

5. Глобальная нумерация значений

5.1. Базовый алгоритм локальной нумерации значений

5.1.1. Вводные замечания

- ◇ *Нумерация значений* – классический метод анализа и преобразования промежуточного представления (трехадресного кода) в *ОАГ* – ориентированный ациклический граф (лекция 1, п. 1.6.2)

Идея: алгоритм присваивает индивидуальный номер каждому значению, которое будет вычислено во время выполнения компилируемой программы, при этом два выражения e_i и e_j получают один и тот же номер тогда и только тогда, когда удастся доказать, что значения выражений e_i и e_j равны для всех возможных значений их операндов.

5.1. Базовый алгоритм локальной нумерации значений

5.1.1. Вводные замечания

- ◇ При построении ОАГ удобно представить его в виде таблицы значений (*TЗ*), каждая строка ТЗ (массива структур) представляет один узел ОАГ.
- ◇ Терминальные узлы ОАГ – идентификаторы (**id**) переменных, определенных вне рассматриваемого базового блока, и константы (**nm**) представляются в ТЗ структурами из двух полей **<op, ref_ST>** (ссылка **ref_ST** отсылает к таблице символов).
- ◇ Нетерминальные узлы ОАГ – выражения представляются своими сигнатурами: выражению **<op, left, right>**, где **op** – код операции, а **left** и **right** – левый и правый операнды, соответствует *сигнатура* **<op, #left, #right>**, где **#left** и **#right** – номера значений левого и правого операндов (если **op** – унарная операция, то **#right** = 0).
- ◇ Номер значения – это номер строки ТЗ, в которой находится сигнатура операции, определяющей это значение

5.1. Базовый алгоритм локальной нумерации значений

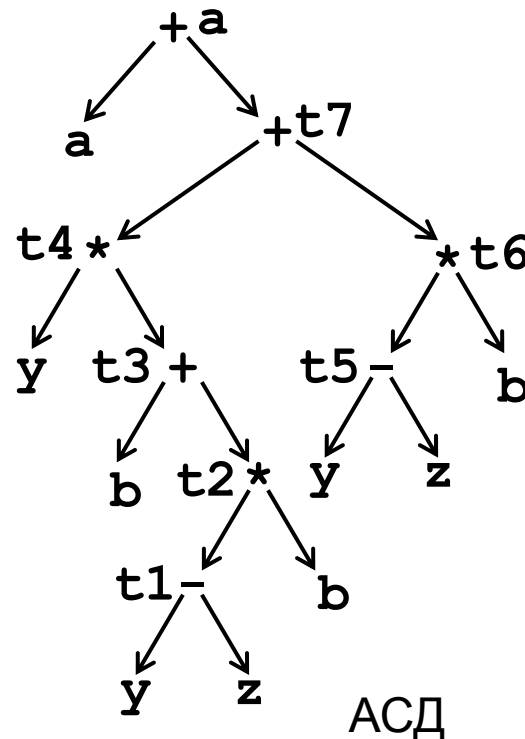
5.1.2 Представление базового блока в виде ориентированного ациклического графа

◇ Пример. Выражение в исходном коде:

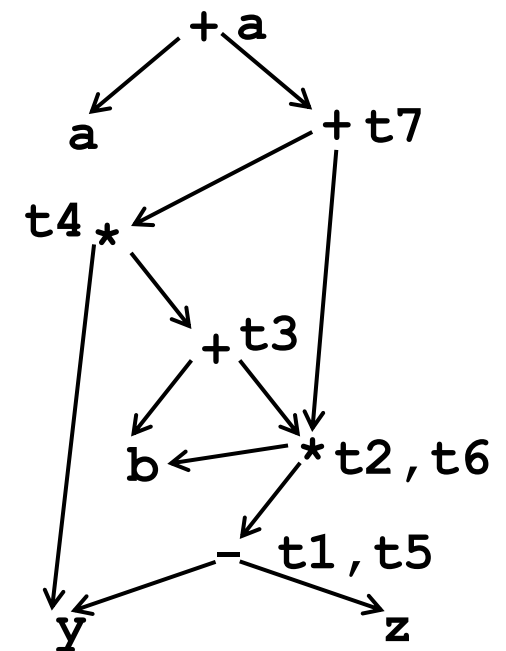
$$a = a + y * (b + (y - z) * b) + (y - z) * b$$

```
t1 ← -, y, z
t2 ← *, t1, b
t3 ← +, b, t2
t4 ← *, y, t3
t5 ← -, y, z
t6 ← *, t5, b
t7 ← +, t4, t6
a ← +, a, t7
```

Выражение в
промежуточном
представлении



АСД



ОАГ

5.1. Базовый алгоритм локальной нумерации значений

5.1.2 Представление ОАГ в виде таблицы значений

◇ Таблица значений рассматриваемого примера имеет вид:

$t1^5 \leftarrow -, y^3, z^4$
 $t2^6 \leftarrow *, t1^5, b^2$
 $t3^7 \leftarrow +, b^2, t2^6$
 $t4^8 \leftarrow *, y^3, t3^7$
 $t5^5 \leftarrow -, y^3, z^4$
 $t6^6 \leftarrow *, t5^5, b^2$
 $t7^9 \leftarrow +, t4^8, t6^6$
 $a^{10} \leftarrow +, a^1, t7^9$

1	id	ссылка в ТС		a
2	id	ссылка в ТС		b
3	id	ссылка в ТС		y
4	id	ссылка в ТС		z
5	–	3	4	t1, t5
6	*	5	2	t2, t6
7	+	2	6	t3
8	*	3	7	t4
9	+	8	6	t7
10	+	1	9	a
# значения	КОП	# операнда	# операнда	Присоединенные переменные
	Определение значения (сигнатура)			

Верхние индексы у идентификаторов – номера соответствующих значений

5.1. Базовый алгоритм локальной нумерации значений

5.1.3 Алгоритм построения ТЗ

Алгоритм (на псевдокоде) построения ТЗ для базового блока B , содержащего n инструкций вида $t_i \leftarrow op_i, l_i, r_i$.

Функция $\#val(s)$ определяет номер значения, соответствующего сигнатуре $s = (op, \#val(l), \#val(r))$:

```
for each " $t_i \leftarrow op_i, l_i, r_i$ " do
     $s_i = (op_i, \#val(l_i), \#val(r_i))$ 
    if (ТЗ содержит  $s_j == s_i$ )
        then
            вернуть  $j$  в качестве значения  $\#val(s_i)$ 
    else
        завести в ТЗ новую строку  $ТЗ_k$ 
        записать сигнатуру  $s_i$  в строку  $ТЗ_k$ 
        вернуть  $k$  в качестве значения  $\#val(s_i)$ 
```

5.1. Базовый алгоритм локальной нумерации значений

5.1.3. Модификация подхода

- ◇ Чтобы ускорить алгоритм нумерации значений для отображения переменных, констант и вычисляемых значений (выражений) на их номера значений используется хэш-таблица.
- ◇ Для переменных и констант в качестве аргумента функции $\#Val(s)$ используются их имена по таблице символов.
- ◇ Для выражения вида $op, opnd_1, opnd_2$ сигнатура (аргумент функции $\#Val(s)$) имеет вид:

$$op, \#Val(opnd_1), \#Val(opnd_2)$$

где $\#Val(opnd_i)$ – номер значения операнда $opnd_i$,
а op – знак операции (например, +).

- ◇ В инструкциях присваивания и копирования
номер значения правой части становится номером значения левой части.

5.1. Базовый алгоритм локальной нумерации значений

5.1.3 Алгоритм построения T3

- ◇ Известно несколько подходов к построению алгоритма глобальной нумерации значений.
- ◇ Мы хотим получить **алгоритм глобальной нумерации значений**, распространив базовый алгоритм на всю анализируемую процедуру.

5.2. Нумерация значений в суперблоках

5.2.1 Расширенные базовые блоки (суперблоки).

- ◇ **Определение.** *Расширенный базовый блок или суперблок E определяется как множество базовых блоков $B_1, B_2, \dots, B_n$, где у блока B_1 может быть несколько предшественников, а каждый из блоков B_i , $2 \leq i \leq n$ имеет в суперблоке единственного предшественника.*

Блоки $B_i \in E$ формируют дерево с корнем B_1 .

