4. Форма статического единственного присваивания (SSA-форма)

4.1 Доминаторы

4.1.1 Определение

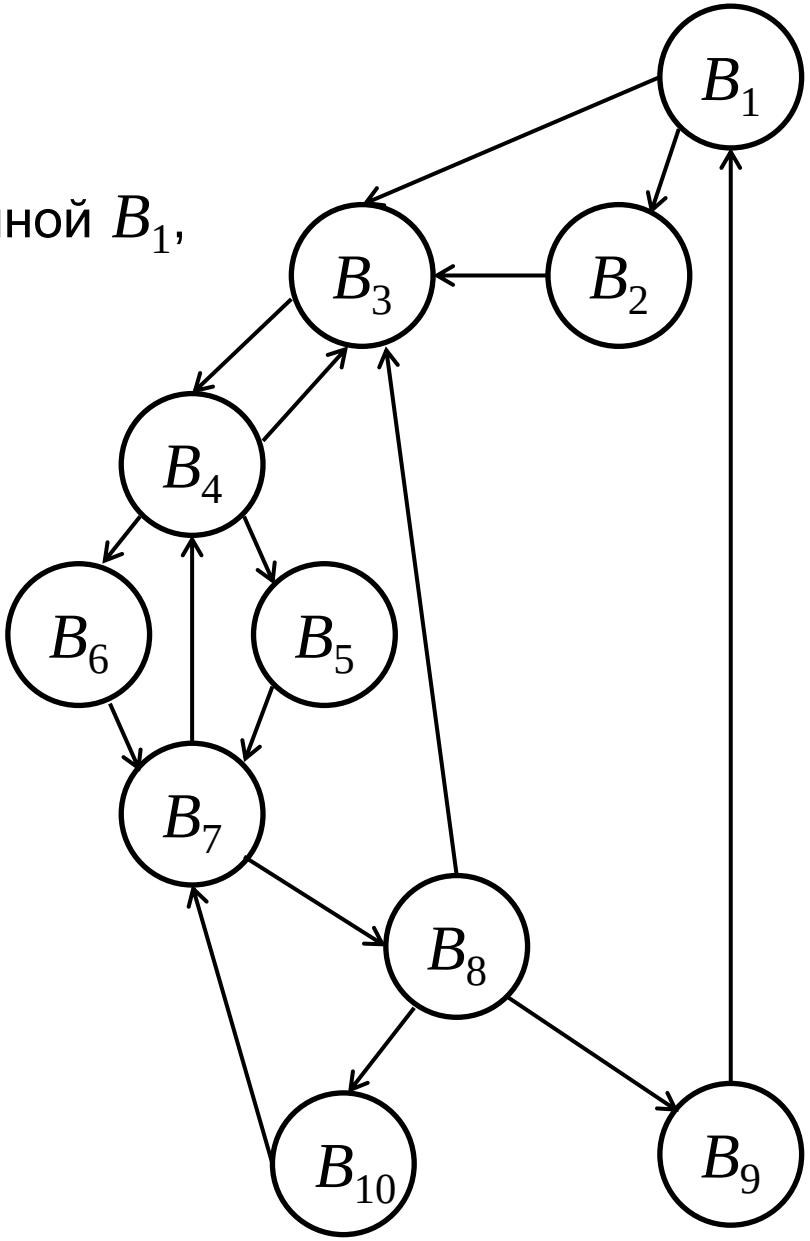
◊ В ГПУ вершина d является *доминатором* вершины n (этот факт записывается как $d\ dom\ n$ или $d = Dom(n)$), если любой путь от вершины $Entry$ до вершины n проходит через вершину d.

◊ **Замечание**. Из определения 4.2.1 следует, что каждая вершина n является доминатором самой себя, так как путь от $Entry$ до n проходит через n.

4.1 Доминаторы

4.1.2 Примеры доминаторов

◊ Рассмотрим ГПУ с входной вершиной B_1, показанный на рисунке.



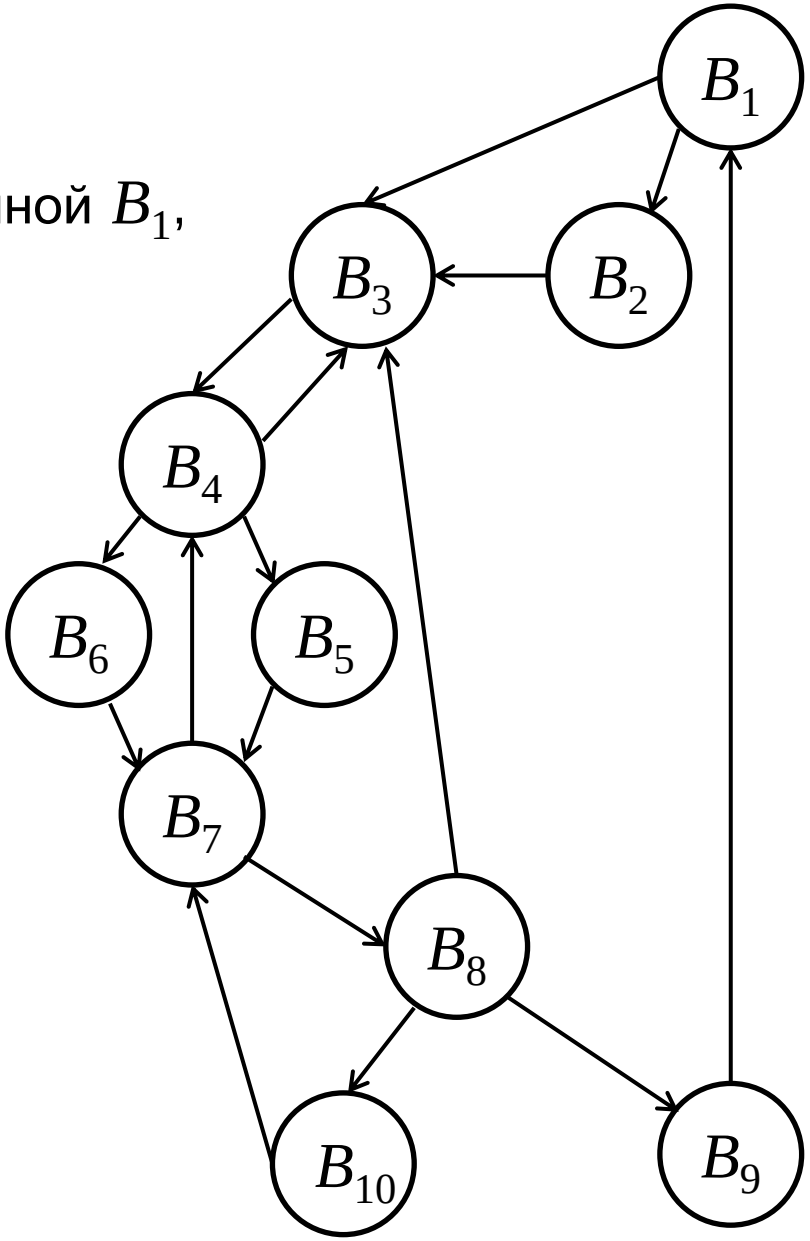
- B_1 является доминатором всех узлов, включая себя самого.
- B_2 является доминатором только себя самого.
- B_3 является доминатором всех вершин, кроме B_1 и B_2.
- B_4 является доминатором всех вершин, кроме B_1, B_2 и B_3.
- B_5 и B_6 являются доминаторами только себя.

4.1 Доминаторы

4.1.2 Примеры доминаторов

◊ Рассмотрим ГПУ с входной вершиной B_1, показанный на рисунке.

- B_7 является доминатором вершин B_7, B_8, B_9 и B_{10}.
- B_8 является доминатором вершин B_8, B_9 и B_{10}.
- B_9 и B_{10} являются доминаторами только себя.



4.1 Доминаторы

4.1.3 Свойства отношения *dom*

◊ 1. Отношение dom рефлексивно, антисимметрично и транзитивно, т.е. является отношением частичного порядка.

(1) *Рефлексивность*: $a\ dom\ a$.

(2) *Антисимметрчность*: если $a\ dom\ b$ и $b\ dom\ a$, то $a = b$.

(3) *Транзитивность*: если $a\ dom\ b$ и $b\ dom\ c$, то $a\ dom\ c$.

◊ 2. Для любой вершины n ГПУ каждый ациклический путь от $Entry$ до n проходит через все доминаторы n, причем на всех таких путях доминаторы проходятся в *одном и том же порядке*

4.1 Доминаторы

4.1.3 Свойства отношения *dom*

◊ 3. Вершина i ГПУ является *непосредственным доминатором* вершины n (i *idom* n), если

(1) i *dom* n

(2) не существует вершины m, $m \neq i$, $m \neq n$, такой что i *dom* m и m *dom* n.

◊ 4. У каждой вершины n за исключением *Entry* существует единственный непосредственный доминатор.

◊ 5. Вершина s ГПУ является *строгим доминатором* вершины n (s *sdom* n), если s *dom* n и $s \neq n$.

4.1 Доминаторы

4.1.3 Свойства отношения *dom*

◊ **6** Пусть n – вершина ГПУ,

$Pred(n) = \{p_1, p_2, \ldots, p_k\}$ и $d \neq n$.

Тогда $d\ dom\ n$ тогда и только тогда, когда $\forall\ i\ \ d\ dom\ p_i$.

◊ **7** Множество строгих доминаторов вершины n является пересечением множеств доминаторов всех ее предшественников.

4.1 Доминаторы

4.1.4 Алгоритм вычисления доминаторов

◊ **Задача поиска всех доминаторов** вершин ГПУ формулируется как задача анализа потока данных в прямом направлении.

Значением потока данных на входе в блок B является множество вершин (базовых блоков), являющихся доминаторами B.
Операцией сбора является операция пересечения множеств.
Передаточная функция f_B добавляет вершину B к рассматриваемому множеству вершин.
Граничное условие: единственным доминатором вершины $Entry$ является она сама.

4.1 Доминаторы

4.1.4 Алгоритм вычисления доминаторов

◊ **Алгоритм**

Вход: граф потока $G = \langle N, E \rangle$ с входным узлом $Entry$.

Выход: для каждой вершины $n \in N$ множество $D(n)$ ее доминаторов.

Метод: найти решение следующей задачи потока данных (вершины n соответствуют базовым блокам):

$$\forall n \in N \qquad D(n) = Out[Pred(n)].$$

4.1 Доминаторы

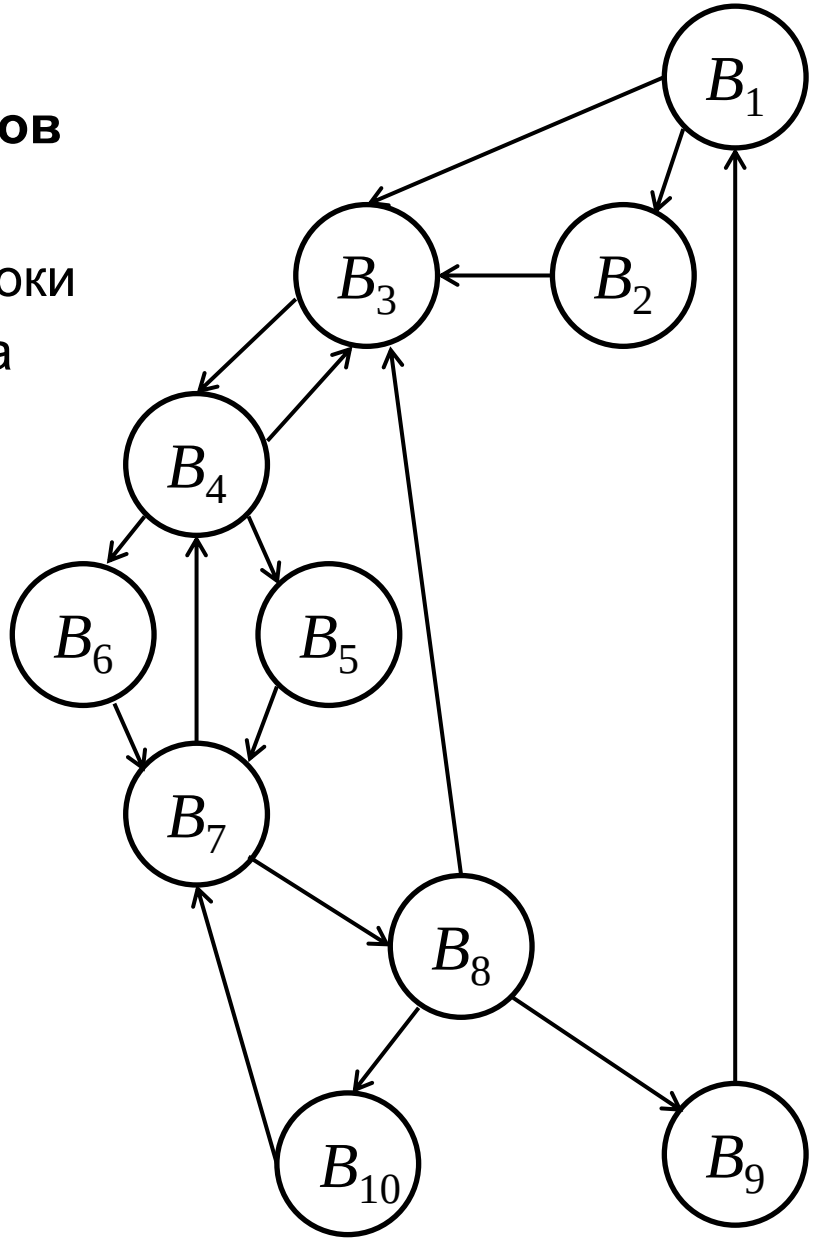
4.1.4 Алгоритм вычисления доминаторов

Область определения	Множество подмножеств базовых блоков
Направление обхода	$Forward$
Передаточная функция	$f_B(x) = x \cup \{B\}$
Граничное условие	$Out\,[Entry] = Entry$
Операция сбора ($\wedge$)	$\cap$
Система уравнений	$Out[B] = f_B(In[B])$ $In[B] = \bigcap_{P \in Pred(B)} Out[P]$
Начальное приближение	$Out\,[B] = N$

4.1 Доминаторы

4.1.4 Алгоритм вычисления доминаторов

◊ **Пример.** Применим алгоритм к ГПУ на рисунке в предположении, что блоки посещаются в порядке их номеров, а $Entry$ это блок B_1.



