(e).
3. **vars[r] = VN(e), где r – имя результата команды.**

VN(x):

Если x в vars, вернуть vars[x].

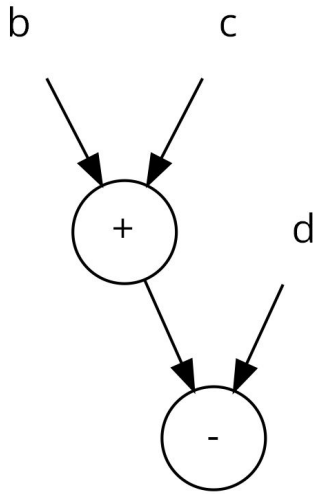
dag[x] = уникальный номер.
Вернуть dag[x].

Алгоритм нумерации значений (10)


```
a = b + c
b = a - d
c = b + c
e = a
d = e - d
```

```
dag = {b: 0, c: 1, (+, 0, 1): 2, d: 3,
       (-, 2, 3): 4}

vars = {a: 2, b: 4}
```



Для каждой команды:

1. **Для каждого аргумента a получить VN(a).**
2. Для e = [оп, VN(a_1), …, VN(a_n)] получить VN(e).
3. vars[r] = VN(e), где r — имя результата команды.

VN(x):

Если x в vars, вернуть vars[x].

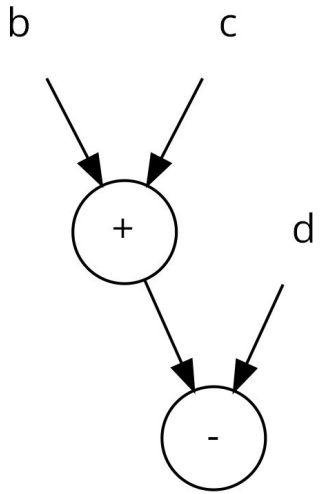
dag[x] = уникальный номер.
Вернуть dag[x].

Алгоритм нумерации значений (11)


```
a = b + c
b = a - d
c = b + c
e = a
d = e - d
```

```
dag = {b: 0, c: 1, (+, 0, 1): 2, d: 3,
       (-, 2, 3): 4}

vars = {a: 2, b: 4}
```



Для каждой команды:

1. **Для каждого аргумента a получить VN(a).**
2. Для e = [оп, VN(a_1), …, VN(a_n)] получить VN(e).
3. vars[r] = VN(e), где r — имя результата команды.

VN(x):

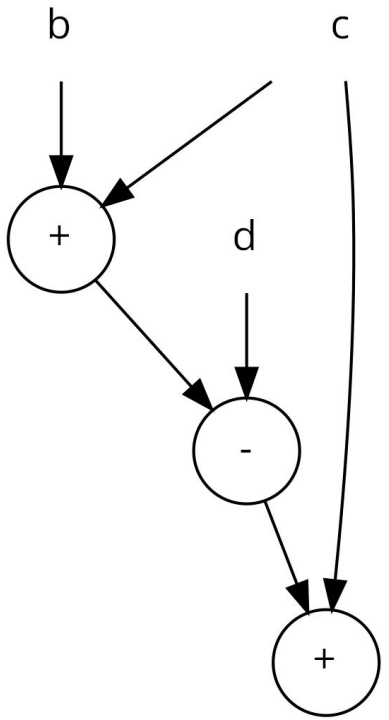
1. Если x в vars, вернуть vars[x].
2. **Если x в dag, вернуть dag[x].**
3. dag[x] = уникальный номер.
4. Вернуть dag[x].

Алгоритм нумерации значений (2)


```
a = b + c
b = a - d
c = b + c
e = a
d = e - d
```

```
dag = {b: 0, c: 1, (+, 0, 1): 2, d: 3,
       (-, 2, 3): 4, (+, 4, 1): 5}

vars = {a: 2, b: 4}
```



Для каждой команды:

1. Для каждого аргумента a получить VN(a).
2. **Для e = [оп, VN(a_1), …, VN(a_n)] получить VN(e).**
3. vars[r] = VN(e), где r — имя результата команды.

VN(x):

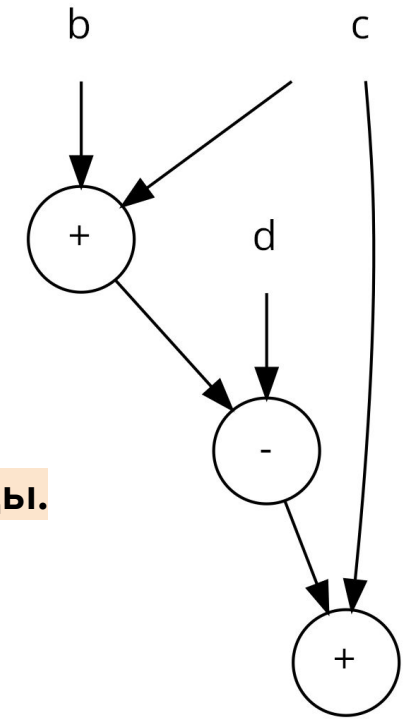
1. Если x в vars, вернуть vars[x].
2. Если x в dag, вернуть dag[x].
3. **dag[x] = уникальный номер.**
4. **Вернуть dag[x].**

Алгоритм нумерации значений (13)


```
a = b + c
b = a - d
c = b + c
e = a
d = e - d
```

```
dag = {b: 0, c: 1, (+, 0, 1): 2, d: 3,
       (-, 2, 3): 4, (+, 4, 1): 5}

vars = {a: 2, b: 4, c: 5}
```



Для каждой команды:

1. Для каждого аргумента a получить VN(a).
2. Для e = [оп, VN(a_1), …, VN(a_n)] получить VN(e).
3. **vars[r] = VN(e), где r – имя результата команды.**

VN(x):

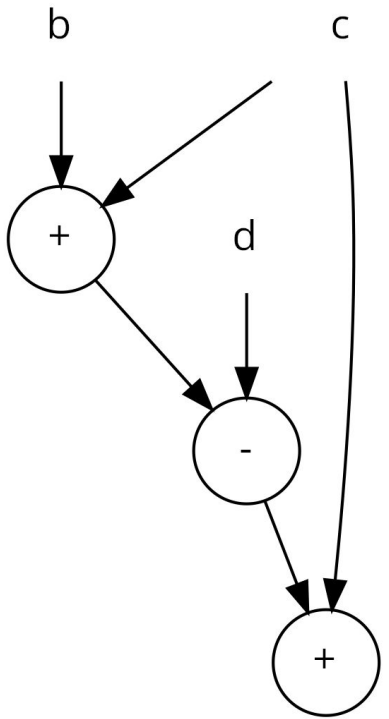
1. Если x в vars, вернуть vars[x].
2. Если x в dag, вернуть dag[x].
3. dag[x] = уникальный номер.
4. Вернуть dag[x].