У суперблока E может быть несколько выходов на блоки (или суперблоки), не входящие в состав E .

- ◇ Алгоритм нумерации значений в суперблоках будем называть *расширенным алгоритмом локальной нумерации значений*.

5.2. Нумерация значений в суперблоках

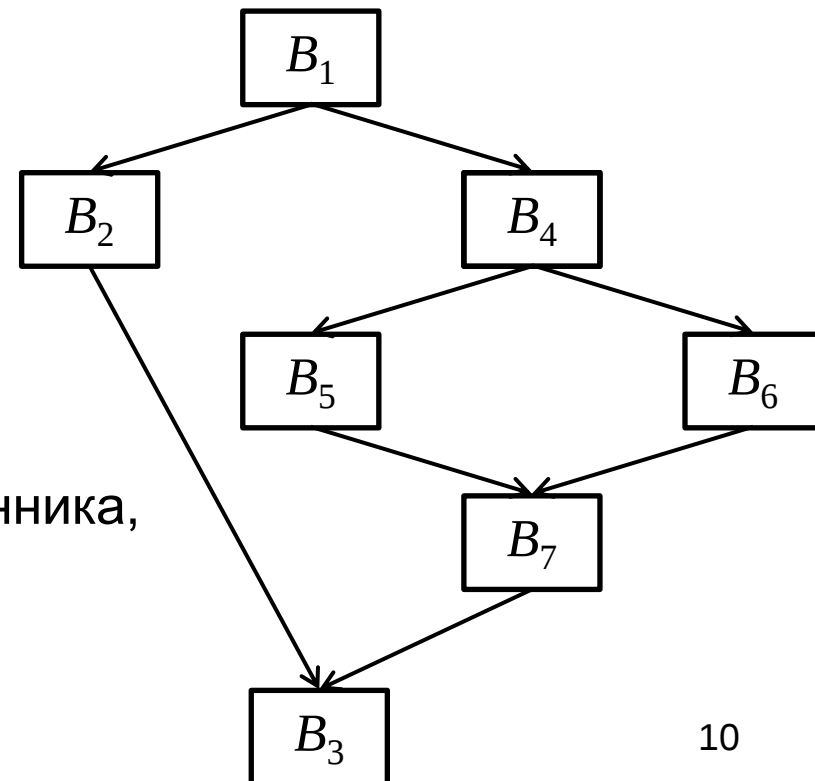
5.2.1 Суперблоки. Пример.

B_1 $m \leftarrow +, a, b$ $n \leftarrow +, a, b$	B_2 $p \leftarrow +, c, d$ $r \leftarrow +, c, d$	B_3 $y \leftarrow +, a, b$ $z \leftarrow +, c, d$
B_4 $q \leftarrow +, a, b$ $r \leftarrow +, c, d$	B_5 $e \leftarrow +, b, 18$ $s \leftarrow +, a, b$ $u \leftarrow +, e, f$	B_6 $e \leftarrow +, b, 17$ $t \leftarrow +, c, d$ $u \leftarrow +, e, f$
B_7 $v \leftarrow +, a, b$ $w \leftarrow +, c, d$ $x \leftarrow +, e, f$		

В коде, показанном на рисунке можно выделить три суперблока:

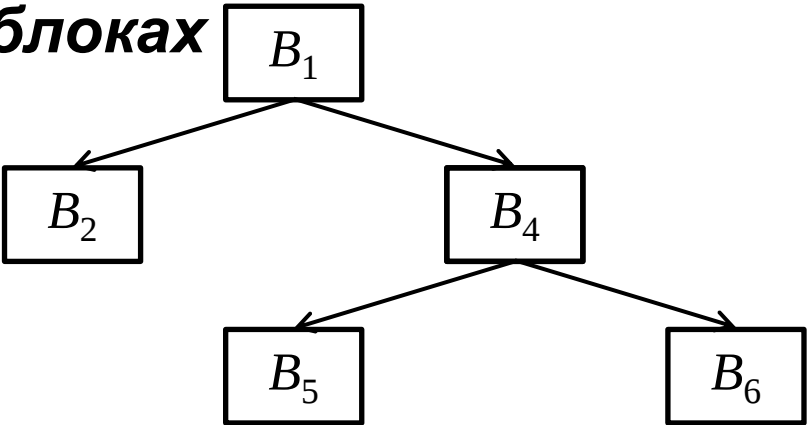
$\{B_1, B_2, B_4, B_5, B_6\}$, $\{B_7\}$ и $\{B_3\}$.

Блоки B_7 и B_3 имеют по два предшественника, поэтому их нельзя включить в суперблок больших размеров



5.2. Нумерация значений в суперблоках

5.3.1 Суперблоки. Пример.



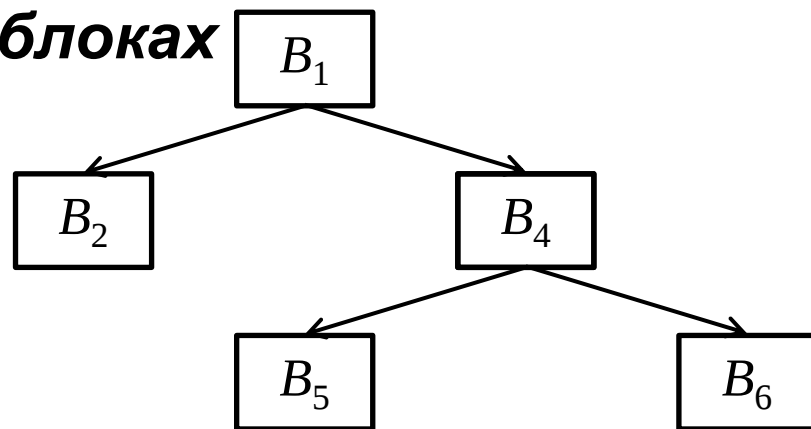
B_1 $m \leftarrow +, a, b$ $n \leftarrow +, a, b$	B_2 $p \leftarrow +, c, d$ $r \leftarrow +, c, d$	
B_4 $q \leftarrow +, a, b$ $r \leftarrow +, c, d$	B_5 $e \leftarrow +, b, 18\ s$ $\leftarrow +, a, b$ $u \leftarrow +, e, f$	B_6 $e \leftarrow +, b, 17\ t$ $\leftarrow +, c, d$ $u \leftarrow +, e, f$

В суперблоке $\{B_1, B_2, B_4, B_5, B_6\}$ можно выделить три пути:
 $\{B_1, B_2\}$, $\{B_1, B_4, B_5\}$ и $\{B_1, B_4, B_6\}$.

Пронумеруем значения вдоль каждого из этих путей.

5.2. Нумерация значений в суперблоках

5.2.1 Суперблоки. Пример.



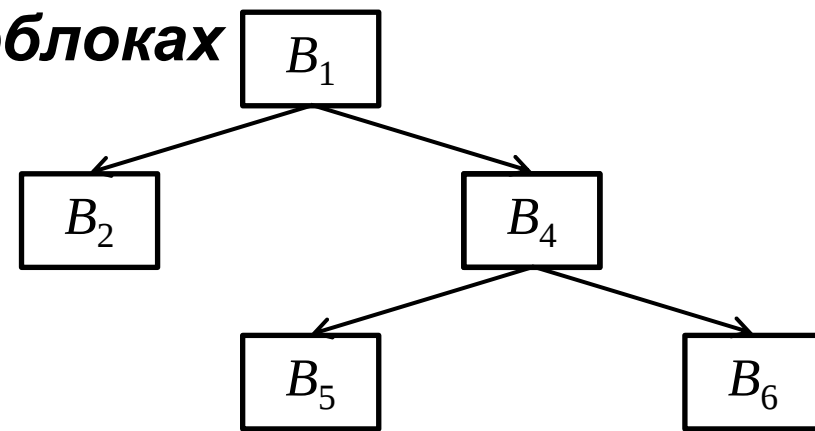
B_1 $m \leftarrow +, a, b$ $n \leftarrow +, a, b$	B_2 $p \leftarrow +, c, d$ $r \leftarrow +, c, d$	
B_4 $q \leftarrow +, a, b$ $r \leftarrow +, c, d$	B_5 $e \leftarrow +, b, 18 s$ $\leftarrow +, a, b$ $u \leftarrow +, e, f$	B_6 $e \leftarrow +, b, 17 t$ $\leftarrow +, c, d$ $u \leftarrow +, e, f$

а) Путь $\{B_1, B_4, B_5\}$:

- ◇ строится (хэш) таблица значений для блока B_1
- ◇ таблица значений для пути $\{B_1, B_4\}$ строится как продолжение таблицы значений для блока B_1
- ◇ таблица значений для пути $\{B_1, B_4, B_5\}$ строится как продолжение хэш-таблицы таблица значений для пути $\{B_1, B_4\}$.