◊ **Первая итерация**

Граничное условие: $\boldsymbol{D(B_1) = \{B_1\}}$

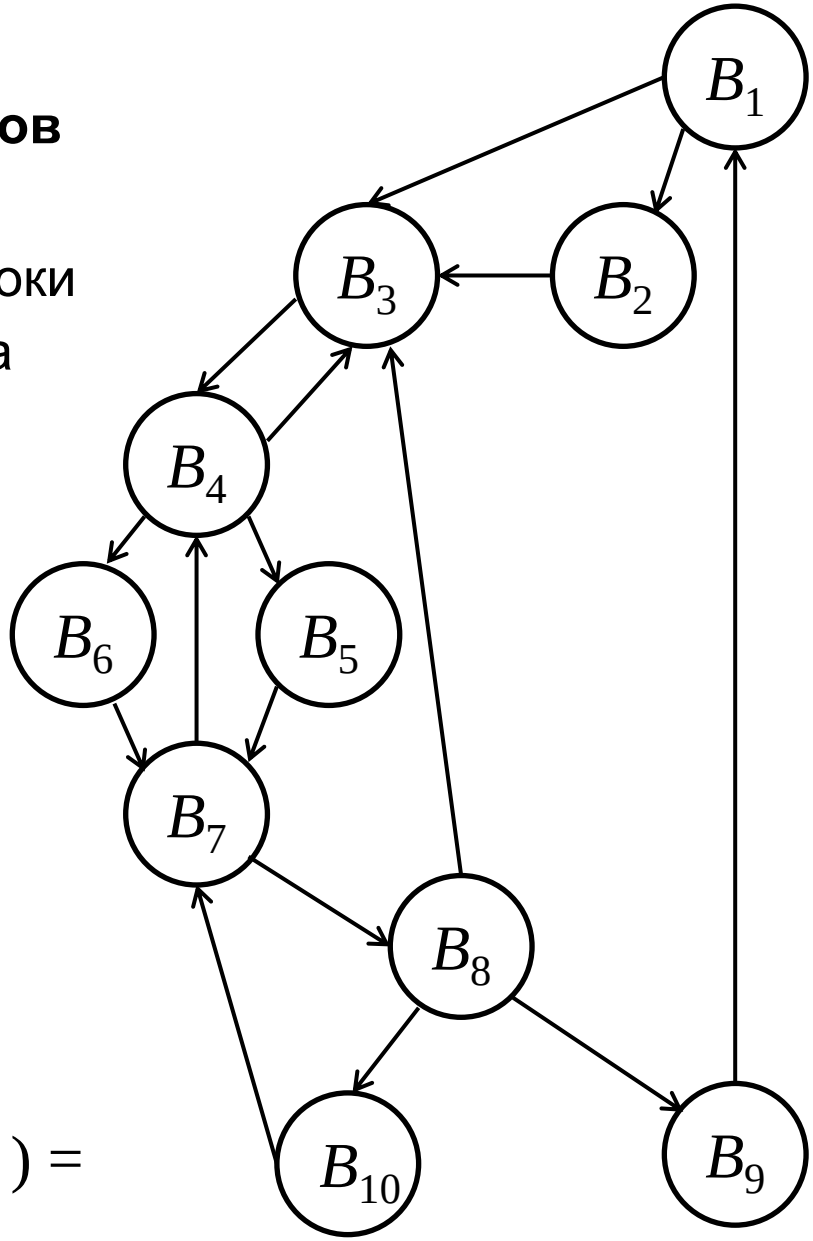
$Pred(B_2) = \{B_1\}$

$\boldsymbol{D(B_2) = \{B_2\} \cup D(B_1) = \{B_1, B_2\}}$

4.1 Доминаторы

4.1.4 Алгоритм вычисления доминаторов

◊ **Пример.** Применим алгоритм к ГПУ на рисунке в предположении, что блоки посещаются в порядке их номеров, а $Entry$ это блок B_1.



◊ **Первая итерация**

Граничное условие: $\boldsymbol{D(B_1) = \{B_1\}}$

$Pred(B_2) = \{B_1\}$

$\boldsymbol{D(B_2) = \{B_2\} \cup D(B_1) = \{B_1, B_2\}}$

$Pred(B_3) = \{B_1, B_2, B_4, B_8\}$

$\boldsymbol{D(B_3) = \{B_3\} \cup (D(B_1) \cap D(B_2) \cap D(B_4) \cap D(B_8))} = \{B_3\} \cup (\{B_1\} \cap \{B_1, B_2\} \cap N \cap N) = \{B_1, B_3\}$

$N = \{B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8, B_9, B_{10}\}$

4.1 Доминаторы
4.1.4 Алгоритм вычисления доминаторов

◊ Первая итерация (окончание)

$Pred(B_4) = \{B_3, B_7\}$

$\boldsymbol{D(B_4) = \{B_4\} \cup (D(B_3) \cap D(B_7)) = \{B_4\} \cup (\{B_1, B_3\} \cap N) =}$

$\boldsymbol{= \{B_1, B_3, B_4\}}$

4.1 Доминаторы
4.1.4 Алгоритм вычисления доминаторов

◊ Первая итерация (окончание)

$Pred(B_4) = \{B_3, B_7\}$

$\boldsymbol{D(B_4) = \{B_4\} \cup (D(B_3) \cap D(B_7)) = \{B_4\} \cup (\{B_1, B_3\} \cap N) =}$

$\boldsymbol{= \{B_1, B_3, B_4\}}$

$Pred(B_5) = Pred(B_6) = \{B_4\}$

$\boldsymbol{D(B_5) = \{B_5\} \cup D(B_4) = \{B_5\} \cup \{B_1, B_3, B_4\} = \{B_1, B_3, B_4, B_5\}}$

$\boldsymbol{D(B_6) = \{B_6\} \cup D(B_4) = \{B_6\} \cup \{B_1, B_3, B_4\} = \{B_1, B_3, B_4, B_6\}}$

4.1 Доминаторы
4.1.4 Алгоритм вычисления доминаторов

◊ Первая итерация (окончание)

$Pred(B_4) = \{B_3, B_7\}$

$\boldsymbol{D(B_4) = \{B_4\} \cup (D(B_3) \cap D(B_7)) = \{B_4\} \cup (\{B_1, B_3\} \cap N) =}$

$\boldsymbol{= \{B_1, B_3, B_4\}}$

$Pred(B_5) = Pred(B_6) = \{B_4\}$

$\boldsymbol{D(B_5) = \{B_5\} \cup D(B_4) = \{B_5\} \cup \{B_1, B_3, B_4\} = \{B_1, B_3, B_4, B_5\}}$

$\boldsymbol{D(B_6) = \{B_6\} \cup D(B_4) = \{B_6\} \cup \{B_1, B_3, B_4\} = \{B_1, B_3, B_4, B_6\}}$

$Pred(B_7) = \{B_5, B_6, B_{10}\}$

$\boldsymbol{D(B_7) = \{B_7\} \cup (D(B_5) \cap D(B_6)) \cap D(B_{10})) = \{B_7\} \cup (\{B_1, B_3, B_4, B_5\} \cap}$

$\boldsymbol{= \{B_1, B_3, B_4, B_6\} \cap N) = \{B_1, B_3, B_4, B_7\}}$

$Pred(B_8) = \{B_7\}$

4.1 Доминаторы
4.1.4 Алгоритм вычисления доминаторов

◊ Первая итерация (окончание)

$Pred(B_4) = \{B_3, B_7\}$

$\boldsymbol{D(B_4) = \{B_4\} \cup (D(B_3) \cap D(B_7)) = \{B_4\} \cup (\{B_1, B_3\} \cap N) =}$

$\boldsymbol{= \{B_1, B_3, B_4\}}$

$Pred(B_5) = Pred(B_6) = \{B_4\}$

$\boldsymbol{D(B_5) = \{B_5\} \cup D(B_4) = \{B_5\} \cup \{B_1, B_3, B_4\} = \{B_1, B_3, B_4, B_5\}}$

$\boldsymbol{D(B_6) = \{B_6\} \cup D(B_4) = \{B_6\} \cup \{B_1, B_3, B_4\} = \{B_1, B_3, B_4, B_6\}}$

$Pred(B_7) = \{B_5, B_6, B_{10}\}$

$\boldsymbol{D(B_7) = \{B_7\} \cup (D(B_5) \cap D(B_6)) \cap D(B_{10})) = \{B_7\} \cup (\{B_1, B_3, B_4, B_5\} \cap}$

$\boldsymbol{= \{B_1, B_3, B_4, B_6\} \cap N) = \{B_1, B_3, B_4, B_7\}}$

$Pred(B_8) = \{B_7\}$

$\boldsymbol{D(B_8) = \{B_8\} \cup D(B_7) = \{B_8\} \cup \{B_1, B_3, B_4, B_7\} = \{B_1, B_3, B_4, B_7, B_8\}}$

$Pred(B_9) = Pred(B_{10}) = \{B_8\}$

$\boldsymbol{D(B_9) = \{B_9\} \cup D(B_8) = \{B_9\} \cup \{B_1, B_3, B_4, B_7, B_8\} = \{B_1, B_3, B_4, B_7, B_8, B_9\}}$

$\boldsymbol{D(B_{10}) = \{B_{10}\} \cup D(B_8) = \{B_{10}\} \cup \{B_1, B_3, B_4, B_7, B_8\} =}$

$\boldsymbol{= \{B_1, B_3, B_4, B_7, B_8, B_{10}\}}$

4.1 Доминаторы
4.1.4 Алгоритм вычисления доминаторов

◊ Первая итерация (окончание)

$Pred(B_4) = \{B_3, B_7\}$

$\boldsymbol{D(B_4) = \{B_4\} \cup (D(B_3) \cap D(B_7)) = \{B_4\} \cup (\{B_1, B_3\} \cap N) =}$

$\boldsymbol{= \{B_1, B_3, B_4\}}$

$Pred(B_5) = Pred(B_6) = \{B_4\}$

$\boldsymbol{D(B_5) = \{B_5\} \cup D(B_4) = \{B_5\} \cup \{B_1, B_3, B_4\} = \{B_1, B_3, B_4, B_5\}}$

$\boldsymbol{D(B_6) = \{B_6\} \cup D(B_4) = \{B_6\} \cup \{B_1, B_3, B_4\} = \{B_1, B_3, B_4, B_6\}}$

$Pred(B_7) = \{B_5, B_6\}$

$\boldsymbol{D(B_7) = \{B_7\} \cup (D(B_5) \cap D(B_6)) = \{B_7\} \cup (\{B_1, B_3, B_4, B_5\} \cap \{B_1, B_3, B_4, B_6\}) \cap N) = \{B_1, B_3, B_4, B_7\}}$

$Pred(B_8) = \{B_7\}$

$\boldsymbol{D(B_8) = \{B_8\} \cup D(B_7) = \{B_8\} \cup \{B_1, B_3, B_4, B_7\} = \{B_1, B_3, B_4, B_7, B_8\}}$

$Pred(B_9) = Pred(B_{10}) = \{B_8\}$

$\boldsymbol{D(B_9) = \{B_9\} \cup D(B_8) = \{B_9\} \cup \{B_1, B_3, B_4, B_7, B_8\} = \{B_1, B_3, B_4, B_7, B_8, B_9\}}$

$\boldsymbol{D(B_{10}) = \{B_{10}\} \cup D(B_8) = \{B_{10}\} \cup \{B_1, B_3, B_4, B_7, B_8\} =}$