Алгоритм нумерации значений (14)


```
a = b + c
b = a - d
c = b + c
e = a
d = e - d
```

```
dag = {b: 0, c: 1, (+, 0, 1): 2, d: 3,
       (-, 2, 3): 4, (+, 4, 1): 5}

vars = {a: 2, b: 4, c: 5}
```



Для каждой команды:

1. **Для каждого аргумента a получить VN(a).**
2. Для e = [оп, VN(a_1), …, VN(a_n)] получить VN(e).
3. vars[r] = VN(e), где r — имя результата команды.

VN(x):

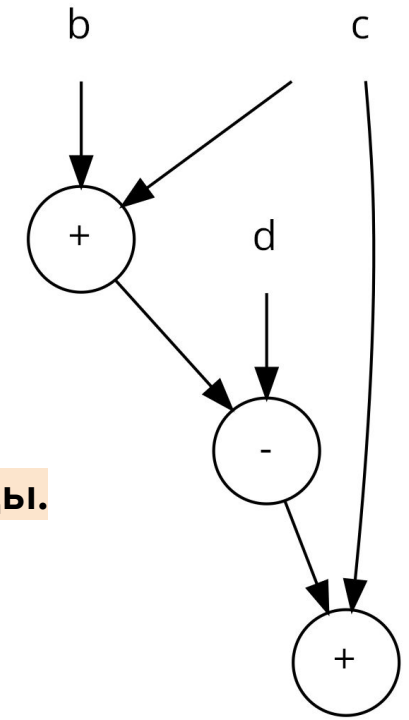
1. **Если x в vars, вернуть vars[x].**
2. Если x в dag, вернуть dag[x].
3. dag[x] = уникальный номер.
4. Вернуть dag[x].

Алгоритм нумерации значений (15)


```
a = b + c
b = a - d
c = b + c
e = a
d = e - d
```

```
dag = {b: 0, c: 1, (+, 0, 1): 2, d: 3,
       (-, 2, 3): 4, (+, 4, 1): 5}

vars = {a: 2, b: 4, c: 5, e: 2}
```



Для каждой команды:

1. Для каждого аргумента a получить VN(a).
2. Для e = [оп, VN(a_1), …, VN(a_n)] получить VN(e).
3. **vars[r] = VN(e), где r – имя результата команды.**

VN(x):

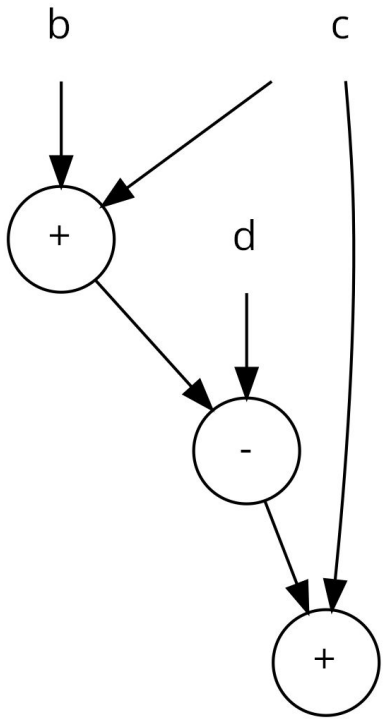
1. Если x в vars, вернуть vars[x].
2. Если x в dag, вернуть dag[x].
3. dag[x] = уникальный номер.
4. Вернуть dag[x].

Алгоритм нумерации значений (16)


```
a = b + c
b = a - d
c = b + c
e = a
d = e - d
```

```
dag = {b: 0, c: 1, (+, 0, 1): 2, d: 3,
       (-, 2, 3): 4, (+, 4, 1): 5}

vars = {a: 2, b: 4, c: 5, e: 2}
```



Для каждой команды:

1. **Для каждого аргумента a получить VN(a).**
2. Для e = [оп, VN(a_1), …, VN(a_n)] получить VN(e).
3. vars[r] = VN(e), где r — имя результата команды.

VN(x):

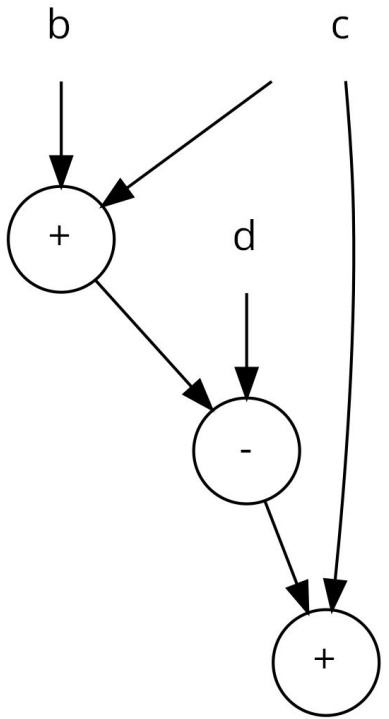
1. **Если x в vars, вернуть vars[x].**
2. Если x в dag, вернуть dag[x].
3. dag[x] = уникальный номер.
4. Вернуть dag[x].

Алгоритм нумерации значений (17)


```
a = b + c
b = a - d
c = b + c
e = a
d = e - d
```

```
dag = {b: 0, c: 1, (+, 0, 1): 2, d: 3,
       (-, 2, 3): 4, (+, 4, 1): 5}

vars = {a: 2, b: 4, c: 5, e: 2}
```



Для каждой команды:

1. **Для каждого аргумента a получить VN(a).**
2. Для e = [оп, VN(a_1), …, VN(a_n)] получить VN(e).
3. vars[r] = VN(e), где r — имя результата команды.

VN(x):

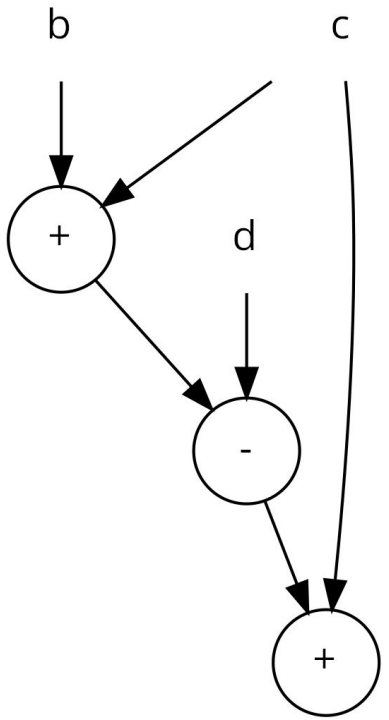
1. Если x в vars, вернуть vars[x].
2. **Если x в dag, вернуть dag[x].**
3. dag[x] = уникальный номер.
4. Вернуть dag[x].