5.3. Нумерация значений в суперблоках

5.3.1 Суперблоки. Пример.



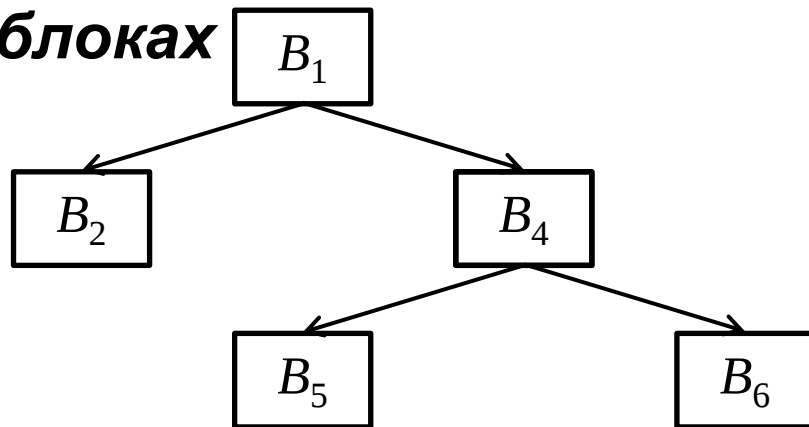
B_1	$m \leftarrow +, a, b$ $n \leftarrow +, a, b$	B_2	$p \leftarrow +, c, d$ $r \leftarrow +, c, d$	B_5	
B_4	$q \leftarrow +, a, b$ $r \leftarrow +, c, d$	B_5	$e \leftarrow +, b, 18 s$ $\leftarrow +, a, b$ $u \leftarrow +, e, f$	B_6	$e \leftarrow +, b, 17 t$ $\leftarrow +, c, d$ $u \leftarrow +, e, f$

а) Путь $\{B_1, B_4, B_5\}$:

- ◇ строится (хэш) таблица значений для блока B_1
- ◇ таблица значений для пути $\{B_1, B_4\}$ строится как продолжение таблицы значений для блока B_1
- ◇ таблица значений для пути $\{B_1, B_4, B_5\}$ строится как продолжение таблицы значений для пути $\{B_1, B_4\}$.
- ◇ путь $\{B_1, B_4, B_5\}$ рассматривается как единый базовый блок

5.2. Нумерация значений в суперблоках

5.2.1 Суперблоки. Пример.



B_1	$m \leftarrow +, a, b$ $n \leftarrow +, a, b$	B_2	$p \leftarrow +, c, d$ $r \leftarrow +, c, d$	B_5	
B_4	$q \leftarrow +, a, b$ $r \leftarrow +, c, d$	B_5	$e \leftarrow +, b, 18 s$ $\leftarrow +, a, b$ $u \leftarrow +, e, f$	B_6	$e \leftarrow +, b, 17 t$ $\leftarrow +, c, d$ $u \leftarrow +, e, f$

б) Путь $\{B_1, B_4, B_6\}$:

- ♦ таблица значений для пути $\{B_1, B_4, B_6\}$ строится как продолжение таблицы значений для пути $\{B_1, B_4\}$.
- ♦ путь $\{B_1, B_4, B_6\}$ рассматривается как единый базовый блок

в) Путь $\{B_1, B_2\}$:

- ♦ таблица значений для пути $\{B_1, B_2\}$ строится как продолжение таблицы значений для блока B_1 .

5.2. Нумерация значений в суперблоках

5.2.2 Контекстные таблицы значений

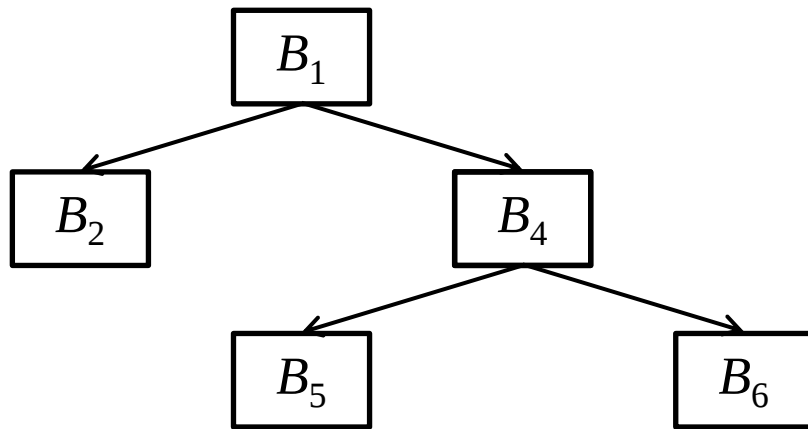
- ◇ Чтобы нумерация значений над суперблоками была эффективной, компилятор должен повторно использовать результаты для блоков, которые входят в несколько путей:
- ◇ После обработки ТЗ для $\{A, C, D\}$ необходимо воссоздать состояние ТЗ в конце обработки $\{A, C\}$, чтобы можно было использовать ТЗ для $\{A, C\}$ при обработке E из $\{A, C, E\}$.
- ◇ Оценка размера ТЗ для каждого контекста (очевидная): в рассматриваемом промежуточном представлении (последовательность трехадресных инструкций) число имен не может более чем в три раза превышать число инструкций, так как в инструкции используется не более трех имен. Это соображение позволяет сразу выделить достаточно памяти под каждую ТЗ и исключить возможность переполнения ТЗ.

5.2. Нумерация значений в суперблоках

5.2.2 Контекстные таблицы значений

◇ Обеспечивается это, естественно, с помощью стэка:

В нашем примере сначала в стэке строится ТЗ для B_1 , потом – ТЗ для B_2 , как продолжение ТЗ для B_1 , потом выталкивается ТЗ для B_2 , и на ее месте строится ТЗ для $\{B_1, B_2, B_4, B_5, B_6\}$, и т.д., пока не будут обойдены все блоки.



Объединенные ТЗ образуют *текущий контекст*.

После того как будут обработаны все пути из суперблока $\{B_1, B_2, B_4, B_5, B_6\}$ будут обработаны блоки B_7 и B_3 . При их обработке не удастся использовать ТЗ других блоков, так как каждый из них достижим по двум путям, что может привести к путанице при объединении ТЗ.