$\boldsymbol{= \{B_1, B_3, B_4, B_7, B_8, B_{10}\}}$

Полученные значения $D(B_1) - D(B_{10})$ на второй итерации не изменяются

4.1 Доминаторы

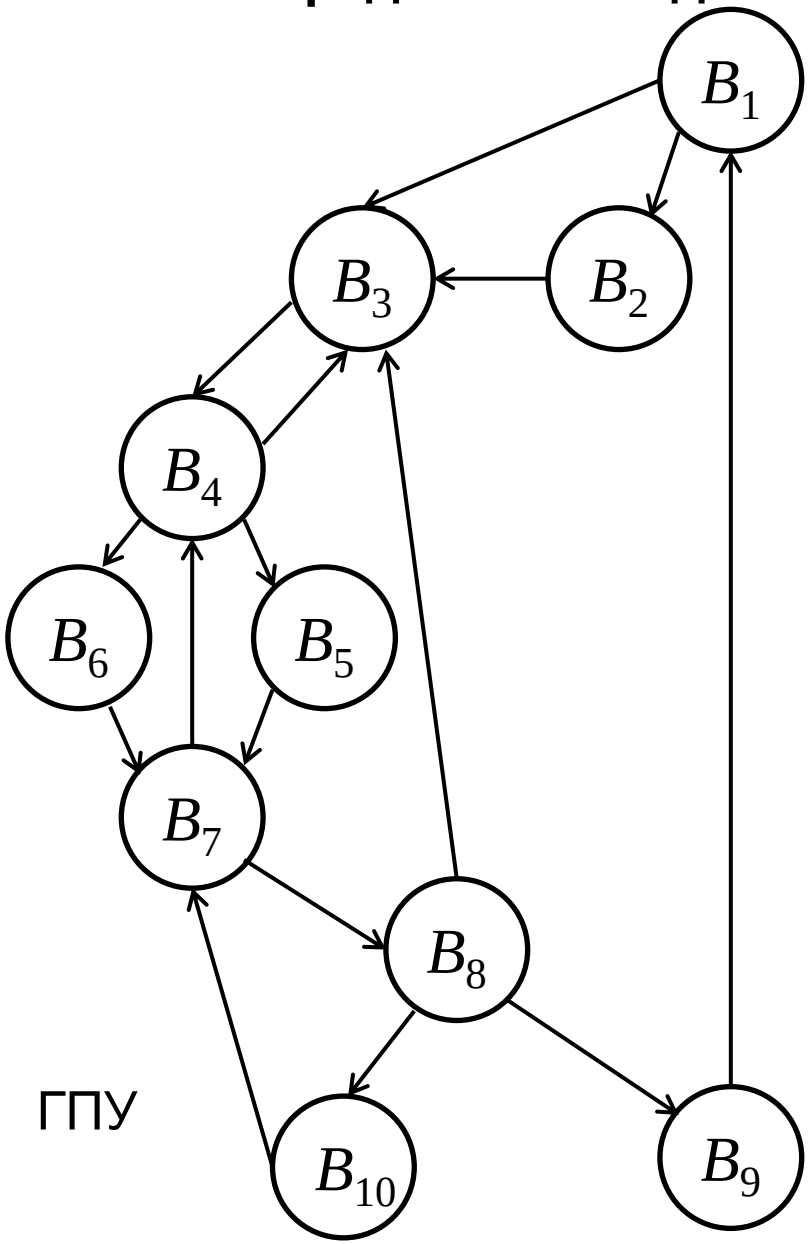
4.1.5 Непосредственные доминаторы и дерево доминаторов

n	$D(n)$	$IDom(n)$
B_1	B_1	–
B_2	$\boldsymbol{B_1}$, B_2	B_1
B_3	$\boldsymbol{B_1}$, B_3	B_1
B_4	B_1, $\boldsymbol{B_3}$, B_4	B_3
B_5	B_1, B_3, $\boldsymbol{B_4}$, B_5	B_4
B_6	B_1, B_3, $\boldsymbol{B_4}$, B_6	B_4
B_7	B_1, B_3, $\boldsymbol{B_4}$, B_7	B_4
B_8	B_1, B_3, B_4, $\boldsymbol{B_7}$, B_8	B_7
B_9	B_1, B_3, B_4, B_7, $\boldsymbol{B_8}$, B_9	B_8
B_{10}	B_1, B_3, B_4, B_7, $\boldsymbol{B_8}$, B_{10}	B_8

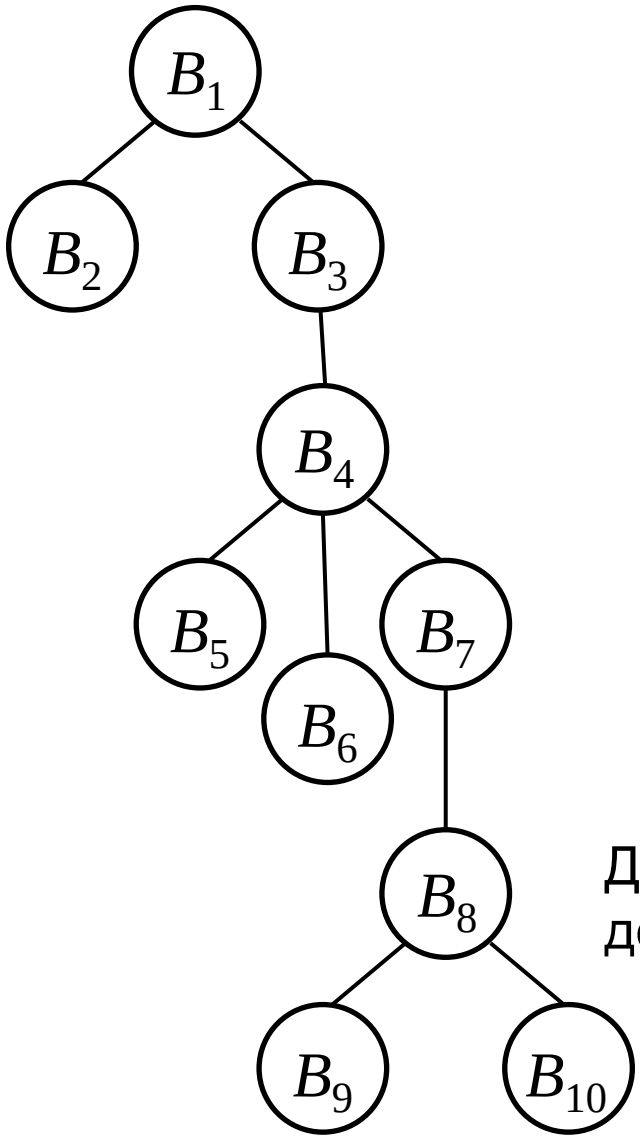
В таблице приведены списки доминаторов каждой вершины ГПУ из рассмотренного примера. Непосредственный доминатор в каждом списке предпоследний. Соединив дугами для каждого $n \in N$ $IDom(n)$ с n, получим дерево доминаторов. Оно изображено на следующем слайде

4.1 Доминаторы

4.1.5 Непосредственные доминаторы и дерево доминаторов



ГПУ



Дерево доминаторов

4.2 Форма статического единственного присваивания (SSA-форма)

4.2.1. Постановка задачи

◊ *Форма статического единственного присваивания* (*SSA*) позволяет в каждой точке программы объединить

- информацию об имени переменной
- с информацией о текущем значении этой переменной (или, что то же самое, с информацией о том, какое из определений данной переменной определяет ее текущее значение в данной точке).

4.2 Форма статического единственного присваивания (*SSA-форма*)

4.2.1. Постановка задачи

◊ *Форма статического единственного присваивания* (*SSA*) позволяет в каждой точке программы объединить

- информацию об имени переменной
- с информацией о текущем значении этой переменной (или, что то же самое, с информацией о том, какое из определений данной переменной определяет ее текущее значение в данной точке).

◊ Хотелось бы, чтобы программа в *SSA*-форме удовлетворяла двум условиям:

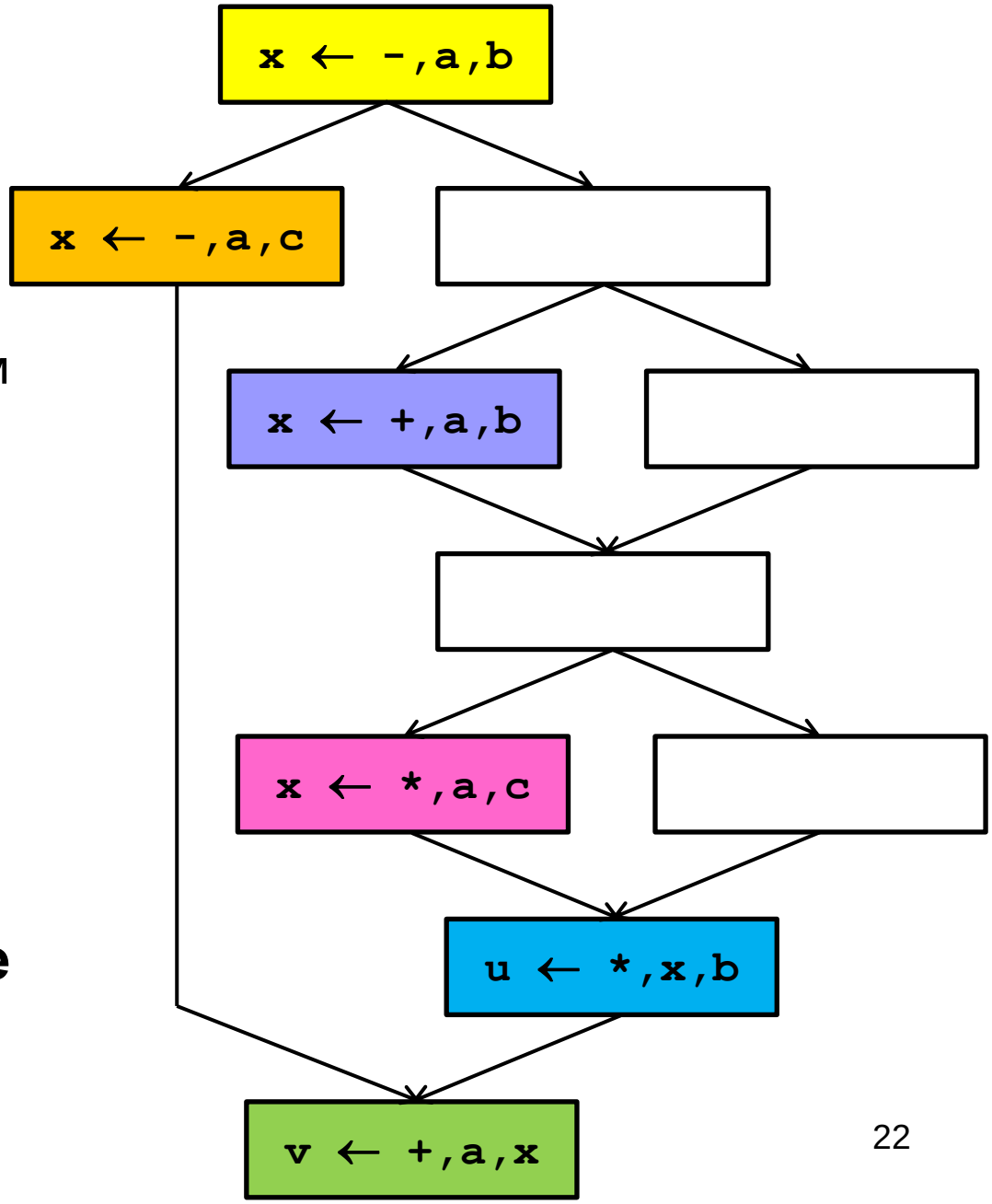
(1) каждое определение переменной имеет индивидуальное имя;

(2) каждое использование переменной ссылается на единственное определение.

4.2 SSA-форма

4.2.1. Постановка задачи

- Использования в синем блоке достигают три определения **x**, в зеленом – четыре определения **x**
- Цель же состоит в том, чтобы **каждого использования достигало только одно определение**
- Введем «функцию» объединения значений или *ϕ-функцию*



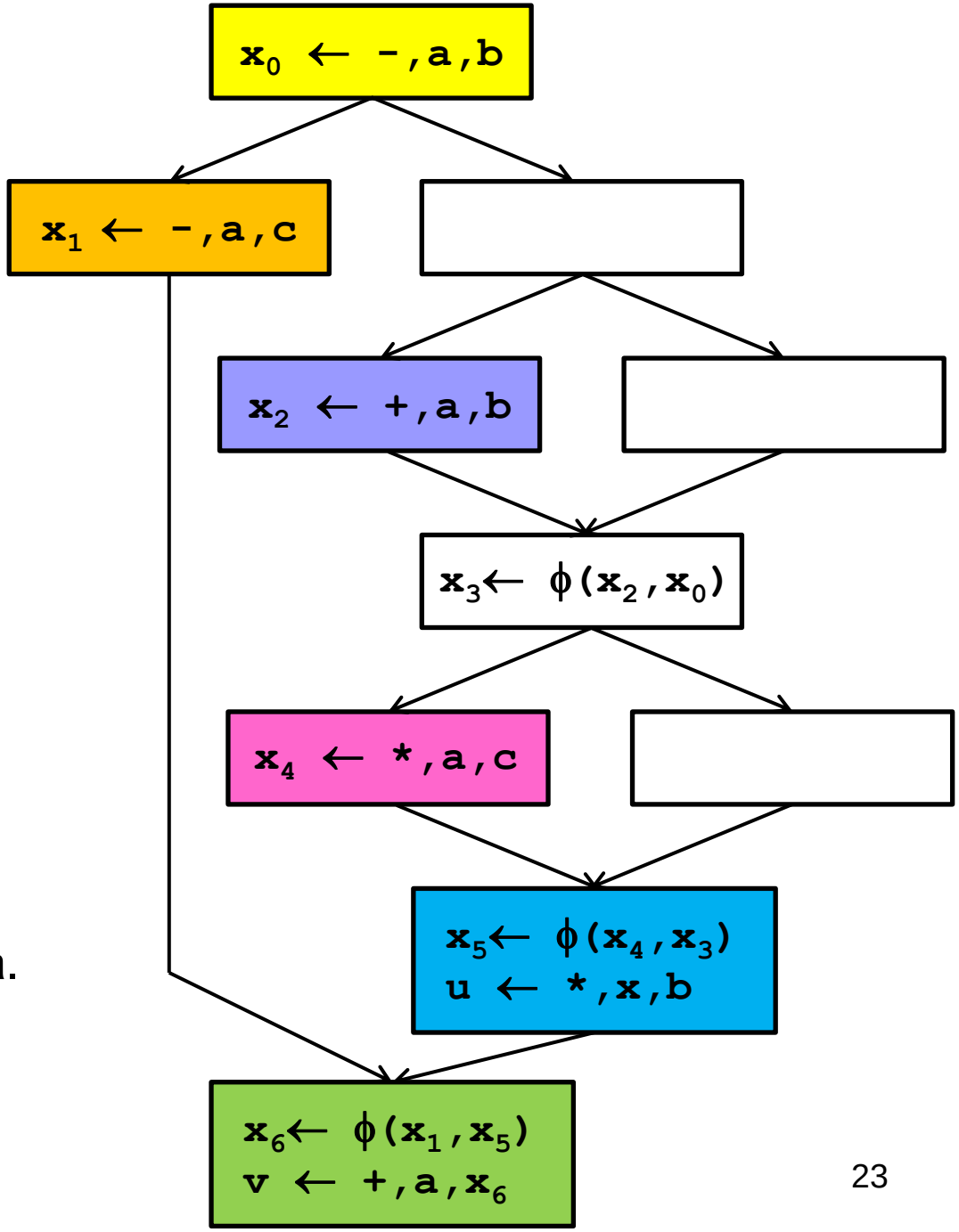
4.2 SSA-форма

4.2.1. Постановка задачи

◊ По определению $x_3 \leftarrow \phi(x_2, x_0)$ является новым определением переменной x:

значение x_3 равно x_2, когда управление попадает в блок слева,

значение x_3 равно x_0, когда управление попадает в блок справа.