Алгоритм нумерации значений (18)


```
a = b + c
b = a - d
c = b + c
e = a
d = e - d
```

```
dag = {b: 0, c: 1, (+, 0, 1): 2, d: 3,
       (-, 2, 3): 4, (+, 4, 1): 5}

vars = {a: 2, b: 4, c: 5, e: 2}
```



Для каждой команды:

1. Для каждого аргумента a получить VN(a).
2. **Для e = [оп, VN(a_1), …, VN(a_n)] получить VN(e).**
3. vars[r] = VN(e), где r — имя результата команды.

VN(x):

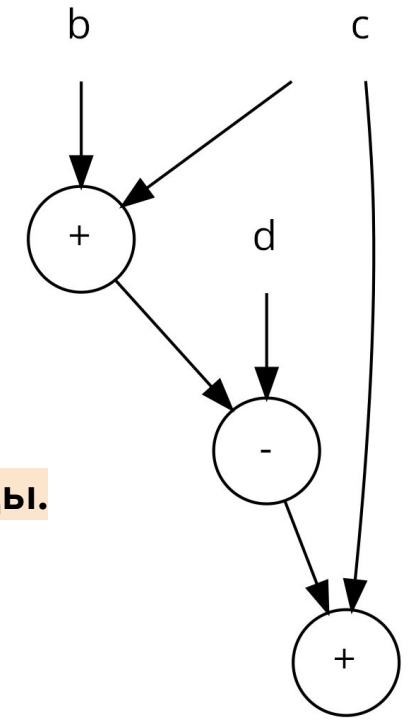
1. Если x в vars, вернуть vars[x].
2. **Если x в dag, вернуть dag[x].**
3. dag[x] = уникальный номер.
4. Вернуть dag[x].

Алгоритм нумерации значений (19)


```
a = b + c
b = a - d
c = b + c
e = a
d = e - d
```

```
dag = {b: 0, c: 1, (+, 0, 1): 2, d: 3,
       (-, 2, 3): 4, (+, 4, 1): 5}

vars = {a: 2, b: 4, c: 5, e: 2, d: 4}
```



Для каждой команды:

1. Для каждого аргумента a получить VN(a).
2. Для e = [оп, VN(a_1), …, VN(a_n)] получить VN(e).
3. **vars[r] = VN(e), где r – имя результата команды.**

VN(x):

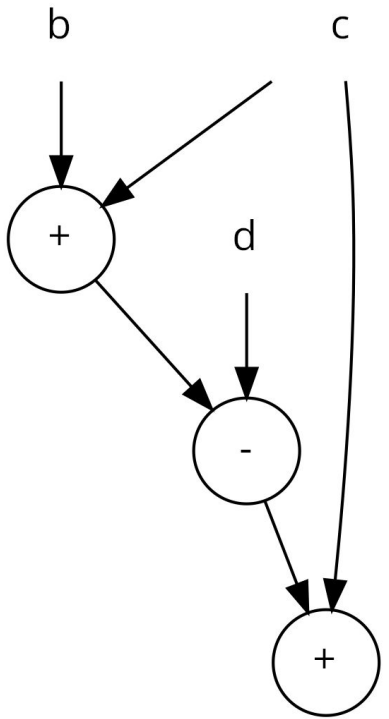
1. Если x в vars, вернуть vars[x].
2. Если x в dag, вернуть dag[x].
3. dag[x] = уникальный номер.
4. Вернуть dag[x].

Алгоритм нумерации значений (20)


```
a = b + c
b = a - d
c = b + c
e = a
d = e - d
```

```
dag = {b: 0, c: 1, (+, 0, 1): 2, d: 3,
       (-, 2, 3): 4, (+, 4, 1): 5}

vars = {a: 2, b: 4, c: 5, e: 2, d: 4}
```



Для каждой команды:

1. Для каждого аргумента a получить VN(a).
2. Для e = [оп, VN(a_1), …, VN(a_n)] получить VN(e).
3. vars[r] = VN(e), где r — имя результата команды.

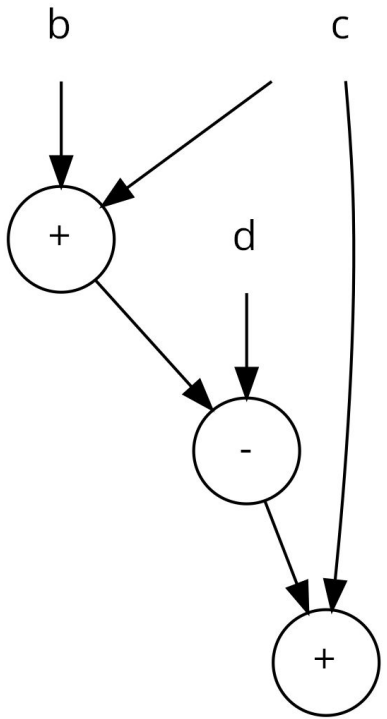
VN(x):

1. Если x в vars, вернуть vars[x].
2. Если x в dag, вернуть dag[x].
3. dag[x] = уникальный номер.
4. Вернуть dag[x].

Алгоритм нумерации значений (21)


```
a = b + c
b = a - d
c = b + c
e = a
d = e - d
```

```
dag = {b: 0, c: 1, (+, 0, 1): 2, d: 3,
       (-, 2, 3): 4, (+, 4, 1): 5}
```