5.2. Нумерация значений в суперблоках

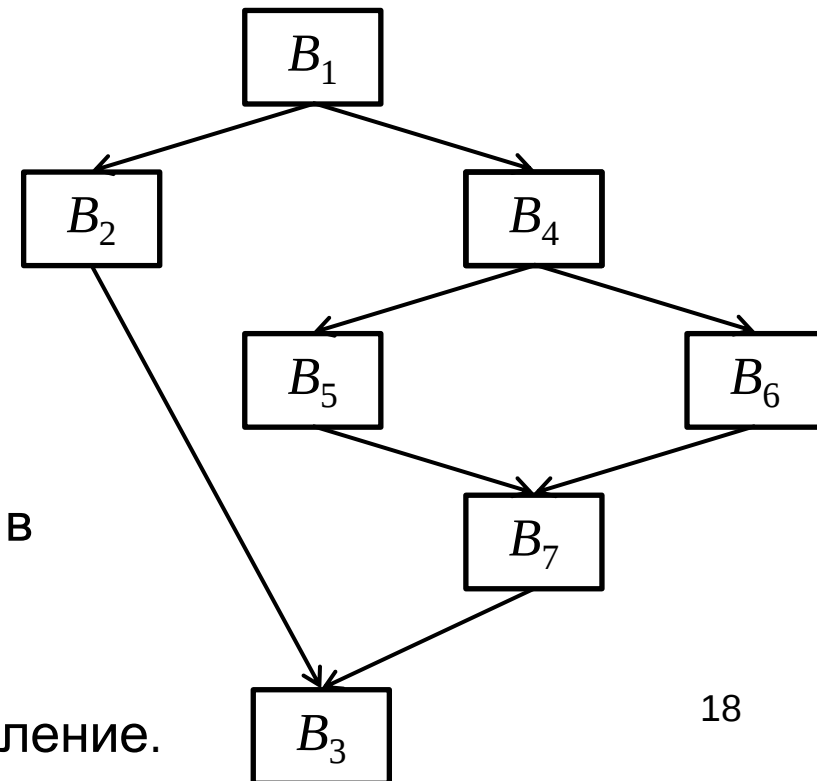
5.2.3 Затруднение

- ◇ Если в нескольких базовых блоках, входящих в состав суперблока (например, в блоках B_1 и B_4), определяются номера значений одной и той же переменной (используется одно и то же имя переменной) результат его определения в одном блоке (B_4) может попасть в контекст, связанный с другим блоком (B_1).
- ◇ В этом случае при удалении ТЗ блока B_4 из стэка в стэке могут сохраниться записи об этой переменной в ТЗ блока B_1 . Так как это необходимо учесть в алгоритме нумерации значений, алгоритм усложнится.
- ◇ Описанного затруднения легко избежать, если выполнять нумерацию значений для суперблоков в SSA-форме: тогда каждое имя будет определяться только один раз, и описанные конфликты невозможны.

5.2. Нумерация значений в суперблоках

5.2.4 Контекстные таблицы значений и SSA-форма

B_1 $m_0 \leftarrow +, a_0, b_0$ $n_0 \leftarrow +, a_0, b_0$	B_2 $p_0 \leftarrow +, c_0, d_0$ $r_0 \leftarrow +, c_0, d_0$	B_3 $r_2 \leftarrow \phi(r_0, r_1)$ $y_0 \leftarrow +, a_0, b_0$ $z_0 \leftarrow +, c_0, d_0$
B_4 $q_0 \leftarrow +, a_0, b_0$ $r_1 \leftarrow +, c_0, d_0$	B_5 $e_0 \leftarrow +, b_0, 18$ $s_0 \leftarrow +, a_0, b_0$ $u_0 \leftarrow +, e_0, f_0$	B_6 $e_1 \leftarrow +, b_0, 17$ $t_0 \leftarrow +, c_0, d_0$ $u_1 \leftarrow +, e_1, f_0$
B_7 $e_2 \leftarrow \phi(e_0, e_1)$ $u_2 \leftarrow \phi(u_0, u_1)$ $v_0 \leftarrow +, a_0, b_0$ $w_0 \leftarrow +, c_0, d_0$ $x_0 \leftarrow +, e_2, f_0$		



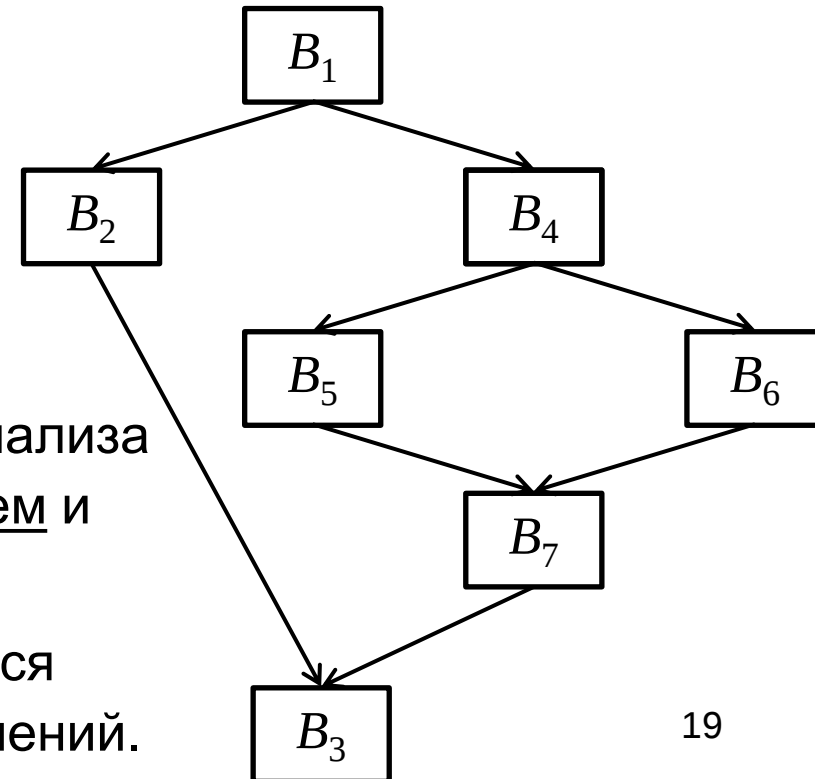
◇ SSA-форма обладает двумя важными свойствами:

- (1) каждое имя определяется только в одной инструкции
- (2) каждое использование значения ссылается только на одно определение.

5.2. Нумерация значений в суперблоках

5.2.4 Контекстные таблицы значений и SSA-форма

B_1 $m_0 \leftarrow +, a_0, b_0$ $\underline{n_0 \leftarrow +, a_0, b_0}$	B_2 $p_0 \leftarrow +, c_0, d_0$ $\underline{r_0 \leftarrow +, c_0, d_0}$	B_3 $r_2 \leftarrow \phi(r_0, r_1)$ $y_0 \leftarrow +, a_0, b_0$ $z_0 \leftarrow +, c_0, d_0$
B_4 $\underline{q_0 \leftarrow +, a_0, b_0}$ $r_1 \leftarrow +, c_0, d_0$	B_5 $e_0 \leftarrow +, b_0, 18$ $\underline{s_0 \leftarrow +, a_0, b_0} \cdot$ $u_0 \leftarrow +, e_0, f_0$	B_6 $e_1 \leftarrow +, b_0, 17$ $\underline{t_0 \leftarrow +, c_0, d_0} \cdot$ $u_1 \leftarrow +, e_1, f_0$
B_7 $e_2 \leftarrow \phi(e_0, e_1)$ $u_2 \leftarrow \phi(u_0, u_1)$ $v_0 \leftarrow +, a_0, b_0$ $w_0 \leftarrow +, c_0, d_0$ $x_0 \leftarrow +, e_2, f_0$		



- ◇ Инструкции, удаляемые алгоритмом анализа суперблоков, выделены подчеркиванием и **ЦВЕТОМ**
- ◇ Инструкции, помеченные **•**, не удаляются алгоритмом локальной нумерации значений.

5.2. Нумерация значений в суперблоках

5.2.5 Оценка расширенного алгоритма локальной нумерации значений

- ◇ Расширенный алгоритм локальной нумерации значений отлично работает внутри суперблока. Однако, при переходе от одного суперблока к другому он пропускает некоторые избыточности.
- ◇ В рассматриваемом примере:
 - ◇ B_3 и B_7 составляют отдельные суперблоки, поэтому вычисления $a_0 + b_0$ и $c_0 + d_0$ в блоках B_7 и B_3 не рассматриваются алгоритмом вместе с вычислениями в других блоках и не распознаются как избыточные, хотя на самом деле они избыточны.
 - ◇ Выражение $e + f$ является доступным в блоке B_7 :
 $e + f$ вычисляется на любом пути, достигающем блока B_7 , и ни e , ни f не переопределяются до вычисления в B_7 .
Потому оно избыточно и будет исключено, хотя это неверно.
Причина: расширенный алгоритм не может установить, что e имеет разные номера значений в блоках D и E .