4.2 SSA-форма

4.2.2. Определение ϕ-функции

- ◊ ϕ-функция определяет *SSA-имя* для значения своего аргумента, соответствующего ребру, по которому управление входит в блок.
- ◊ **При входе в базовый блок все его ϕ-функции выполняются одновременно и до любого другого оператора, определяя целевые *SSA*-имена.**

4.2 SSA-форма

4.2.2. Определение ϕ-функции. Пример

```
x =
y = …
while (x < 100){
    x = x + 1
    y = y + x
}
```

4.2 SSA-форма

4.2.2. Определение ϕ-функции. Пример

```
x =
y = …
while (x < 100){
   x = x + 1
   y = y + x
}
```

```
      x0 = …
      y0 = …
      if (x0 ≥ 100) goto next
loop: x1 = φ(x0, x2)
      y1 = φ(y0, y2)
      x2 = x1 + 1
      y2 = y1 + x2
      if (x2 < 100) goto loop
next: x3 = φ(x0, x2)
      y3 = φ(y0, y2)
```

4.2 SSA-форма

4.2.2. Определение ϕ-функции. Пример

```
x =
y = …
while (x < 100){
   x = x + 1
   y = y + x
}
```

```
      x0 = …
      y0 = …
      if (x0 ≥ 100) goto next
loop: x1 = φ(x0, x2)
      y1 = φ(y0, y2)
      x2 = x1 + 1
      y2 = y1 + x2
      if (x2 < 100) goto loop
next: x3 = φ(x0, x2)
      y3 = φ(y0, y2)
```

Каждая из ϕ-функций объединяет определения (значения) своих аргументов в новое значение, определением которого она является.

4.2 SSA-форма

4.2.2. Определение ϕ-функции. Пример

```
x =
y = …
while (x < 100){
   x = x + 1
   y = y + x
}
```

```
      x0 = …
      y0 = …
      if (x0 ≥ 100) goto next
loop: x1 = φ(x0, x2)
      y1 = φ(y0, y2)
      x2 = x1 + 1
      y2 = y1 + x2
      if (x2 < 100) goto loop
next: x3 = φ(x0, x2)
      y3 = φ(y0, y2)
```

Каждая из ϕ-функций объединяет определения (значения) своих аргументов в новое значение, определением которого она является.

До цикла –	x_0, y_0
На входе в цикл –	x_1, y_1
Внутри цикла –	x_2, y_2
После цикла –	x_3, y_3

4.2 SSA-форма

4.2.2. Определение ϕ-функции. Пример

```
x =
y = …
while (x < 100){
   x = x + 1
   y = y + x
}
```

```
      x0 = …
      y0 = …
      if (x0 ≥ 100) goto next
loop: x1 = φ(x0, x2)
      y1 = φ(y0, y2)
      x2 = x1 + 1
      y2 = y1 + x2
      if (x2 < 100) goto loop
next: x3 = φ(x0, x2)
      y3 = φ(y0, y2)
```

◊ **Затруднение**. Первый аргумент $\varphi(x_0, x_2)$ определяется в блоке, который предшествует циклу, второй аргумент определяется позже в блоке, содержащем $\varphi(x_0, x_2)$. Следовательно, при первом выполнении $\varphi(x_0, x_2)$ ее второй аргумент еще не определен.

4.2 SSA-форма

4.2.2. Определение ϕ-функции. Пример

```
x =
y = …
while (x < 100){
   x = x + 1
   y = y + x
}
```

```
      x0 = …
      y0 = …
      if (x0 ≥ 100) goto next
loop: x1 = φ(x0, x2)
      y1 = φ(y0, y2)
      x2 = x1 + 1
      y2 = y1 + x2
      if (x2 < 100) goto loop
next: x3 = φ(x0, x2)
      y3 = φ(y0, y2)
```

◊ **Выход из затруднения**: по определению любая ϕ-функция (а значит и $\varphi(x_0, x_2)$) **читает только один из своих аргументов**, а именно тот аргумент, который соответствует самому последнему пройденному ребру ГПУ.

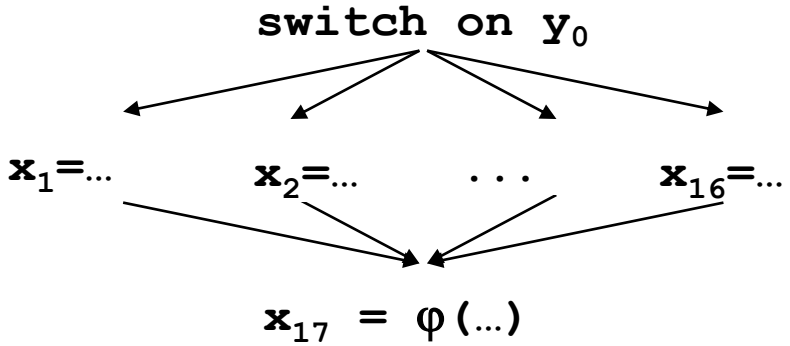
Поэтому ϕ-функция никогда не прочитает неопределенного значения.

4.2 SSA-форма

4.2.3. Количество аргументов ϕ-функции

◊ По определению у ϕ **-функции может быть любое число аргументов**.

Например, на рисунке ϕ-функция, у которой 16 аргументов



4.3 Построение SSA-формы

4.3.1. Постановка задачи

◊ Для преобразования процедуры в SSA-форму **компилятор** должен:

- ⧫ вставить в точки сбора необходимые ϕ-функции для каждой переменной
- ⧫ переименовать переменные (в том числе и временные) таким образом, чтобы выполнялись следующие два правила:
  (1) каждое определение имеет индивидуальное имя; и
  (2) каждое использование ссылается на единственное определение.

4.3 Построение SSA-формы

4.3.2. Базовый алгоритм построения SSA-формы

◊ **Вход**: программа в промежуточном представлении

◊ **Выход**: промежуточное представление программы в SSA-форме

◊ **Метод**: Выполнить следующие действия:

(1) $Вставить\ \phi\text{-}функции$:

в начало каждого блока B, у которого $|Pred(B)| > 1$ вставить ϕ-функцию вида $y = \phi(y, y, ...)$ для каждого имени y, которое либо определяется, либо используется в B.

Вставленная ϕ-функция должна иметь по одному аргументу для каждого $B' \in Pred(B)$:

Порядок вставляемых ϕ-функций несуществен

4.3 Построение SSA-формы

4.3.2. Базовый алгоритм построения *SSA*-формы

◊ **Вход**: программа в промежуточном представлении

◊ **Выход**: промежуточное представление программы в *SSA*-форме

◊ **Метод**: Выполнить следующие действия:

(2) *Переименовать переменные*:

(a) вычислить достигающие определения; при этом только одно определение будет достигать любого использования:

это гарантируют вставленные ϕ-функции, которые тоже являются определениями;

(b) переименовать каждое использование (как основных, так и временных) переменных таким образом, чтобы новое имя соответствовало единственному определению, которое достигает его.

4.3 Построение SSA-формы

4.3.2. Базовый алгоритм построения SSA-формы

◊ **Вход**: программа в промежуточном представлении

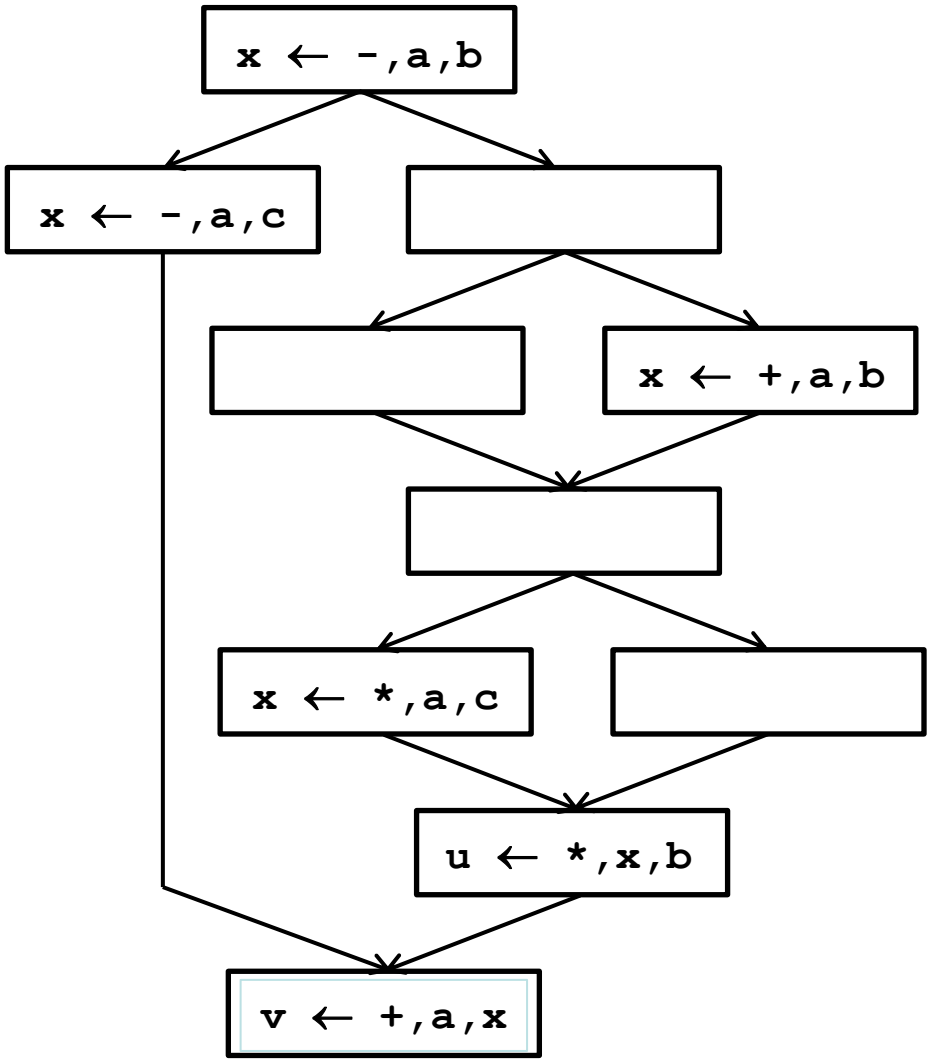
◊ **Выход**: промежуточное представление программы в SSA-форме

◊ **Метод**: Выполнить следующие действия:

(3) *Отсортировать определения*, достигающие каждой ϕ-функции и для каждой ϕ-функции обеспечить соответствие имен ее аргументов путям, по которым определения этих аргументов достигают блока, содержащего указанную ϕ-функцию.