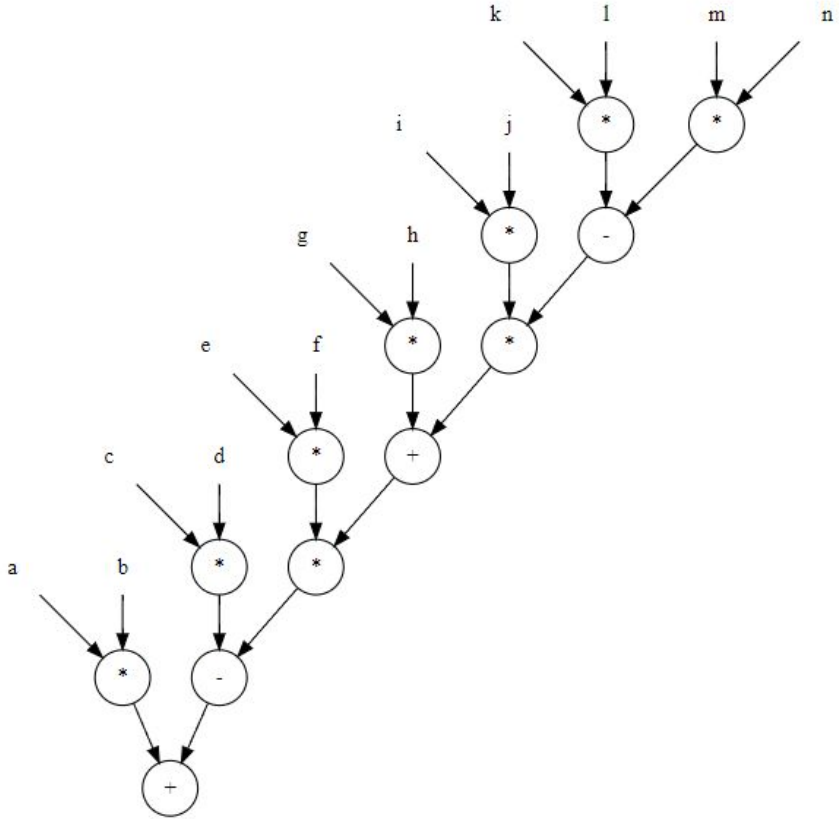


После упорядочивания:

```
t2 = b + c
t4 = t2 - d
t5 = t4 + c
```

Алгоритм на основе числа Ершова для планирования команд на линейном участке

Примеры упорядочивания НАГ



Исходный НАГ

```
t0 = a * b
t2 = e * f
t3 = g * h
t4 = i * j
t6 = m * n
t5 = k * l
t7 = t5 - t6
t8 = t7 * t4
t9 = t8 + t3
t10 = t9 * t2
t1 = c * d
t11 = t1 - t10
t12 = t11 + t0
```

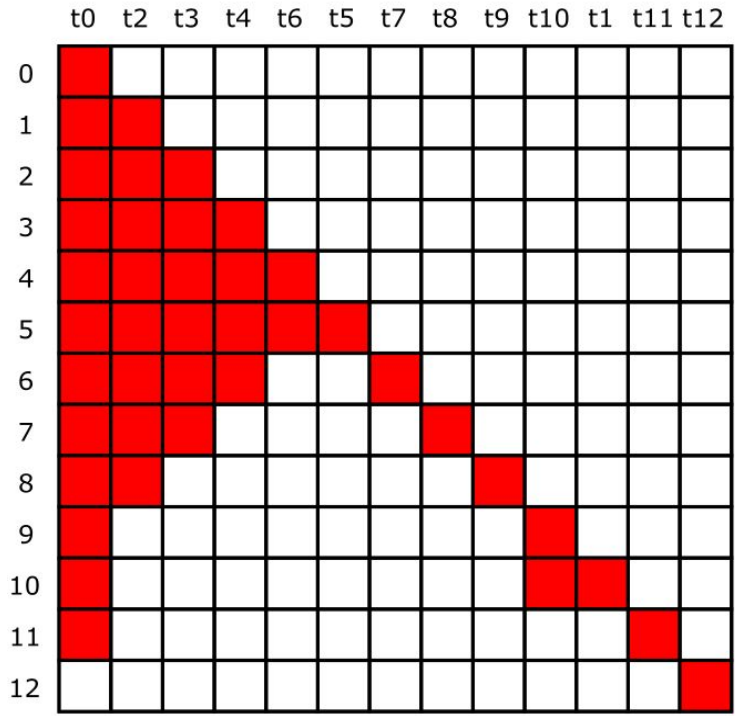
Вариант 1

```
t6 = m * n
t5 = k * l
t7 = t5 - t6
t4 = i * j
t8 = t7 * t4
t3 = g * h
t9 = t8 + t3
t2 = e * f
t10 = t9 * t2
t1 = c * d
t11 = t1 - t10
t0 = a * b
t12 = t11 + t0
```

Вариант 2

Вариант упорядочивания №1

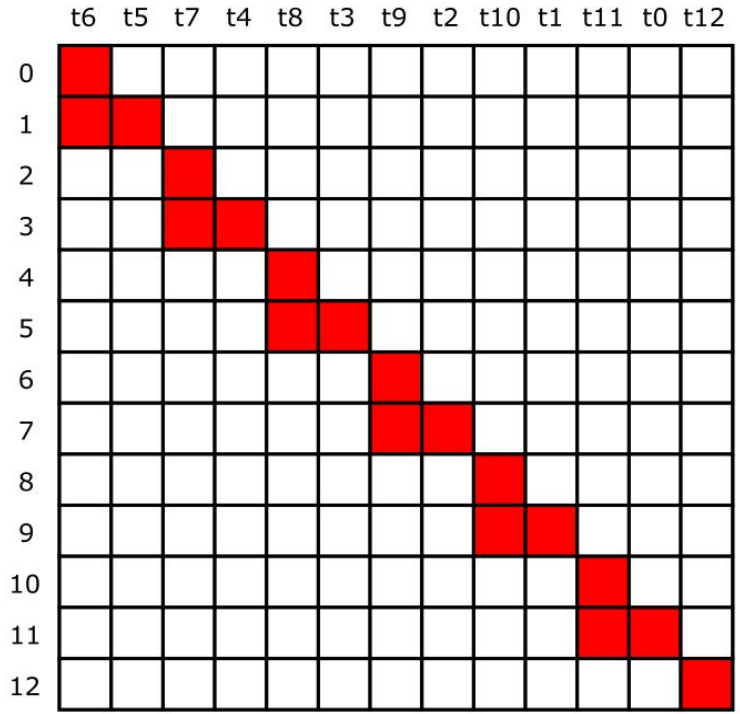
```
 0: t0 = a * b
 1: t2 = e * f
 2: t3 = g * h
 3: t4 = i * j
 4: t6 = m * n
 5: t5 = k * l
 6: t7 = t5 - t6
 7: t8 = t7 * t4
 8: t9 = t8 + t3
 9: t10 = t9 * t2
10: t1 = c * d
11: t11 = t1 - t10
12: t12 = t11 + t0
```



Требуется **6** регистров

Вариант упорядочивания №2

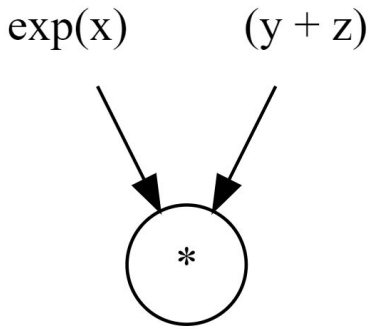

```
 0: t6 = m * n
 1: t5 = k * l
 2: t7 = t5 - t6
 3: t4 = i * j
 4: t8 = t7 * t4
 5: t3 = g * h
 6: t9 = t8 + t3
 7: t2 = e * f
 8: t10 = t9 * t2
 9: t1 = c * d
10: t11 = t1 - t10
11: t0 = a * b
12: t12 = t11 + t0
```



Требуется **2** регистра

Общая идея

Пример трансляции `exp(x) * (y + z)`.