5.2. Нумерация значений в суперблоках

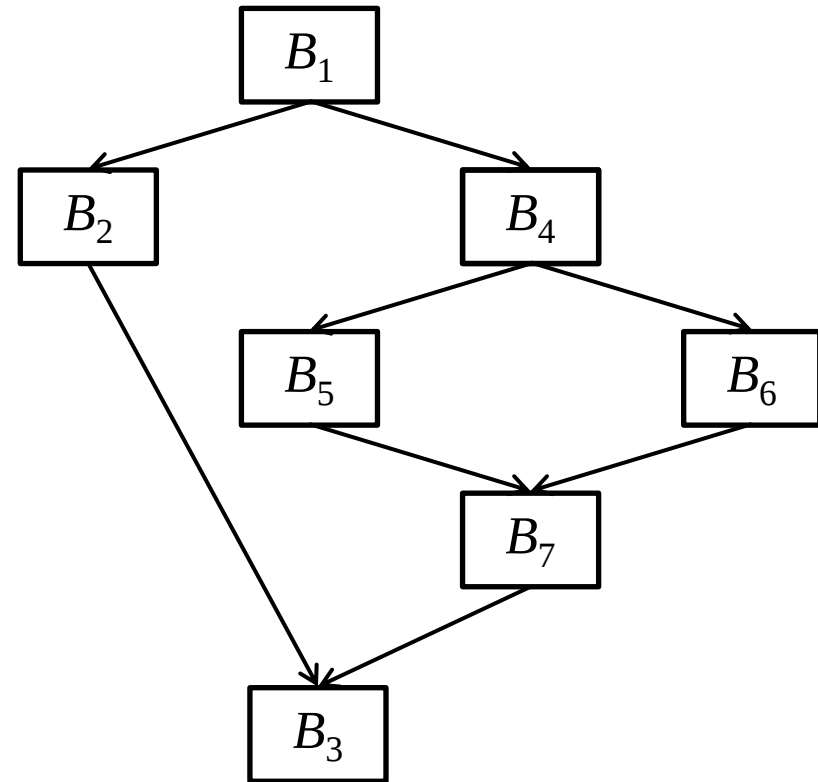
5.2.5 Оценка расширенного алгоритма локальной нумерации значений

- ◇ Несмотря на перечисленные недостатки, расширенный алгоритм заслуживает применения:
он позволяет найти намного больше избыточностей, чем локальный алгоритм, за минимальные дополнительные накладные расходы.
- ◇ Использование механизма контекстно-ориентированных ТЗ позволяет существенно сократить накладные расходы на работу с суперблоками.

5.3 Глобальная нумерация значений

5.3.1 Необходимость обработки точек сбора

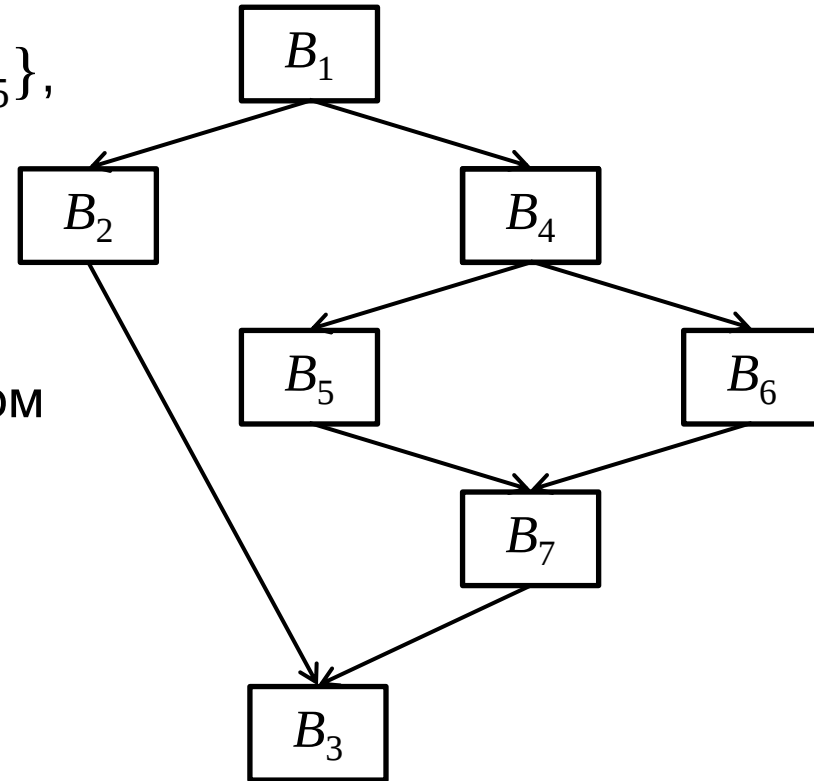
- ◇ Расширенный алгоритм локальной нумерации значений пропускает некоторые избыточности, так как он удаляет **всю** таблицу значений, когда достигает блока, имеющего в ГПУ более одного предшественника (в нашем примере это блоки B_7 и B_3).
- ◇ Нужен алгоритм, который может распространять информацию через точки сбора графа потока (в нашем примере это переходы от B_5 и B_6 к B_7 , и от B_2 и B_7 к B_3).



5.3 Глобальная нумерация значений

5.3.2 Проблема точек сбора

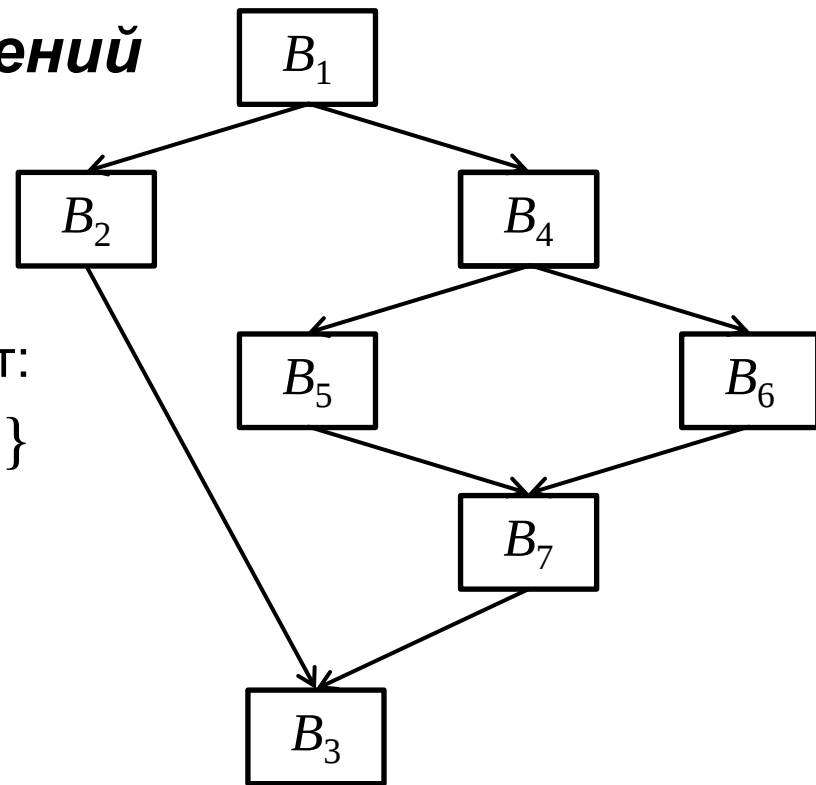
- ◇ Для нумерации значений в блоке B_7 , расширенный алгоритм не может использовать ТЗ для пути $\{B_1, B_4, B_5\}$, так как при этом не учитывается путь от B_6 к B_7 .
- ◇ Алгоритм не может использовать ТЗ для пути $\{B_1, B_4, B_6\}$, так как при этом не учитывается путь от B_5 к B_7 .
- ◇ Алгоритм не может слить ТЗ для B_5 и B_6 , так как операция слияния унифицирует номера значений, полученные вдоль непересекающихся путей. При унификации не учитывается, что, например, вычисления $e + f$ в B_5 и B_6 могут иметь разные номера значений.



5.3 Глобальная нумерация значений

5.3.3 Обработка точек сбора

- ◇ ТЗ, которую алгоритм может использовать для блока B_7 , существует: оба пути, достигающие B_7 ($\{B_1, B_4, B_5\}$ и $\{B_1, B_4, B_6\}$), имеют общее начало $\{B_1, B_4\}$.



Следовательно, алгоритм может использовать ТЗ для B_4 (она содержит в себе таблицу для B_1) в качестве начального состояния ТЗ для B_7 .