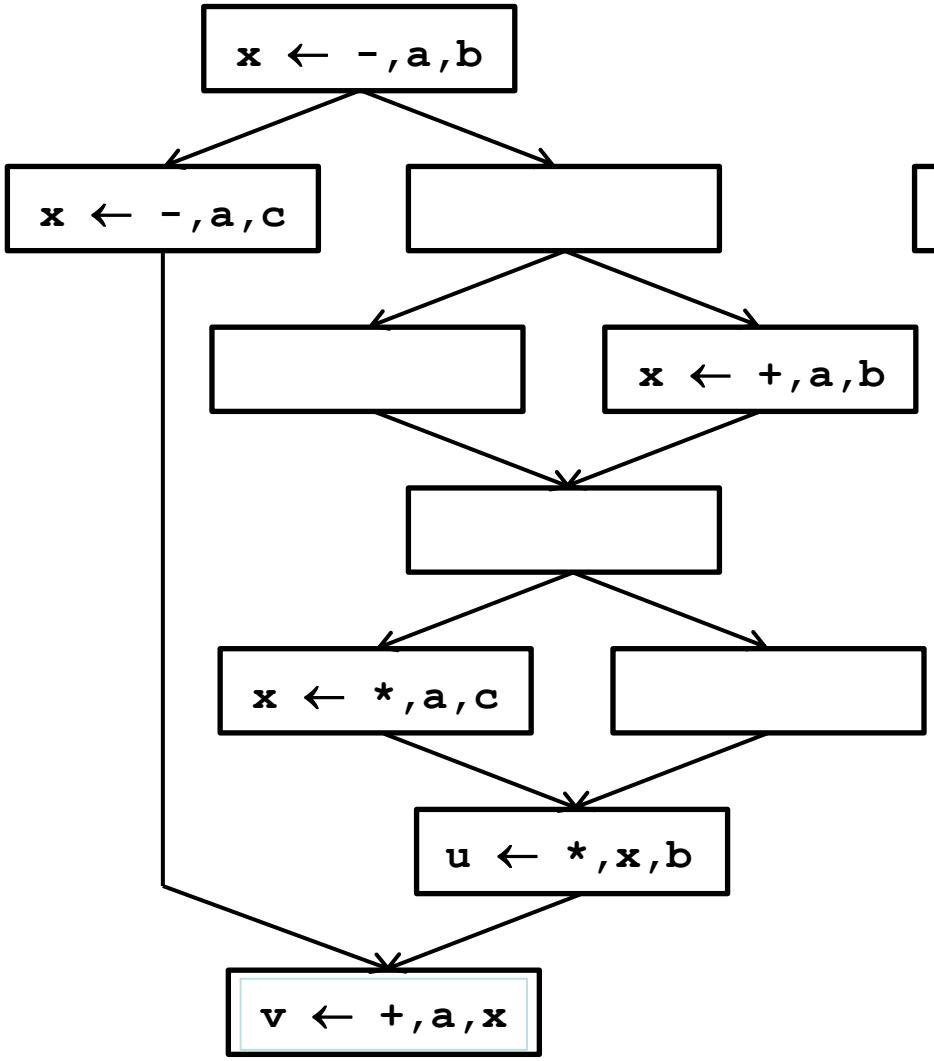
4.3 Построение SSA-формы

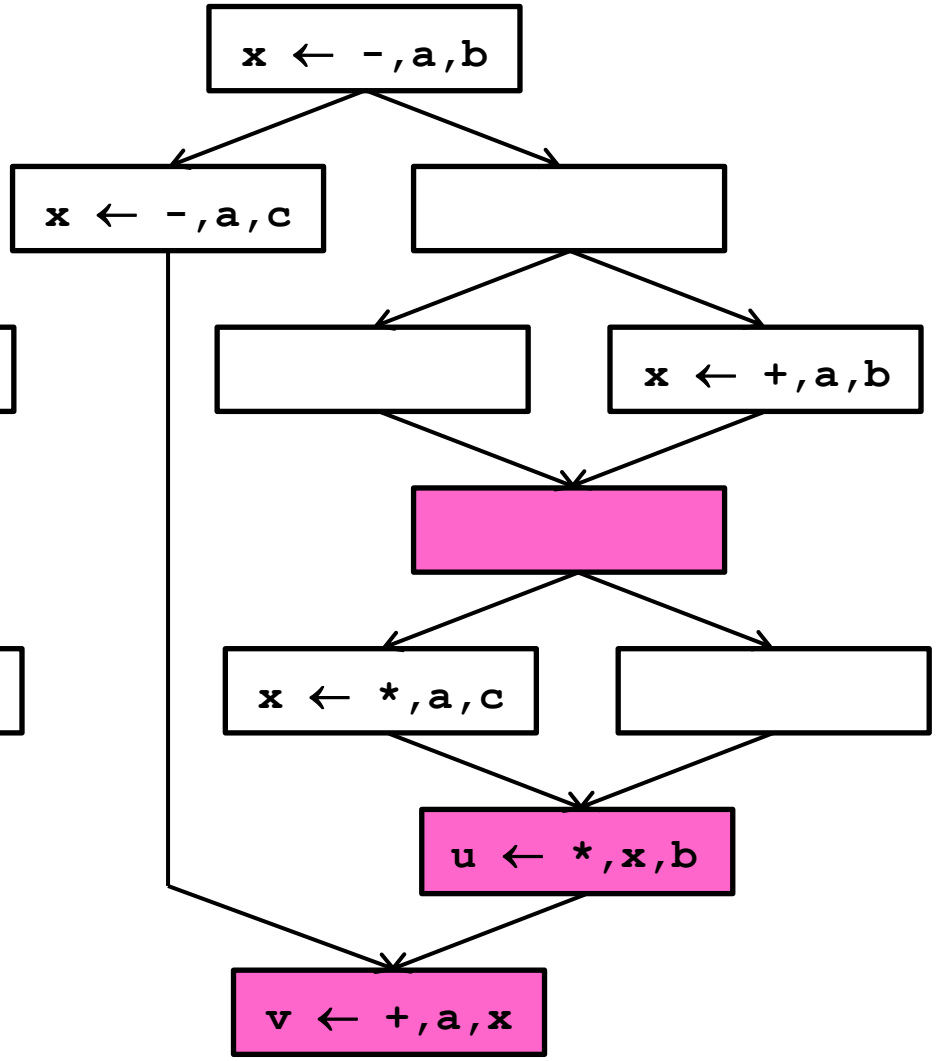
4.3.3. Базовый алгоритм построения SSA-формы. Пример



4.3 Построение SSA-формы

4.3.3. Базовый алгоритм построения SSA-формы. Пример





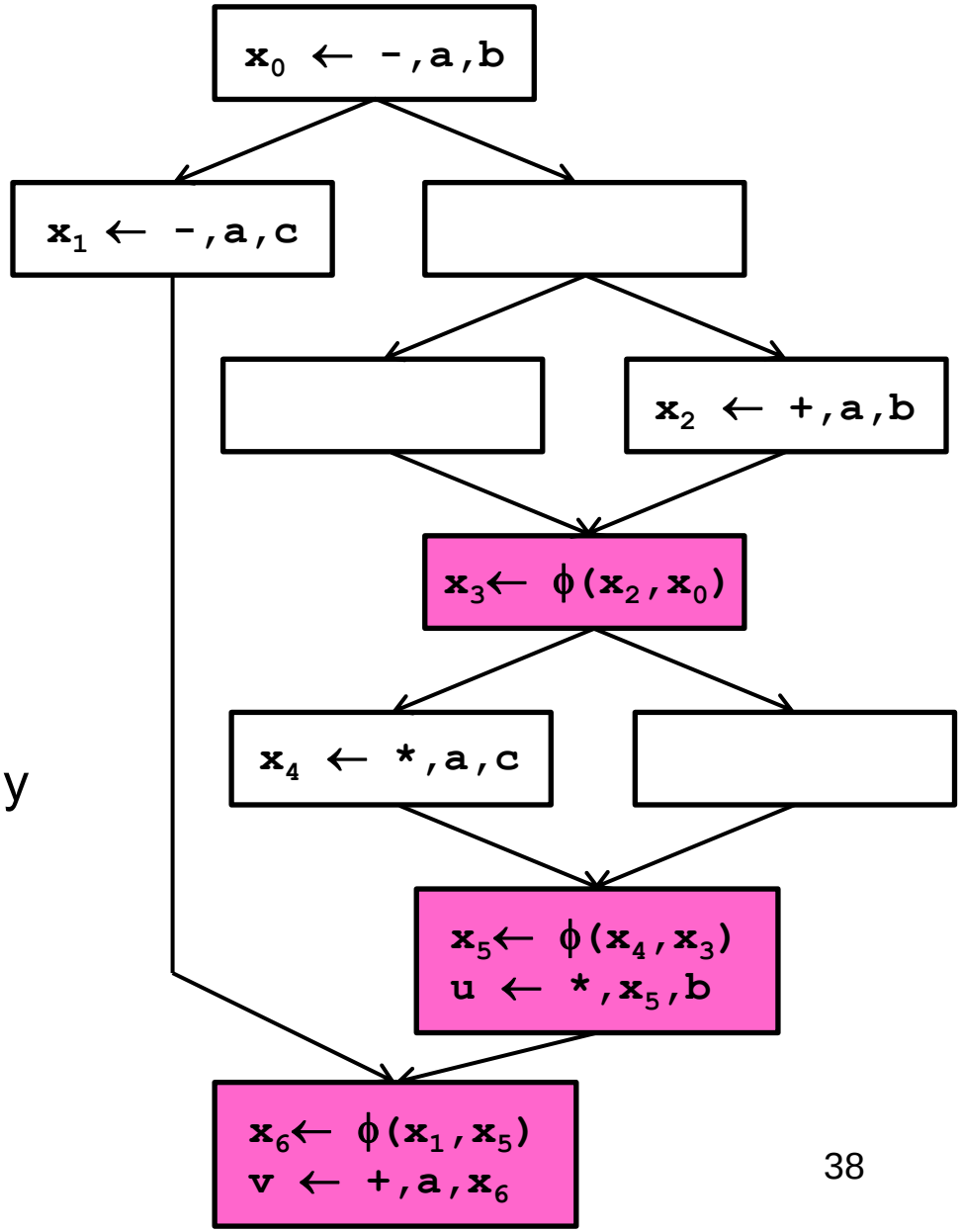
4.3 Построение SSA-формы

4.3.3. Базовый алгоритм построения SSA-формы. Пример

- ◊ Алгоритм вставил ϕ-функцию после каждой точки сбора ГПУ.
- ◊ После преобразования процедуры в SSA-форму

  (1) внутри процедуры каждое определение создает уникальное имя;

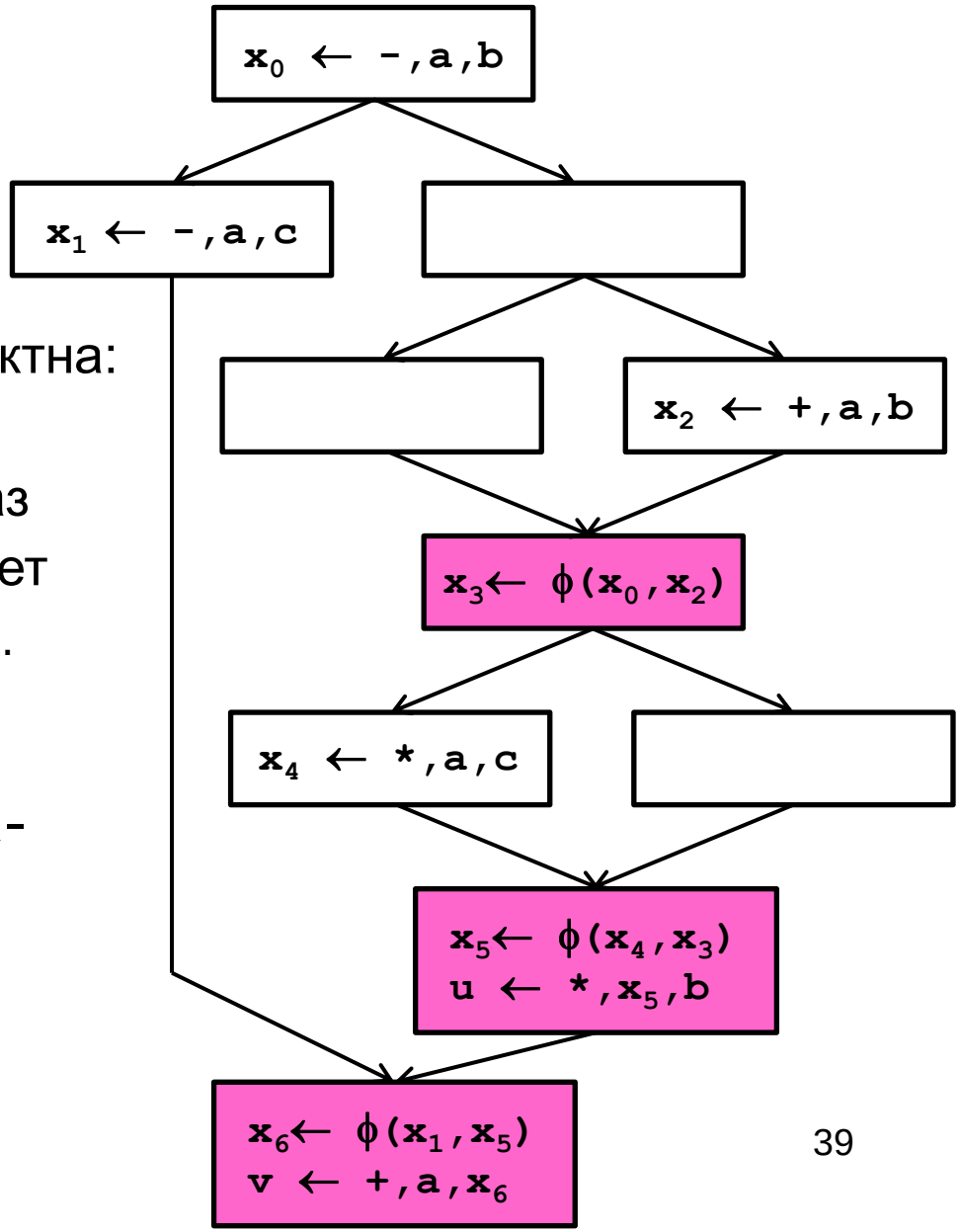
  (2) каждое использование обращается к единственному определению.



4.3 Построение SSA-формы

4.3.3. Базовый алгоритм построения *SSA*-формы. Пример

- Построенная *SSA-форма* корректна:
  - Каждая переменная определяется только один раз
  - Каждое обращение использует имя отдельного определения.
- Построенная *SSA-форма* называется *максимальной SSA-формой,* так как она обычно содержит намного больше ϕ-функций, чем необходимо.



4.4 Построение частично усеченной SSA-формы

4.4.1. Постановка задачи

◊ По определению *частично усеченная SSA-форма* содержит меньше ϕ-функций, чем максимальная (но, к сожалению, как следует и из ее названия она содержит не минимально возможное количество ϕ-функций).

4.4 Построение частично усеченной SSA-формы

4.4.1. Постановка задачи

◊ По определению *частично усеченная SSA-форма* содержит меньше ϕ-функций, чем максимальная (но, к сожалению, как следует и из ее названия она содержит не минимально возможное количество ϕ-функций).

◊ Чтобы не всегда вставлять ϕ-функции необходимо **для каждой точки сбора** уметь выяснять, какие переменные нуждаются в ϕ-функциях.