`exp(x) * (y + z)`:

```
r1 = x
r1 = exp(r1)
r2 = y
r3 = z
r2 = r2 + r3
r1 = r1 * r2
```

3 регистра

`(y + z) * exp(x)`:

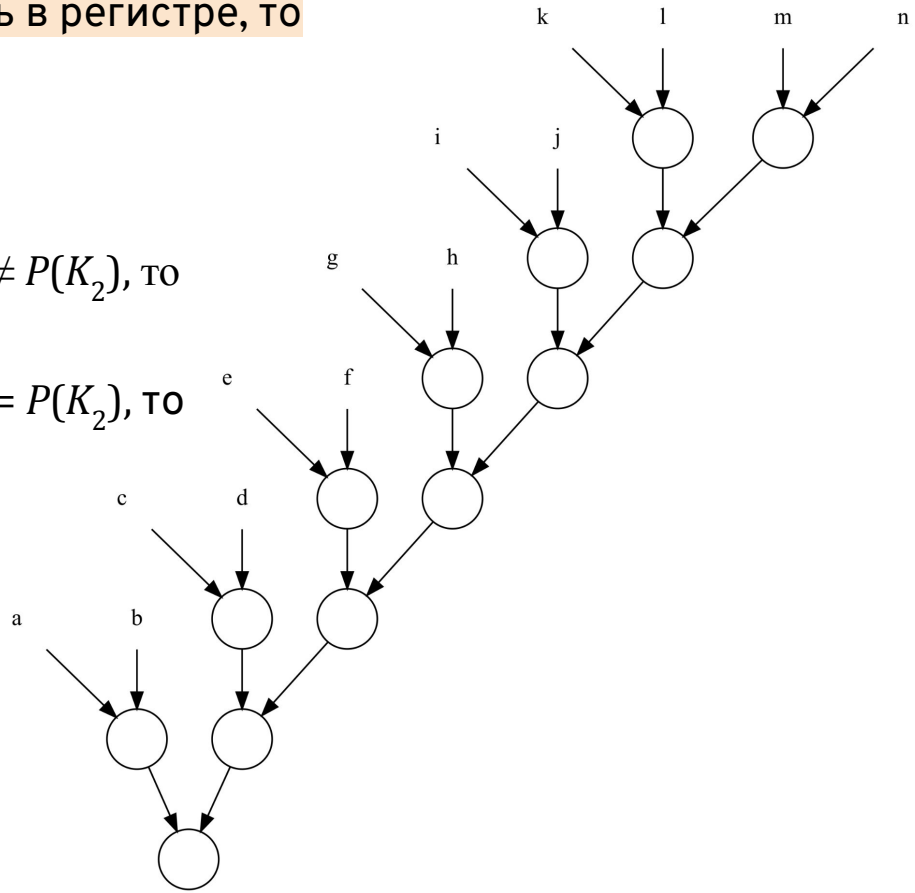
```
r1 = y
r2 = z
r1 = r1 + r2
r2 = x
r2 = exp(r2)
r1 = r1 * r2
```

2 регистра

Лучше начинать трансляцию с подвыражения, требующего **большего** числа регистров, потом эти регистры можно будет переиспользовать!

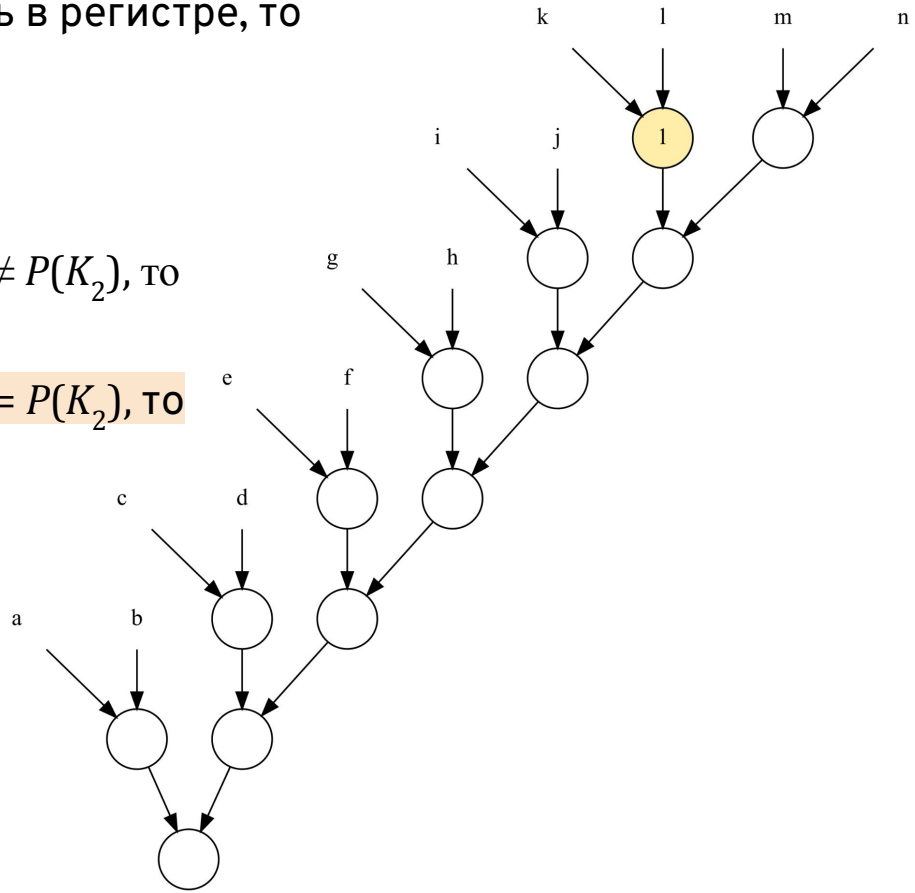
Вычисление числа Ершова — функции порядка P (1)

- Если результат узла K не нужно сохранять в регистре, то
  $P(K) = 0$.
- Если узел K содержит 1 операнд K_1, то
  $P(K) = P(K_1)$.
- Если узел K содержит 2 операнда и $P(K_1) \neq P(K_2)$, то
  $P(K) = \max\{P(K_1), P(K_2)\}$.
- Если узел K содержит 2 операнда и $P(K_1) = P(K_2)$, то
  $P(K) = P(K_1) + 1$.