Легко видеть, что $B_4 = \text{Idom}(B_7)$.

Следовательно для глобальной нумерации значений можно использовать **дерево доминаторов**, визуализирующее отношение *Idom*.

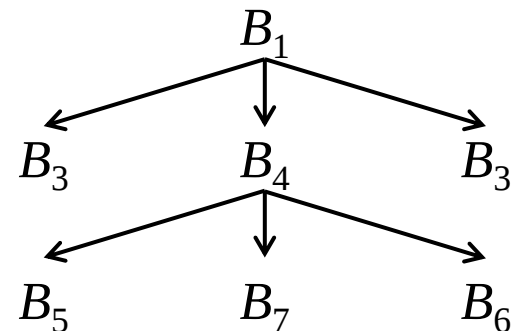
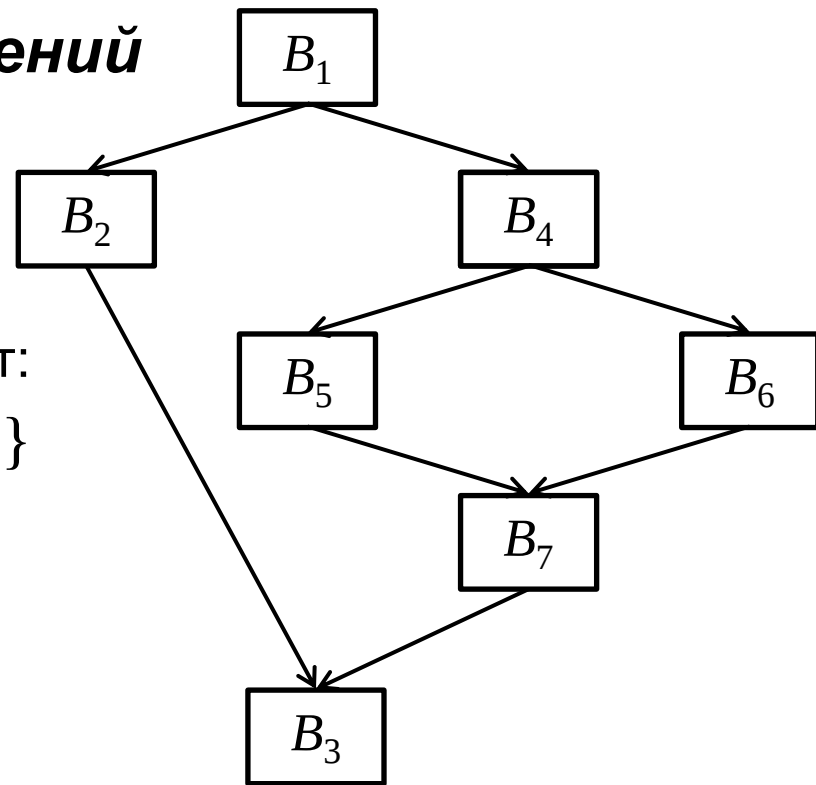
5.3 Глобальная нумерация значений

5.3.3 Обработка точек сбора

- ◇ ТЗ, которую алгоритм может использовать для блока B_7 , существует: оба пути, достигающие B_7 ($\{B_1, B_4, B_5\}$ и $\{B_1, B_4, B_6\}$), имеют общее начало $\{B_1, B_4\}$.

Следовательно, алгоритм может использовать ТЗ для B_4 (она содержит в себе таблицу для B_1) в качестве начального состояния ТЗ для B_7 .

Легко видеть, что $B_4 = \text{Idom}(B_7)$. Следовательно для глобальной нумерации значений можно использовать **дерево доминаторов**, визуализирующее отношение Idom .



Дерево доминаторов

5.4 Алгоритм глобальной нумерации значений

5.4.1 Вводные замечания

- ◇ Алгоритм *DBGVN* (*Dominator Based Global Value Numbering*) выполняет нумерацию значений процедуры с помощью рекурсивного обхода ее дерева доминаторов.
- ◇ ТЗ каждого базового блока инициализируется информацией, полученной при нумерации значений его родителя по дереву доминаторов.
- ◇ Для упрощения реализации алгоритма, *SSA*-имя первого вхождения выражения (на данном пути по дереву доминаторов) становится его номером значения. Это позволяет обойтись без массива имен, так как каждый номер значения – это аналог *SSA*-имени.
- ◇ Когда обнаруживается избыточное вычисление выражения, компилятор удаляет операцию, и заменяет все использования указанного *SSA*-имени на номер значения этого выражения.

5.4 Алгоритм глобальной нумерации значений

5.4.1 Вводные замечания

- ◇ Замена *SSA*-имени на номер значения вычисляющего его выражения допустима в следующих двух ситуациях:
 - ◇ Номер значения может заместить избыточное вычисление выражения в любом блоке, доминатором которого является блок, в котором в первый раз вычисляется это выражение.
 - ◇ Номер значения может заместить избыточное вычисление, результат которого является параметром ϕ -узла, входящего в состав границы доминирования блока, в котором в первый раз вычисляется это выражение.
- ◇ **Обработка ϕ -функций.** Прежде, чем компилятор сможет анализировать ϕ -функцию в блоке, он должен присвоить номера значений всем ее входам. Это возможно не всегда. В частности, любой вход ϕ -функции, значение которого приходит по обратному ребру (относительно дерева доминаторов) не может иметь номера значения.

5.4 Алгоритм глобальной нумерации значений

5.4.2 Обработка ϕ -функций

- ◇ Следующие два условия гарантируют, что при попадании алгоритма в блок всем параметрам ϕ -функций этого блока уже присвоены номера значений:
 1. Рекурсивная процедура *DBGVN* обходит дерево доминаторов по ширине. Это гарантирует, что все предшественники блока будут обработаны до самого блока.
 2. Блок не имеет обратных входных ребер.
- ◇ Если ϕ -функция бессмысленна или избыточна, она должна быть удалена:
 - ◇ ϕ -функция *бессмысленна* если все ее входы имеют одинаковый номер значения. При удалении бессмысленной ϕ -функции ссылки на ее результат заменяются номером значения ее входов.
 - ◇ ϕ -функция *избыточна*, если она вычисляет то же самое значение, что и другая ϕ -функция в том же блоке

5.4 Алгоритм глобальной нумерации значений

5.4.3 Рекурсивный алгоритм глобальной нумерации значений

- ◇ **Вход:** (1) граф потока управления $\langle N, E \rangle$, дерево доминаторов DT
(2) множество Val значений переменных, констант и выражений
- ◇ **Выход:** отображение $VN: Val \rightarrow N \cup \{0\}$ (N – множество натуральных чисел), ставящее в соответствие каждому значению его номер: натуральное число или 0.
- ◇ **Метод:** Применить к корню DT рекурсивную процедуру $DBGVN$ (*Dominator Based Global Value Numbering*)

5.4 Алгоритм глобальной нумерации значений

5.4.4 Рекурсивная процедура *DBGVN* (на псевдокоде)

(1) Обработка ϕ -функций

procedure *DBGVN* (**Block** *B*)

 Отметить начало новой области имен

 || Обработка ϕ -функций

for each *p* $\in$ *B*, где *p* – ϕ -функция вида “ $n \leftarrow \phi(\dots)$ ”

if *p* бессмысленна или избыточна

 поместить номер значения *p* в *VN*[*n*]

 удалить *p*

else

VN[*n*] $\leftarrow$ *n*;

 добавить *p* в ТЗ

5.4 Алгоритм глобальной нумерации значений

5.4.4 Рекурсивная процедура *DBGVN* (на псевдокоде)

(2) Обработка остальных инструкций

procedure *DBGVN*(**Block** *B*)

for each $a \in B$ где a – присваивание вида “ $x \leftarrow op, y, z$ ”

заменить y на $VN[y]$ и z на $VN[z]$

$expr \leftarrow op, y, z$ || $expr$ – вход в ТЗ

if $expr$ может быть упрощено до $expr'$

 Заменить a на “ $x \leftarrow expr'$ ”