4.4 Построение частично усеченной SSA-формы

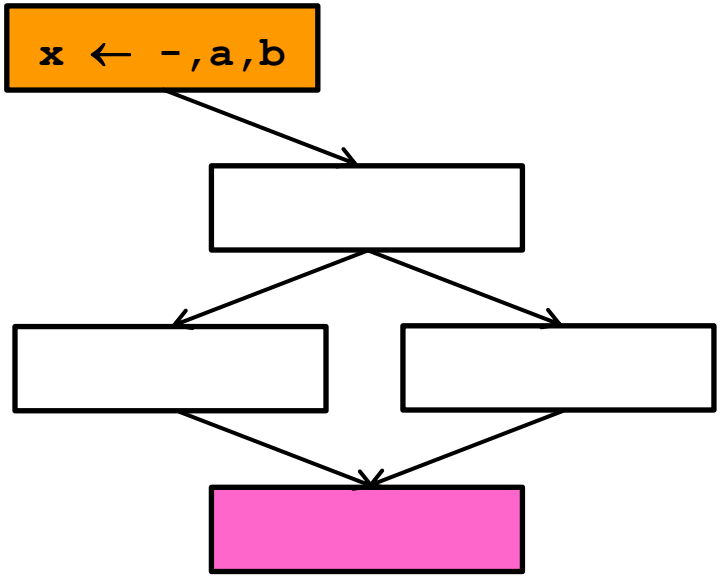
4.4.1. Постановка задачи

◊ По определению *частично усеченная SSA-форма* содержит меньше ϕ-функций, чем максимальная (но, к сожалению, как следует и из ее названия она содержит не минимально возможное количество ϕ-функций).

◊ Чтобы не всегда вставлять ϕ-функции необходимо
для каждой точки сбора уметь выяснять, какие переменные нуждаются в ϕ-функциях.
Или
для каждого определения переменной уметь находить множество всех точек сбора, которые нуждаются в ϕ-функциях для значения, порожденного этим определением.

4.4 Построение частично усеченной SSA-формы

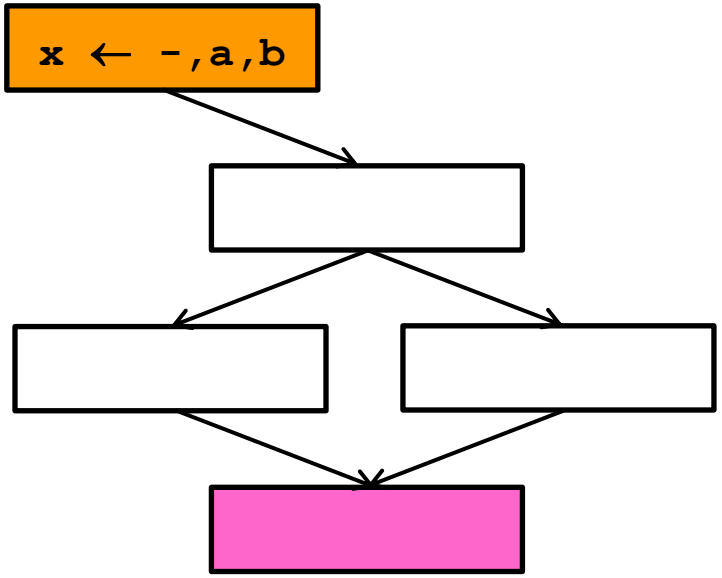
4.4.1. Постановка задачи



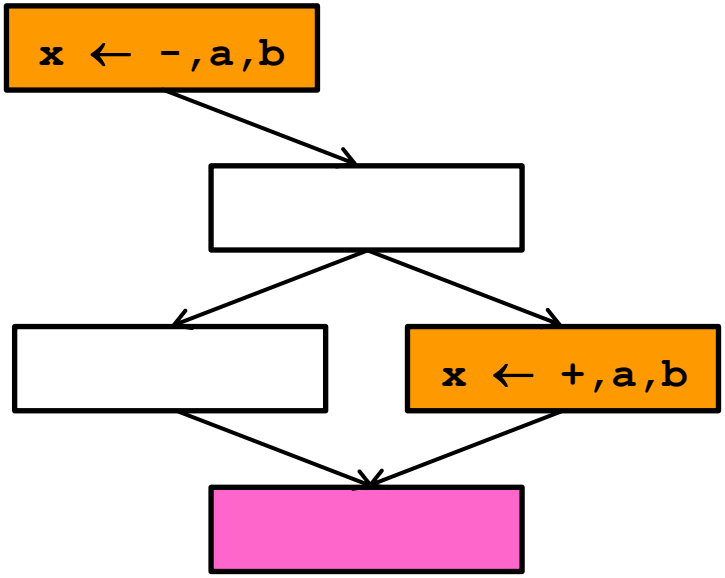
В розовом блоке ϕ-функция для **x** не нужна, так как и слева, и справа приходит одно и то же значение **x**

4.4 Построение частично усеченной SSA-формы

4.4.1. Постановка задачи



В розовом блоке ϕ-функция для **x не нужна**, так как и слева, и справа приходит одно и то же значение **x**

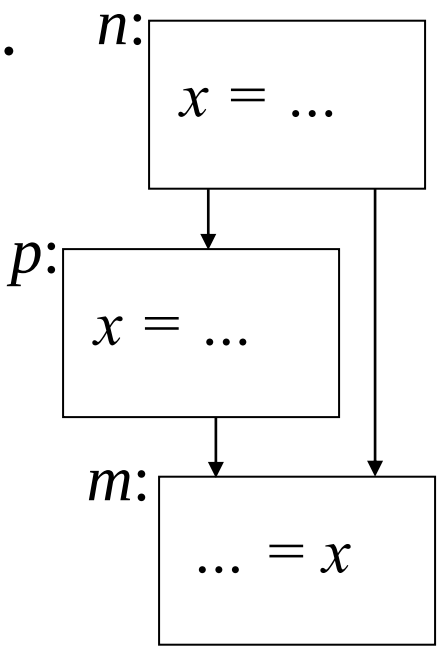


В розовом блоке **нужна** ϕ-функция для **x** так как справа появился блок, в котором вычисляется еще одно значение **x**

4.4 Построение частично усеченной SSA-формы

4.4.1. Постановка задачи

◊ Пусть $Dom(m)$ – множество доминаторов узла m.



- Если $n \in Dom(m)$, определение x_0 в вершине n не требует ϕ-функции в узле m, так как каждый путь, который достигает m проходит через n.
- Если $n \in Dom(m)$, единственным вариантом, при котором определение x_0 не достигнет узла m, является вклинивание определения x в некотором узле $p \notin Dom(m)$, расположенном между n и m.

При этом ϕ-функцию будет требовать не определение x в узле n, а его переопределение в узле p.

4.4 Построение частично усеченной SSA-формы

4.4.2. Граница доминирования (*Dominance Frontier*)

◊ **Определение**. Множество узлов m, удовлетворяющих условиям:

(1) n является доминатором предшественника m

$$p \in Pred(m) \;\&\; n \in Dom(p)$$

(2) n не является строгим доминатором m

$$n \notin (Dom\,(m) - \{m\}).$$

называется *границей доминирования* n и обозначается $DF(n)$.

◊ **Неформально**: $DF(n)$ содержит все первые узлы, которые достижимы из n, на любом пути графа потока, проходящем через n, но над которыми n не доминирует.

4.4 Построение частично усеченной SSA-формы

4.4.2. Граница доминирования

◊ **Пример.** На рисунке справа

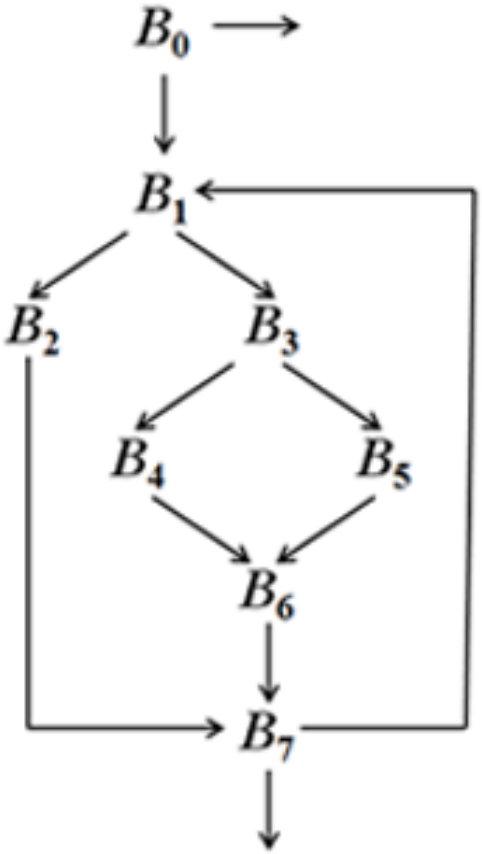
$B_3 \in Dom(B_4)$, $B_3 \in Dom(B_5)$,

$B_3 \in Dom(B_6)$,

$B_3 \notin (Dom(B_7) - \{B_7\})$,

т.е. B_3 является доминатором B_4, B_5 и B_6,

но не является доминатором B_7.



Более того, на любом пути, выходящем из B_3

B_7 – первая вершина, для которой

B_3 не является доминатором

Следовательно, $B_7 \in DF(B_3)$

а так как узел B_3 не является доминатором

узлов B_0, B_1 и B_2, то $DF(B_3) = \{B_7\}$

4.4 Построение частично усеченной SSA-формы

4.4.3. Построение границы доминирования

◊ Свойства узлов, входящих в границу доминирования

(1) узел, принадлежащий границе доминирования должен быть точкой сбора графа потока.

(2) если j – точка сбора: $n \in Pred(j)\ \&\ n \notin Dom(j)$,
то $j \in DF(n)$
т.е. точка сбора j входит в границу доминирования любого своего предшественника n, не являющегося доминатором j.

(3) если j – точка сбора:
$m \in Pred(j)\ \&\ n \in Dom(m)\ \&\ n \notin Dom(j)$, то $j \in DF(n)$
т.е. доминаторы предшественников точки сбора j должны иметь j в своих множествах границ доминирования, если только они не доминируют над j.

4.4 Построение частично усеченной SSA-формы

4.4.3. Построение границы доминирования

◊ Свойства узлов границы доминирования позволяют составить простой алгоритм ее построения

Шаг 1. Найти все точки сбора j графа потока, т.е. все узлы j, у которых $|Pred(j)| > 1$.

Шаг 2. Исследовать каждый узел $p \in Pred(j)$ и продвинуться по дереву доминаторов, начиная с p и вплоть до непосредственного доминатора j: при этом j входит в состав границы доминирования каждого из пройденных узлов, за исключением непосредственного доминатора j.

4.4 Построение частично усеченной SSA-формы

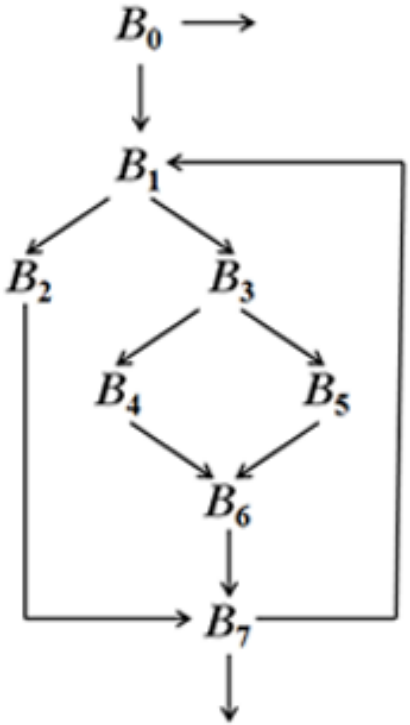
4.4.4. Алгоритм построения границ доминирования

- **Вход**: граф потока
- **Выход**: множество границ доминирования для узлов графа потока
- **Метод**: выполнить следующую программу:

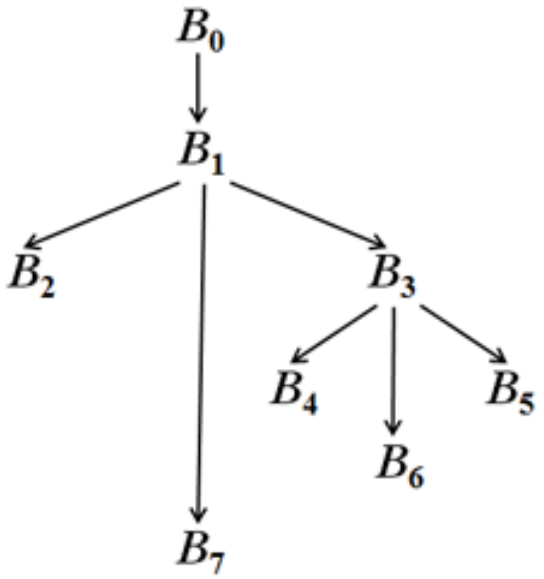
```
for all n ∈ N do DF(n) = ∅;
for all n ∈ N do
    {if |Pred(n)| > 1 then
          for each p ∈ Pred(n) do
                {r = p;
                 while r ≠ IDom(n) do
                   {DF(r) = DF(r) ∪ {n};
                    r = Idom(r);
                   }
                };
    }
```

4.4 Построение частично усеченной SSA-формы

4.4.5. Пример применения алгоритма построения границ доминирования



Граф потока управления



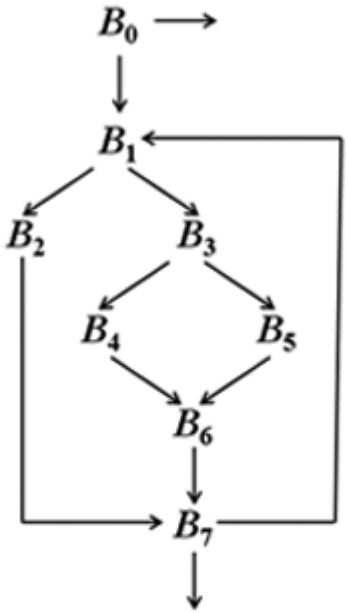
Дерево доминаторов

n	0	1	2	3	4	5	6	7
$Pred(n)$	$\varnothing$	{0,7}	{1}	{1}	{3}	{3}	{4, 5}	{2, 6}
$Dom(n)$	{0}	{0,1}	{0,1,2}	{0,1,3}	{0,1,3,4}	{0,1,3,5}	{0,1,3,6}	{0,1,7}
$Idom(n)$	$\varnothing$	{0}	{1}	{1}	{3}	{3}	{3}	{1}

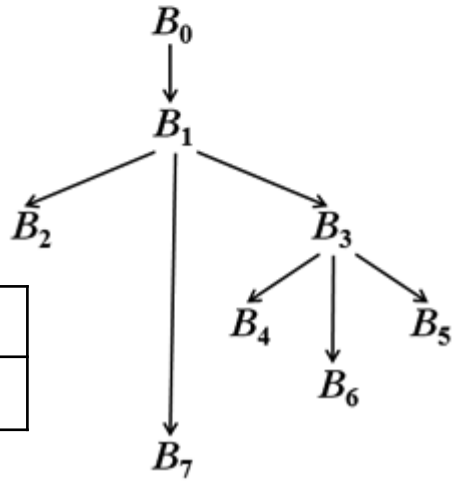
4.4 Построение частично усеченной SSA-формы

4.4.5. Пример применения алгоритма построения границ доминирования

Граф потока управления



Дерево доминаторов



◊ У графа три точки сбора – входы в узлы B_1, B_6 и B_7.