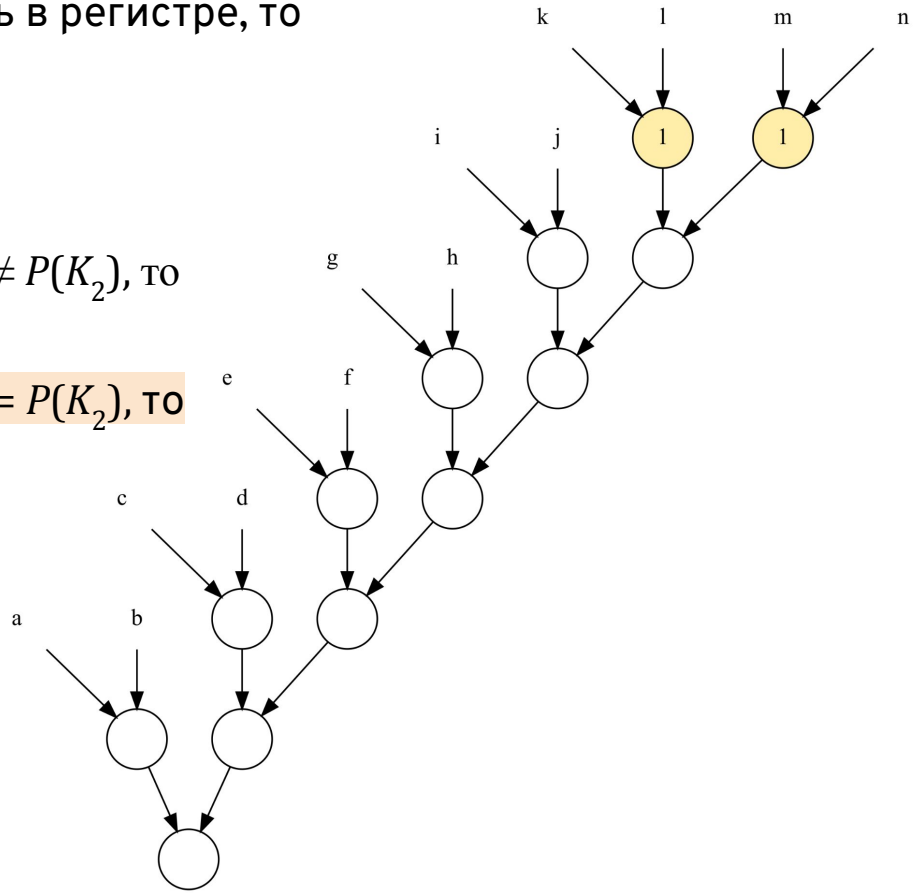
Вычисление числа Ершова – функции порядка P (2)

- Если результат узла K не нужно сохранять в регистре, то
  $P(K) = 0$.
- Если узел K содержит 1 операнд K_1, то
  $P(K) = P(K_1)$.
- Если узел K содержит 2 операнда и $P(K_1) \neq P(K_2)$, то
  $P(K) = \max\{P(K_1), P(K_2)\}$.
- Если узел K содержит 2 операнда и $P(K_1) = P(K_2)$, то
  $P(K) = P(K_1) + 1$.



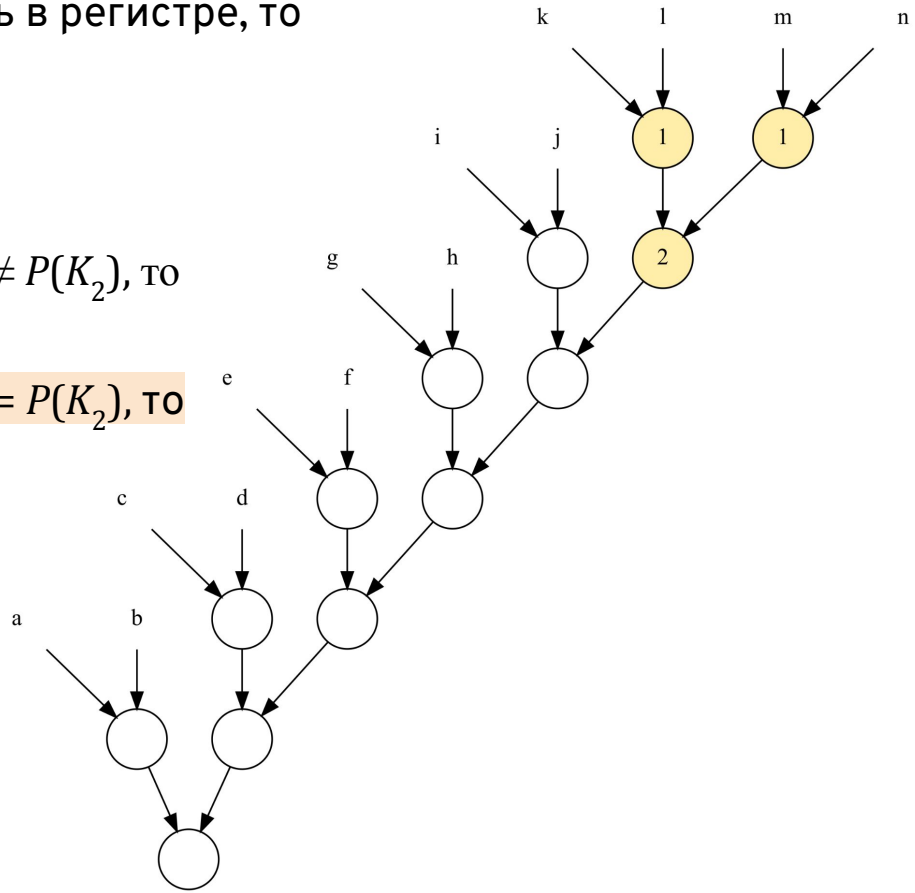
Вычисление числа Ершова – функции порядка P (3)

- Если результат узла K не нужно сохранять в регистре, то
  $P(K) = 0$.
- Если узел K содержит 1 операнд K_1, то
  $P(K) = P(K_1)$.
- Если узел K содержит 2 операнда и $P(K_1) \neq P(K_2)$, то
  $P(K) = \max\{P(K_1), P(K_2)\}$.
- Если узел K содержит 2 операнда и $P(K_1) = P(K_2)$, то
  $P(K) = P(K_1) + 1$.