$expr \leftarrow expr'$

if $expr$ имеется в ТЗ с номером v

$VN[x] \leftarrow v$

 Удалить a

else Добавить $expr$ в ТЗ с номером x $VN[x] \leftarrow x$ 31

5.4 Алгоритм глобальной нумерации значений

5.4.4 Рекурсивная процедура *DBGVN* (на псевдокоде)

(3) Окончание обработки блока B и переход к обработке его дочерних блоков (по дереву доминаторов)

procedure *DBGVN*(Block B)

for each $s \in Succ(B)$

 скорректировать входы ϕ -функций в s

for each дочернего блока c узла B по дереву доминаторов

$DBGVN(c)$

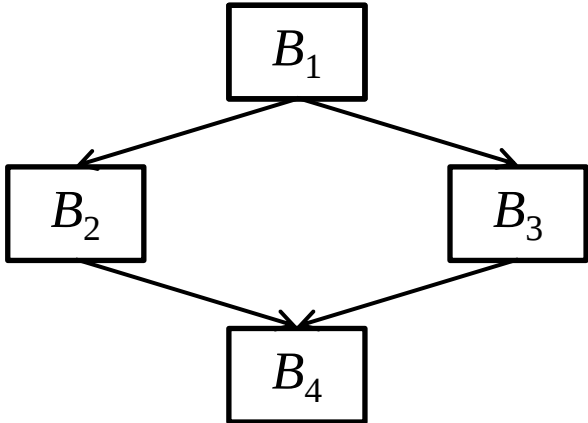
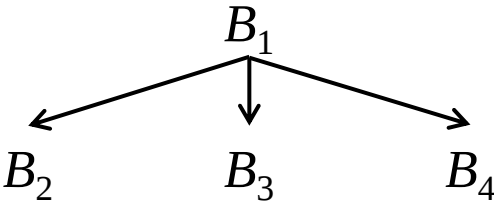
 || рекурсивный вызов

Очистить ТЗ при выходе из области (стэк)

5.4 Алгоритм глобальной нумерации значений

5.4.5 Пример применения алгоритма

◇ Применим алгоритм к фрагменту кода на рисунке. Порядок обработки определяется деревом доминаторов



◇ Обработка **блока B_1** .
Переменным u_0 , v_0 , и w_0
в качестве номеров значений
будут присвоены их *SSA*-имена
 $\langle u_0 \rangle$, $\langle v_0 \rangle$, и $\langle w_0 \rangle$

v	$\#val(v)$
u_0	$\langle u_0 \rangle$
v_0	$\langle v_0 \rangle$
w_0	$\langle w_0 \rangle$
x_0	
y_0	
u_1	
x_1	
y_1	
u_2	
x_2	
y_2	
z_0	
u_3	

B_1 $u_0 \leftarrow a_0 + b_0$ $v_0 \leftarrow c_0 + d_0$ $w_0 \leftarrow e_0 + f_0$	B_2 $x_0 \leftarrow c_0 + d_0$ $y_0 \leftarrow c_0 + d_0$
B_3 $u_1 \leftarrow a_0 + b_0$ $x_1 \leftarrow e_0 + f_0$ $y_1 \leftarrow e_0 + f_0$	B_4 $u_2 \leftarrow \phi(u_0, u_1)$ $x_2 \leftarrow \phi(x_0, x_1)$ $y_2 \leftarrow \phi(y_0, y_1)$ $z_0 \leftarrow u_2 + y_2$ $u_3 \leftarrow a_0 + b_0$

После обработки B_1

5.4 Алгоритм глобальной нумерации значений

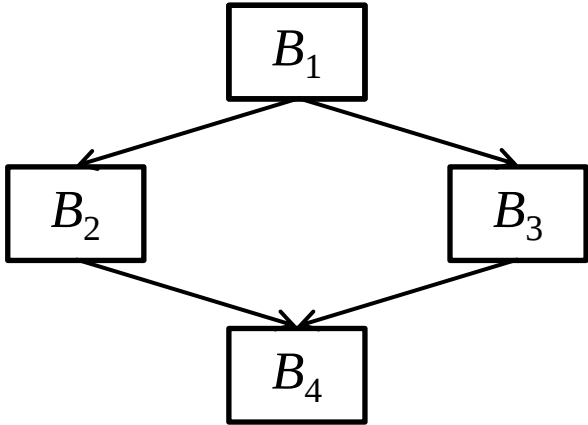
5.4.5 Пример применения алгоритма

◇ Обработка блока B_2 .

Выражение $c_0 + d_0$ определено в блоке B_1 , доминаторе B_2 .

Поэтому в B_2 можно удалить оба присваивания, присвоив x_0 и y_0 номер значения $\langle v_0 \rangle$.
Необходимо также подготовиться к обработке блока B_4 , заменив параметры ϕ -функций u_0 , x_0 , и y_0 номерами их значений $\langle u_0 \rangle$, $\langle v_0 \rangle$, и $\langle v_0 \rangle$.

v	$\#val(v)$
u_0	$\langle u_0 \rangle$
v_0	$\langle v_0 \rangle$
w_0	$\langle w_0 \rangle$
x_0	$\langle v_0 \rangle$
y_0	$\langle v_0 \rangle$
u_1	
x_1	
y_1	
u_2	
x_2	
y_2	
z_0	
u_3	



B_1 $u_0 \leftarrow a_0 + b_0$ $v_0 \leftarrow c_0 + d_0$ $w_0 \leftarrow e_0 + f_0$	B_2
B_3 $u_1 \leftarrow a_0 + b_0$ $x_1 \leftarrow e_0 + f_0$ $y_1 \leftarrow e_0 + f_0$	B_4 $u_2 \leftarrow \phi(\langle u_0 \rangle, u_1)$ $x_2 \leftarrow \phi(\langle v_0 \rangle, x_1)$ $y_2 \leftarrow \phi(\langle v_0 \rangle, y_1)$ $z_0 \leftarrow u_2 + y_2$ $u_3 \leftarrow a_0 + b_0$

После обработки B_2

5.4 Алгоритм глобальной нумерации значений

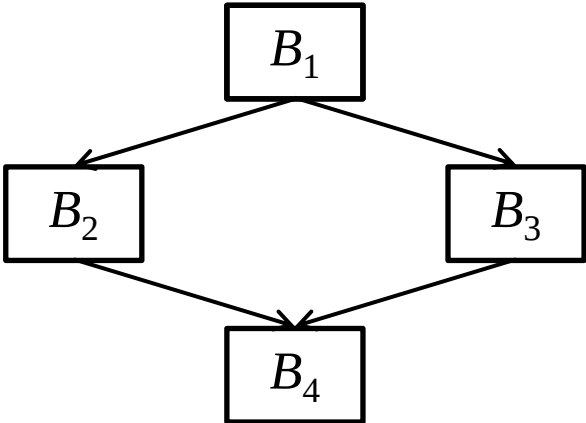
5.4.5 Пример применения алгоритма

◇ Обработка блока B_3 .

Все три выражения в правых частях уже вычислены в B_1 , доминаторе B_3

Поэтому переменным u_1 , x_1 и y_1 присваиваются номера значений $\langle u_0 \rangle$, $\langle w_0 \rangle$ и $\langle w_0 \rangle$ соответственно и удалить присваивания.