Узел B_6: $Pred(B_6) = \{B_4, B_5\}$, $Idom(B_6) = \{B_3\}$,

проходим от B_5 до B_3, добавляем B_6 к $DF(B_5)$,

проходим от B_4 до B_3, добавляем B_6 к $DF(B_4)$.

Узел B_7: $Pred(B_7) = \{B_2, B_6\}$, $Idom(B_7) = \{B_1\}$,

проходим от B_2 до B_1, добавляем B_7 к $DF(B_2)$,

проходим от B_6 до B_3, добавляем B_7 к $DF(B_6)$

проходим от B_3 до B_1, добавляем B_7 к $DF(B_3)$

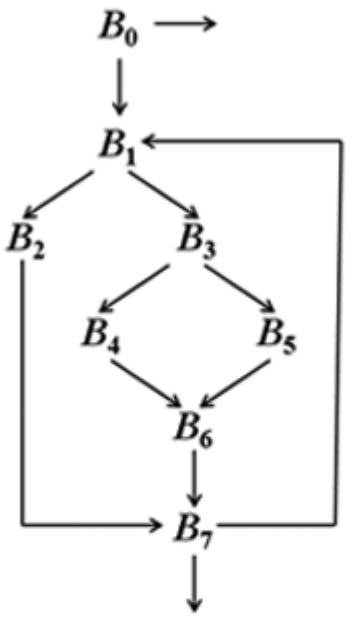
◊ Таблица текущих результатов:

n	0	1	2	3	4	5	6	7
$DF(B_n)$	$\varnothing$	$\varnothing$	$\{B_7\}$	$\{B_7\}$	$\{B_6\}$	$\{B_6\}$	$\{B_7\}$	

4.4 Построение частично усеченной SSA-формы

4.4.5. Пример применения алгоритма построения границ доминирования

Граф потока управления



◊ Узел B_1: $Pred(B_1) = \{B_0, B_7\}$, $Idom(B_1) = \{B_0\}$,

У B_0 нет непосредственного доминатора:

$Idom(B_0) = \varnothing$,

значит $B_1 \notin DF(B_0)$.

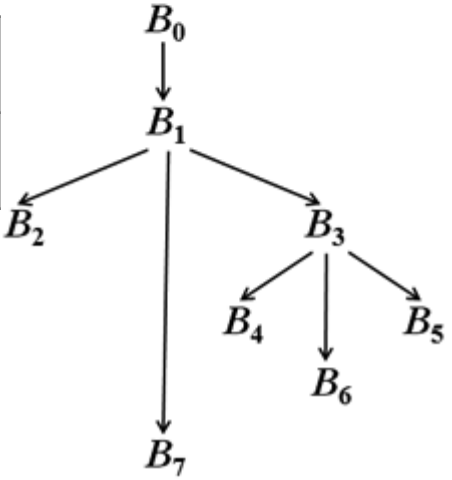
проходим от B_7 до B_1 (по обратному ребру),

добавляем B_1 к $DF(B_7)$.

◊ Окончательная таблица результатов

n	0	1	2	3	4	5	6	7
$DF(B_n)$	$\varnothing$	$\varnothing$	$\{B_7\}$	$\{B_7\}$	$\{B_6\}$	$\{B_6\}$	$\{B_7\}$	$\varnothing$

Дерево доминаторов



4.4 Построение частично усеченной SSA-формы

4.4.6. Размещение ϕ-функций

◊ Базовый алгоритм помещал по ϕ-функции для **каждой** переменной в начало **каждой** вершины сбора.

◊ Границы доминирования позволяют более точно определить, какие именно ϕ -функции необходимы в данной вершине.

Правило: Определение переменной x в базовом блоке B требует соответствующей ϕ-функции в каждой вершине из $DF(B)$.

При этом вставленная ϕ-функция становится новым определением x, так что могут потребоваться новые ϕ-функции.

4.4 Построение частично усеченной SSA-формы

4.4.6. Размещение ϕ-функций

- ◊ Переменная, которая используется только в одном блоке, не может иметь живой ϕ-функции. Поэтому ϕ-функции нужно вставлять только для *глобальных имен*.
- ◊ Для этого вычисляется множество глобальных имен $Globals$. При этом используются результаты анализа живых переменных.
- ◊ Алгоритм вычисления множества $Globals$ попутно вычисляет для каждого базового блока B множество def_B и для каждой $x \in Globals$ – множество $Blocks(x)$ базовых блоков B, в которых $x \in def_B$.

4.4 Построение частично усеченной SSA-формы

4.4.7. Размещение ϕ-функций

Алгоритм построения множеств *Globals* и *Blocks*(x)

◊ **Вход**: граф потока

◊ **Выход**: множество $Globals$,
множества $Blocks(x)$ для каждой переменной x
множества def_B для каждого базового блока B.

◊ **Метод**: Выполнить следующие действия:

4.4 Построение частично усеченной SSA-формы

4.4.7. Размещение ϕ-функций

Алгоритм построения множеств *Globals* и *Blocks*(x)

```
Globals = ∅;
for each variable x do
    Blocks(x) = ∅;
    for each block B do {
        def_B = ∅;
        for each instruction i ∈ B do {
            || пусть команда i имеет вид: x ← op, y, z
            if y ∉ def_B then Globals = Globals ∪ {y};
            if z ∉ def_B then Globals = Globals ∪ {z};
            def_B = def_B ∪ {x};
            Blocks(x) = Blocks(x) ∪ {B}
        }
    }
```

4.4 Построение частично усеченной SSA-формы

4.4.7. Размещение ϕ-функций

Алгоритм размещения ϕ-функций

- **Вход**: исходный граф потока
- **Выход**: преобразованный граф потока
- **Метод**: Выполнить следующие действия

```
for each name x ∈ Globals do {
        WorkList = Blocks(x);
        for each block B ∈ WorkList do {
                for each block D ∈ DF(B) do {
                        вставить φ-функцию для x в D;
                        WorkList = WorkList ∪ {D};
                };
                WorkList = WorkList – {B};
        }
}
```

4.4 Построение частично усеченной SSA-формы

4.4.7. Размещение ϕ-функций

◊ **Замечание 1**. Для каждого блока $B \in WorkList$ алгоритм вставляет ϕ-функцию в начало каждого блока $D \in DF(B)$. Порядок ϕ-функций роли не играет.

После вставления ϕ-функции для x в блок D алгоритм добавляет D в $WorkList$, чтобы отразить новое определение x в блоке D.

4.4 Построение частично усеченной SSA-формы

4.4.7. Размещение ϕ-функций

◊ **Замечание 2**. Для улучшения эффективности алгоритма необходимо избегать следующих двух видов дублирования:

(1) помещение какого-либо блока в $WorkList$ более одного раза для каждого глобального имени; для этого можно вести список уже обработанных блоков (например, не удалять B из $WorkList$, а помечать его как уже обработанный).

(2) рассматриваемый блок может входить в состав границ доминирования нескольких блоков, входящих в $WorkList$; чтобы избежать вставления дублирующих ϕ-функций для переменной x, можно использовать список блоков, которые уже содержат ϕ-функции для x, что быстрее, чем проверять каждый раз какие ϕ-функции включены.

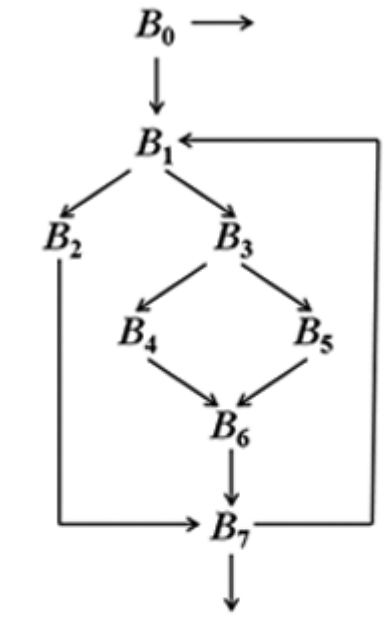
4.4 Построение частично усеченной SSA-формы

4.4.7. Размещение ϕ-функций

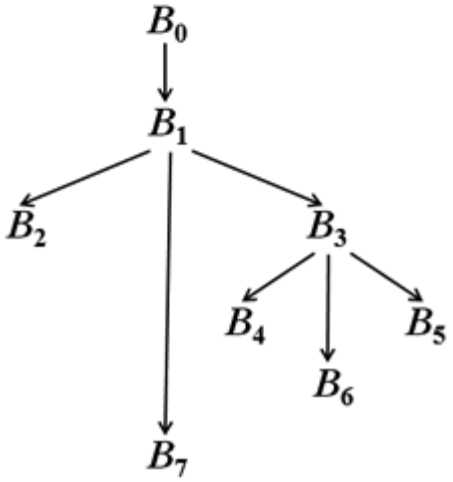
Пример.

B_0	i ← 1	B_5	c ←
B_1	a ← b ←	B_3	a ← d ←
B_2	b ← c ← d ←	B_7	y ← +, a, b z ← +, c, d i ← +, i, 1
B_6	b ←	B_4	d ←

Граф потока управления



Дерево доминаторов



4.4 Построение частично усеченной SSA-формы

4.4.7. Размещение ϕ-функций

Пример.

◊ Сначала вычисляются множества *Globals* и *Blocks(x)*:

$Globals = \{a, b, c, d, i\}$.

Множества $Blocks(x)$ представлены в таблице:

a	b	c	d	i
$\{B_1, B_3\}$	$\{B_3, B_6\}$	$\{B_1, B_2, B_5\}$	$\{B_2, B_3, B_4\}$	$\{B_0, B_7\}$

◊ Алгоритму размещения ϕ -функций потребуются также уже вычисленные границы доминирования

n	0	1	2	3	4	5	6	7
$DF(B_n)$	$\varnothing$	$\varnothing$	$\{B_7\}$	$\{B_7\}$	$\{B_6\}$	$\{B_6\}$	$\{B_7\}$	$\{B_1\}$

4.4 Построение частично усеченной SSA-формы

4.4.7. Размещение ϕ-функций

Пример.

◊ Применяем алгоритм размещения ϕ-функций.

Берем первую переменную из $Globals = \{a, b, c, d, i\}$, т.е . a

$Blocks(a) = \{B_1, B_3\}$, так что алгоритм должен вставить ϕ-функцию для a в каждый блок из границ доминирования $DF(B_1) = \varnothing$ и $DF(B_3) = \{B_7\}$.

Вставив ϕ-функцию для a в B_7, получим новое определение a – ϕ-функцию, вставленную в B_7. Следовательно, необходимо включить B_7 в $WorkList$ и вставить ϕ-функцию для a в каждый блок из $DF(B_7) = \{B_1\}$. Но B_1 уже имеется в $WorkList$, так что, $DF(B_7)$ не добавляется к $WorkList$.

x	a	b	c	d	i
$Blocks(x)$	$\{B_7, B_1\}$	$\{B_7, B_1\}$	$\{B_7, B_1, B_6\}$	$\{B_7, B_1, B_6\}$	$\{B_1\}$

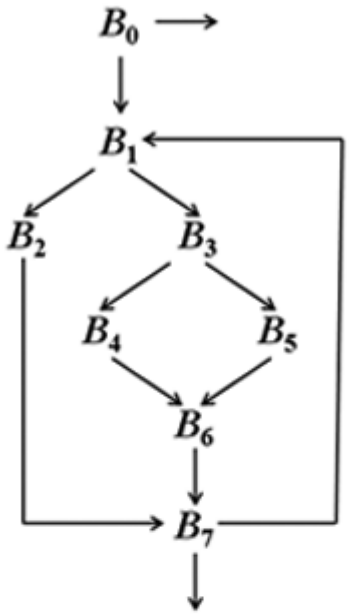
4.4 Построение частично усеченной SSA-формы

4.4.7. Размещение ϕ-функций

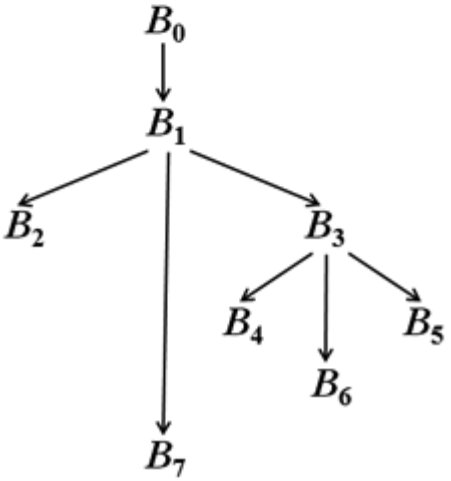
Пример.