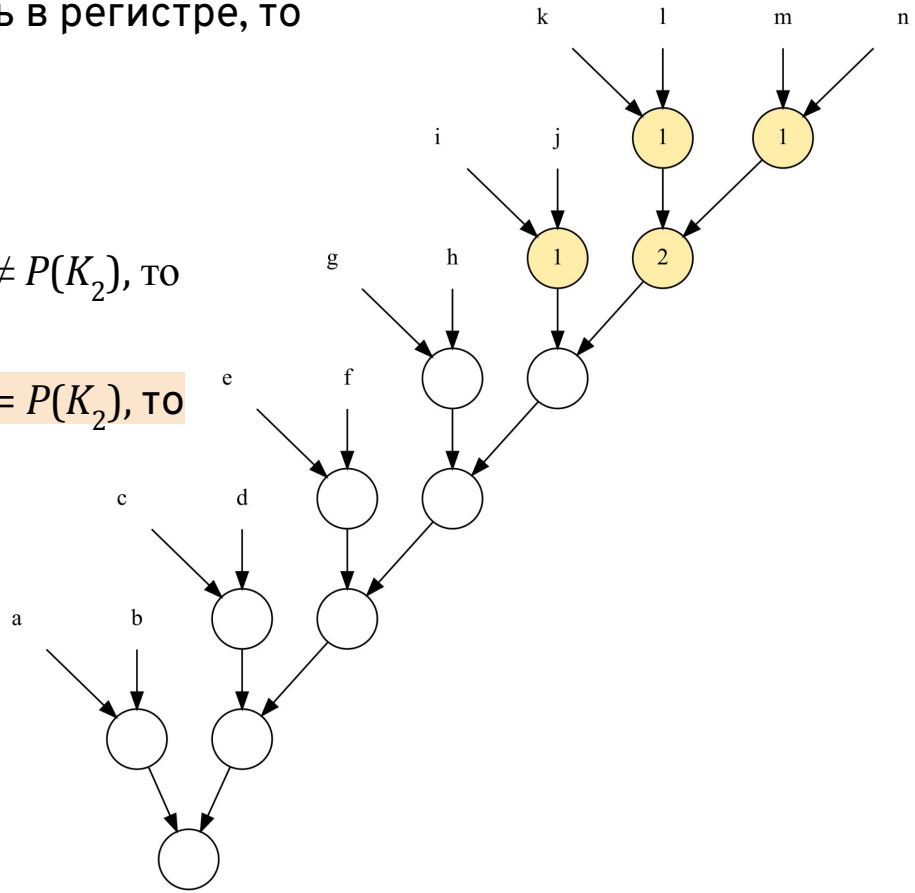
Вычисление числа Ершова – функции порядка P (4)

- Если результат узла K не нужно сохранять в регистре, то
  $P(K) = 0$.
- Если узел K содержит 1 операнд K_1, то
  $P(K) = P(K_1)$.
- Если узел K содержит 2 операнда и $P(K_1) \neq P(K_2)$, то
  $P(K) = \max\{P(K_1), P(K_2)\}$.
- Если узел K содержит 2 операнда и $P(K_1) = P(K_2)$, то
  $P(K) = P(K_1) + 1$.



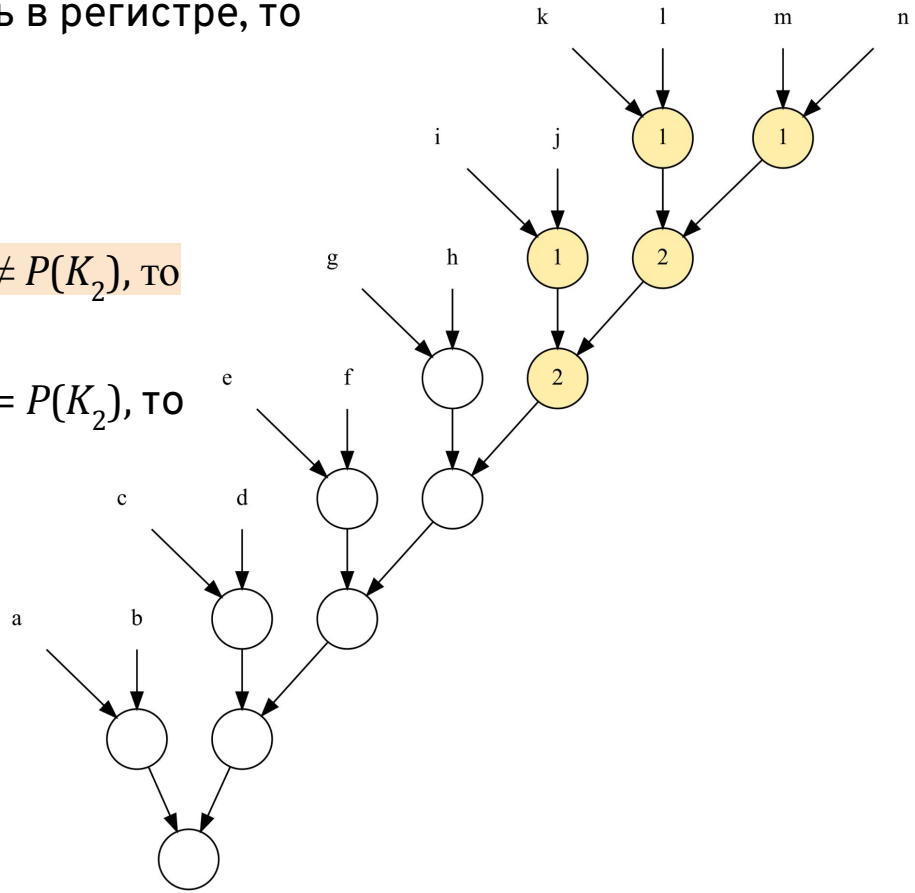
Вычисление числа Ершова – функции порядка P (5)

- Если результат узла K не нужно сохранять в регистре, то
  $P(K) = 0$.
- Если узел K содержит 1 операнд K_1, то
  $P(K) = P(K_1)$.
- Если узел K содержит 2 операнда и $P(K_1) \neq P(K_2)$, то
  $P(K) = \max\{P(K_1), P(K_2)\}$.
- Если узел K содержит 2 операнда и $P(K_1) = P(K_2)$, то
  $P(K) = P(K_1) + 1$.