В заключение обработки B_3 , необходимо заполнить вторые параметры ϕ -функций из B_4 номерами значений $\langle u_0 \rangle$, $\langle w_0 \rangle$ и $\langle w_0 \rangle$.



v	$\#val(v)$
u_0	$\langle u_0 \rangle$
v_0	$\langle v_0 \rangle$
w_0	$\langle w_0 \rangle$
x_0	$\langle v_0 \rangle$
y_0	$\langle v_0 \rangle$
u_1	$\langle u_0 \rangle$
x_1	$\langle w_0 \rangle$
y_1	$\langle w_0 \rangle$
u_2	
x_2	
y_2	
z_0	
u_3	

B_1 $u_0 \leftarrow a_0 + b_0$ $v_0 \leftarrow c_0 + d_0$ $w_0 \leftarrow e_0 + f_0$	B_2
B_3	B_4 $u_2 \leftarrow \phi(\langle u_0 \rangle, \langle u_0 \rangle)$ $x_2 \leftarrow \phi(\langle v_0 \rangle, \langle w_0 \rangle)$ $y_2 \leftarrow \phi(\langle v_0 \rangle, \langle w_0 \rangle)$ $z_0 \leftarrow u_2 + y_2$ $u_3 \leftarrow a_0 + b_0$

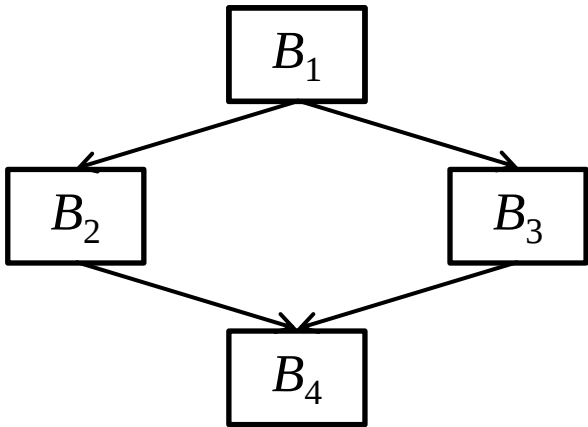
После обработки B_3

5.4 Алгоритм глобальной нумерации значений

5.4.5 Пример применения алгоритма

- ◇ Обработка блока B_4 .
- ◇ Сначала исследуются ϕ -функции. Это возможно, так как выполнены условия :
 - (1) все дети B_1 по дереву доминаторов (B_2, B_3) уже обработаны;
 - (2) в B_4 не входит обратных дуг.
- ◇ ϕ -функция, определяющая u_2 , бессмысленна: оба ее параметра равны $\langle u_0 \rangle$.
 ϕ -функция исключается, а u_2 присваивается номер значения $\langle u_0 \rangle$.

v	$\#val(v)$
u_0	$\langle u_0 \rangle$
v_0	$\langle v_0 \rangle$
w_0	$\langle w_0 \rangle$
x_0	$\langle v_0 \rangle$
y_0	$\langle v_0 \rangle$
u_1	$\langle u_0 \rangle$
x_1	$\langle w_0 \rangle$
y_1	$\langle w_0 \rangle$
u_2	$\langle u_0 \rangle$
x_2	
y_2	
z_0	
u_3	



B_1 $u_0 \leftarrow a_0 + b_0$ $v_0 \leftarrow c_0 + d_0$ $w_0 \leftarrow e_0 + f_0$	B_2
B_3	B_4 $x_2 \leftarrow \phi(\langle v_0 \rangle, \langle w_0 \rangle)$ $y_2 \leftarrow \phi(\langle v_0 \rangle, \langle w_0 \rangle)$ $z_0 \leftarrow u_2 + y_2$ $u_3 \leftarrow a_0 + b_0$

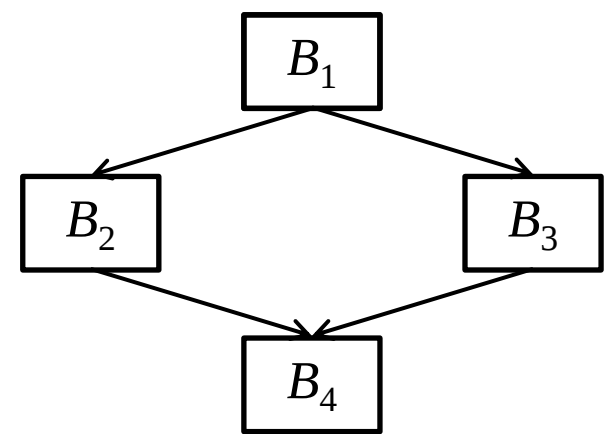
После обработки первой ϕ -функции

5.4 Алгоритм глобальной нумерации значений

5.4.5 Пример применения алгоритма

- ◇ Обработка блока B_4 .
- ◇ Вторая ϕ -функция имеет параметры v_0 и w_0 . Это первое вхождение ϕ -функции с такими параметрами. Ее значению x_2 в качестве номера значения присваивается ее *SSA-имя*.
- ◇ ϕ -функция, определяющая y_2 избыточна, так как y_2 равно x_2 . Поэтому эта ϕ -функция исключается, а y_2 присваивается номер значения $\langle x_2 \rangle$.

v	$\#val(v)$
u_0	$\langle u_0 \rangle$
v_0	$\langle v_0 \rangle$
w_0	$\langle w_0 \rangle$
x_0	$\langle v_0 \rangle$
y_0	$\langle v_0 \rangle$
u_1	$\langle u_0 \rangle$
x_1	$\langle w_0 \rangle$
y_1	$\langle w_0 \rangle$
u_2	$\langle u_0 \rangle$
x_2	$\langle u_0 \rangle$
y_2	$\langle x_2 \rangle$
z_0	
u_3	



B_1 $u_0 \leftarrow a_0 + b_0$ $v_0 \leftarrow c_0 + d_0$ $w_0 \leftarrow e_0 + f_0$	B_2
B_3	B_4 $x_2 \leftarrow \phi(\langle v_0 \rangle, \langle w_0 \rangle)$ $z_0 \leftarrow u_2 + y_2$ $u_3 \leftarrow a_0 + b_0$

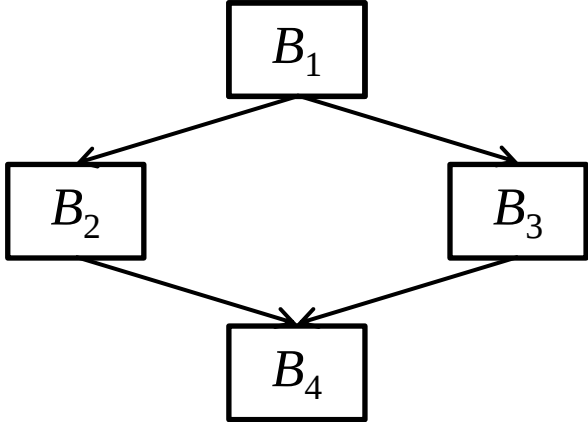
После обработки ϕ -функций

5.4 Алгоритм глобальной нумерации значений

5.4.5 Пример применения алгоритма

- ◇ Обработка блока B_4 .
- ◇ Обработка присваиваний.
- ◇ После подстановки вместо каждого операнда номера его значения для правой части присваивания z_0 получится номер значения $\# = \langle u_0 \rangle + \langle x_2 \rangle$.
- ◇ Присваивание u_3 избыточно (номер значения его правой части такой же, как у правой части присваивания u_0 в блоке B_1 . Поэтому это присваивание исключается, а u_3 присваивается номер значения $\langle u_0 \rangle$.

v	$\#val(v)$
u_0	$\langle u_0 \rangle$
v_0	$\langle v_0 \rangle$
w_0	$\langle w_0 \rangle$
x_0	$\langle v_0 \rangle$
y_0	$\langle v_0 \rangle$
u_1	$\langle u_0 \rangle$
x_1	$\langle w_0 \rangle$
y_1	$\langle w_0 \rangle$
u_2	$\langle u_0 \rangle$
x_2	$\langle u_0 \rangle$
y_2	$\langle x_2 \rangle$
z_0	$\#$
u_3	$\langle u_0 \rangle$



B_1 $u_0 \leftarrow a_0 + b_0$ $v_0 \leftarrow c_0 + d_0$ $w_0 \leftarrow e_0 + f_0$	B_2
B_3	B_4 $x_2 \leftarrow \phi(x_0, x_1)$ $z_0 \leftarrow u_2 + y_2$

После обработки B_4

5.4 Алгоритм глобальной нумерации значений

5.4.5 Заключительные замечания

- ◇ Алгоритм глобальной нумерации значений была описана для программ, уже переведенных в *SSA*-форму. Однако, можно включить нумерацию значений в процесс конструирования *SSA*.
- ◇ В результате включения нумерации значений в процесс конструирования *SSA* повышается производительность оптимизатора, так как
 - ◇ сокращается объем выполняемой работы
 - ◇ уменьшается размер *SSA*-формы программы.
- ◇ Алгоритм глобальной нумерации значений объединяется с алгоритмом переименования переменных при построении *SSA*-формы. В данном курсе этот объединенный алгоритм не рассматривается.