Вычисление числа Ершова – функции порядка P (6)

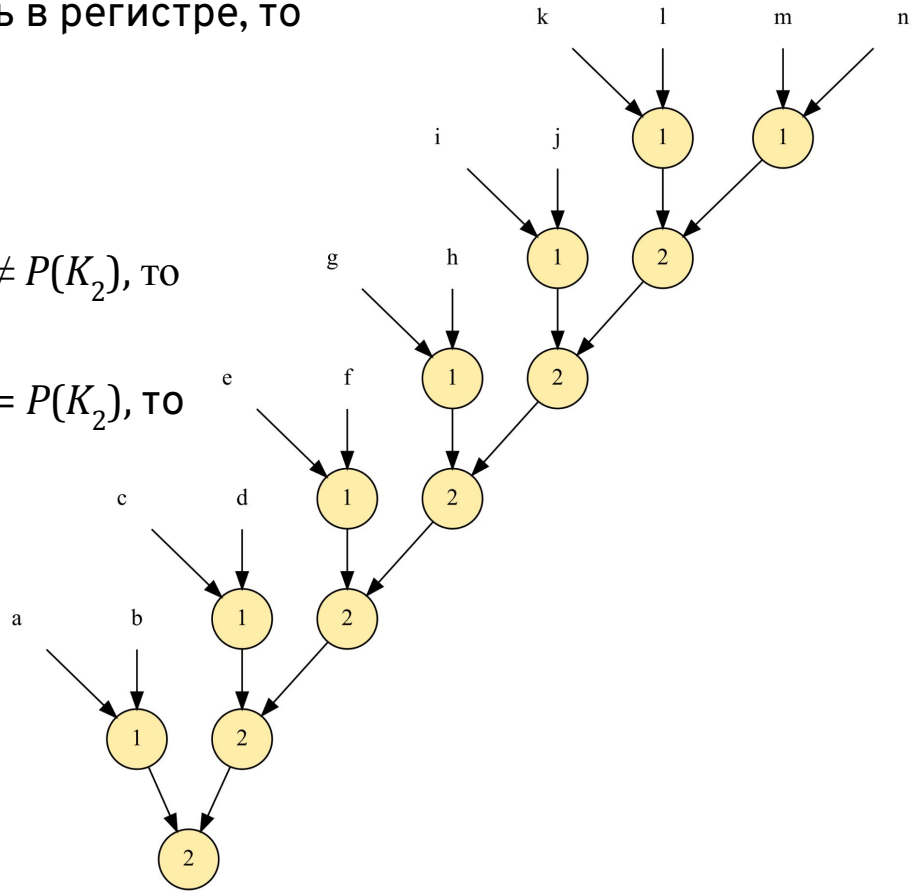
- Если результат узла K не нужно сохранять в регистре, то
  $P(K) = 0$.
- Если узел K содержит 1 операнд K_1, то
  $P(K) = P(K_1)$.
- Если узел K содержит 2 операнда и $P(K_1) \neq P(K_2)$, то
  $P(K) = \max\{P(K_1), P(K_2)\}$.
- Если узел K содержит 2 операнда и $P(K_1) = P(K_2)$, то
  $P(K) = P(K_1) + 1$.



Вычисление числа Ершова – функции порядка P (7)

- Если результат узла K не нужно сохранять в регистре, то
  $P(K) = 0$.
- Если узел K содержит 1 операнд K_1, то
  $P(K) = P(K_1)$.
- Если узел K содержит 2 операнда и $P(K_1) \neq P(K_2)$, то
  $P(K) = \max\{P(K_1), P(K_2)\}$.
- Если узел K содержит 2 операнда и $P(K_1) = P(K_2)$, то
  $P(K) = P(K_1) + 1$.

P дает минимальное число регистров для вычисления **дерева** выражения.



Алгоритм упорядочивания команд

Обойти граф в глубину от корня:

Для бинарной операции первым генерировать код для аргумента, имеющего **наибольшее** значение P.

Для НАГ, в отличие от дерева, минимальности затраченных регистров **не гарантируется** – это **NP-полная** задача.

Через **12** лет после Ершова аналогичный алгоритм предложили Сети и Ульман.

Итоги

Алгоритмы из прошлого, о которых шла речь

1. Синтаксический разбор арифметического выражения.
2. Распределение регистров на линейном участке.
3. Графовые преобразования на основе правил переписывания.
4. Нумерация значений.
5. Планирование команд для экономии регистров.

Достаточно, чтобы написать небольшой компилятор?

О важности обращения к первоисточникам

Выдержка из курса COMP 512: Advanced Compiler Construction [1]:

19. Global Register Allocation Via Graph Coloring, with an emphasis on the Chaitin-Briggs Allocators
    - P. Briggs, K.D. Cooper, L. Torczon, "Improvements to Graph Coloring Register Allocation,", *ACM Transactions on Programming Languages and Systems* (TOPLAS), 16(3), May 1994, pages 428-455. (DOI)
    - G.J. Chaitin, M.A. Auslander, A.K. Chandra, J. Cocke, M.E. Hopkins, and P.W. Markstein, "Register Allocation Via Coloring,", *Computer Languages*, 6(1), January 1981, pages 47--57. (DOI)
    - G.J. Chaitin, "Register Allocation and Spilling Via Graph Coloring," *Proceedings of the ACM SIGPLAN Symposium on Compiler Construction, SIGNPLAN NOTICES* 17(6), June 1982, pages 98--105. (DOI)
    - G.J. Chaitin, "Register Allocation and Spilling Via Graph Coloring," United States Patent 4,571,678, February 1986.
    - S.S. Lavrov, "Store Economy in Closed Operator Scheme", *Journal of Computational Mathematics and Mathematical Physics I*, 4, 1961, pages 687-701. (Published in English translation in *USSR Computational Mathematics and Mathematical Physics* 1(3), 1962. pages 810--828.)
    - A.P. Ershov, "Reduction of the Problem of Memory Allocation in Programming to the Problem of Coloring the Vertices of Graphs", *Doklady Akademii Naux SSSR* 142(4), pages 785-787. (Published in English translation in *Soviet Mathematics* 3(1), January 1962, pages 163-165.)
    - K.D. Cooper, T.J. Harvey, and L. Torczon, "How to Build an Interference Graph", *Software--Practice and Experience*. 28(4), April 1998. (DOI)

[1] https://www.clear.rice.edu/comp512/Lectures/