B_0 i ← 1	B_5 c ←
B_1 a ← φ(a,a) b ← φ(b,b) c ← φ(c,c) d ← φ(d,d) i ← φ(i,i) a ← b ←	B_7 a ← φ(a,a) b ← φ(b,b) c ← φ(c,c) d ← φ(d,d) y ← +, a, b z ← +, c, d i ← +, i, 1
B_2 b ← c ← d ←	B_3 a ← d ←
B_4 d ←	B_6 c ← φ(c,c) d ← φ(d,d) b ←

Граф потока управления



Дерево доминаторов



4.4 Построение частично усеченной SSA-формы
4.4.8. Переименование переменных

Алгоритм

◊ **Вход**: программа с размещенными ϕ-функциями

◊ **Выход**: программа, в которой переменным сопоставлены их *SSA-имена*

◊ **Метод**: Сначала (в основном алгоритме) инициализируются стеки и счетчики, после чего из корня дерева доминаторов n_0 вызывается рекурсивная функция *Rename*.

Rename обрабатывает блок, рекурсивно вызывая его последователей по дереву доминаторов.

Закончив обрабатывать очередной блок, *Rename* выталкивает из стеков все имена, помещенные в них во время обработки блока.

Функция *NewName*, манипулируя со счетчиками и стеками, в случае необходимости создает новые имена.

4.4 Построение частично усеченной SSA-формы
4.4.7. Переименование переменных

Алгоритм

- Основной алгоритм:

```
for each i ∈ Globals do{
        counter[i] = 0;
        stack[i] = ∅;
};
Rename(n0);
```

- Функция **NewName**:

```
NewName(n) {
        i = counter[n];
        counter[n] += 1;
        Push ni onto stack[n];
              return ni
}
```

4.4 Построение частично усеченной SSA-формы
4.4.7. Переименование переменных.

Функция *Rename*(*B*) :

```
for each ϕ-function ∈ B: x = ϕ(…) do
     rename x as NewName(x);
for each instruction ∈ B: x ← op, y, z do{
     rewrite y as top(stack[y]);
     rewrite z as top(stack[z]);
     rewrite x as NewName(x);
for each successor of B in the flowgraph do
     fill in ϕ-function parameters;
for each successor S of B in the
     dominator tree do Rename(S)
for each ϕ-function ∈ B: x = ϕ(…)do
          Pop(stack[x]);
for each instruction ∈ B: x ← op, y, z do
          Pop(stack[x]);
```

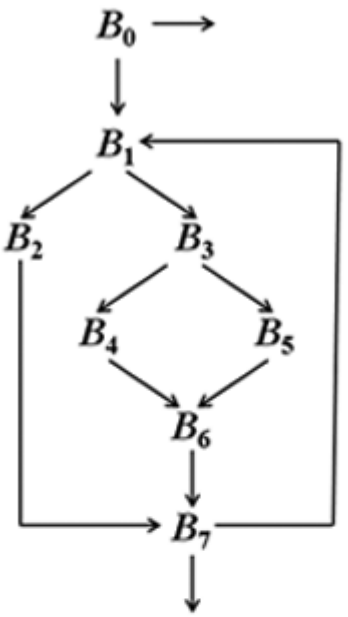
4.4 Построение частично усеченной SSA-формы

4.4.8. Переименование переменных

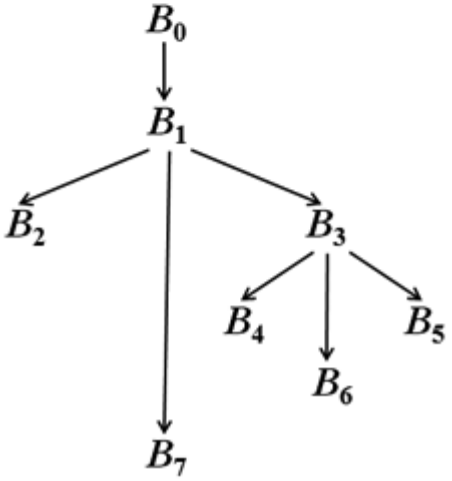
Применим алгоритм переименования к рассматриваемой программе в предположении, что на входе в блок B_0 определены имена a_0, b_0, c_0, d_0.

B_0	i ← 1	B_5	c ←
B_1	a ← φ(a,a) b ← φ(b,b) c ← φ(c,c) d ← φ(d,d) i ← φ(i,i) a ← b ←	B_7	a ← φ(a,a) b ← φ(b,b) c ← φ(c,c) d ← φ(d,d) y ← +, a, b z ← +, c, d i ← +, i, 1
B_2	b ← c ← d ←	B_3	a ← d ←
B_4	d ←	B_6	c ← φ(c,c) d ← φ(d,d) b ←

Граф потока управления



Дерево доминаторов



4.4 Построение частично усеченной SSA-формы

4.4.8. Переименование переменных

Блок B_0

Состояние счетчиков и стеков
в момент вызова $Rename(B_0)$

Глобальные переменные	a	b	c	d	i
Счетчики	1	1	1	1	0
Стеки	a_0	b_0	c_0	d_0	

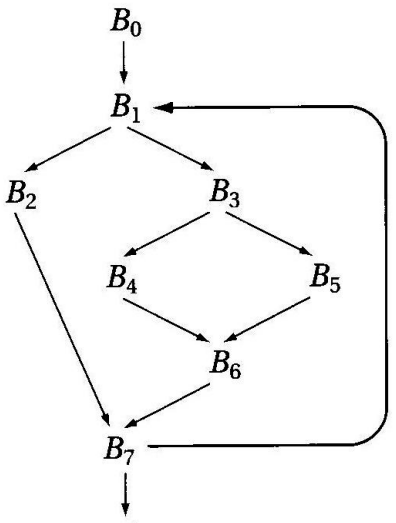
$Rename(B_0)$

1) заменяет i на i_0, увеличивает значение счетчика i и заталкивает i_0 в соответствующий стек.

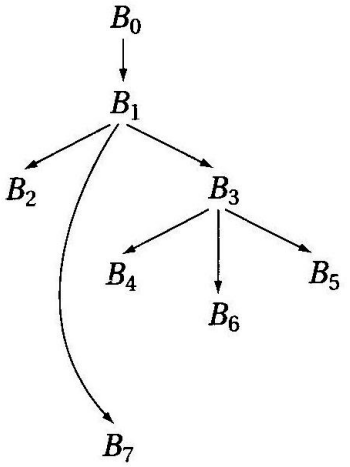
2) заменяет первый параметр каждой φ-функции на соответствующие имена из текущего состояния a_0, b_0, c_0, d_0, i_0.

3) вызывает по рекурсии сына B_0 по дереву доминаторов – узел B_1

Граф потока управления



Дерево доминаторов



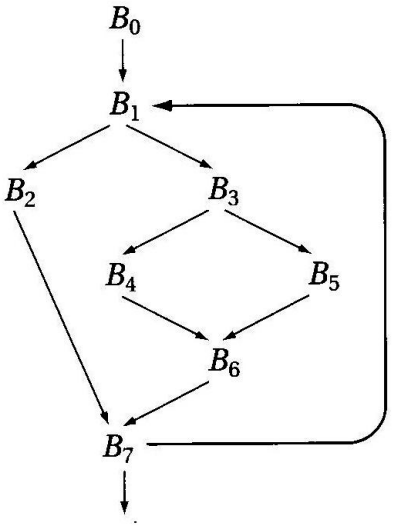
4.4 Построение частично усеченной SSA-формы

4.4.8. Переименование переменных

Блок B_0

Когда в конце обхода дерева доминаторов управление снова попадает в B_0 $Rename$ выталкивает значение из стека для i и происходит выход их нее.

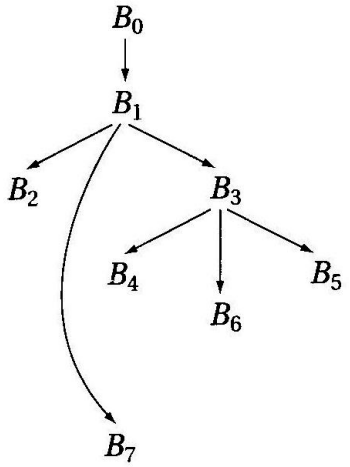
Граф потока управления



Состояние счетчиков и стеков в момент, когда $Rename(B_0)$ уже кончила обрабатывать блок B_0, но еще не удалила имена, связанные с B_0 из соответствующих стеков

Глобальные переменные	a	B	c	d	i
Счетчики	1	1	1	1	1
Стеки	a_0	b_0	c_0	d_0	i_0

Дерево доминаторов



4.4 Построение частично усеченной SSA-формы

4.4.8. Переименование переменных

Блок B_1

$Rename(B_1)$ по рекурсии во время обработки блока B_0.

На входе в блок B_1 $Rename$

1) заменяет имена, определяемые φ-функциями на новые имена a_1, b_1, c_1, d_1, i_1.

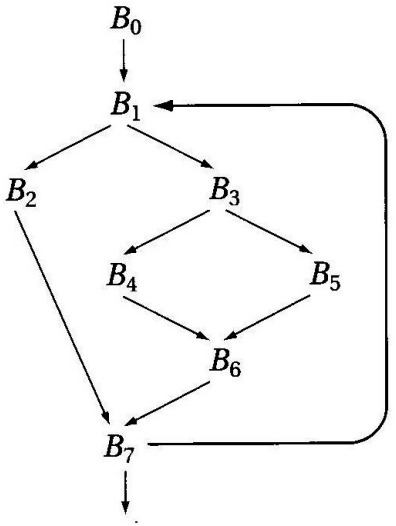
2) заменяет имена в определениях переменных a и c на a_2 и c_2.

3) вызывает по рекурсии сыновей B_1 по дереву доминаторов – узлы B_2, B_3 и B_7.

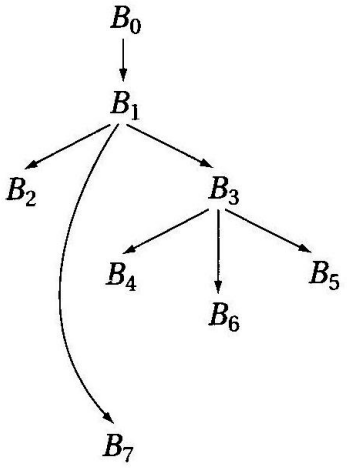
Когда управление снова попадает в B_1

$Rename$ освобождает стеки и идет на выход.

Граф потока управления



Дерево доминаторов



На выходе из B_1

Глобальные переменные	a	b	c	d	i
Счетчики	3	2	3	2	2
Стеки	a_0	b_0	c_0	d_0	i_0
	a_1	b_1	c_1	d_1	i_1
	a_2		c_2		

4.4 Построение частично усеченной SSA-формы

4.4.8. Переименование переменных

Блок B_7 (последний)

$Rename(B_7)$ по рекурсии из блока B_1.

1) заменяет имена, определяемые φ-функциями, создавая новые имена a_4, b_4, c_6 и d_6.
2) заменяет использования глобальных имен в двух присваиваниях, не меняя определяемых ими имен, так как ни y, ни z не являются глобальными.
3) последнем присваивании заменяет используемое имя на i_1, а затем создает новое имя i_2 для определения переменной i.
4) заменяет второй параметр каждой φ-функции в B_1 (единственный последователь по ГПУ) на соответствующее текущее имя (a_4, b_4, c_6, d_6 и i_2).

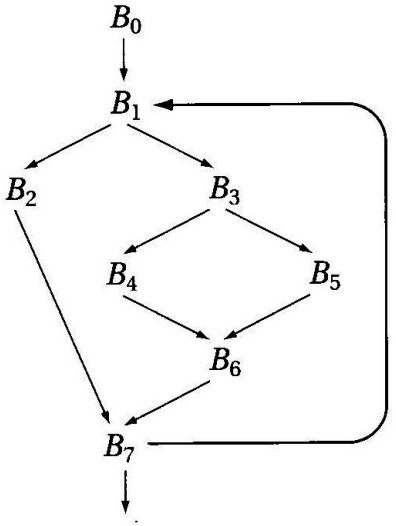
Потом $Rename$ освобождает стеки, возвращается в B_1, в B_0 и прекращает работу.

4.4 Построение частично усеченной SSA-формы

4.4.8. Переименование переменных

Блок B_7 (последний)

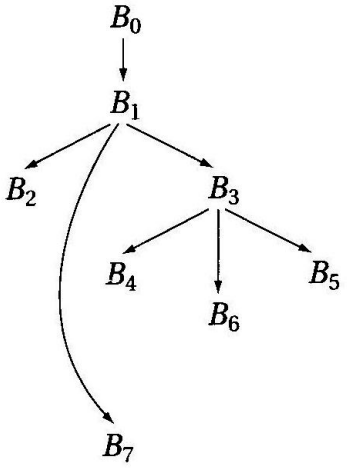
Граф потока управления



Вход в B_7

Глобальные переменные	a	b	c	d	i
Счетчики	4	4	6	6	2
Стеки	a_0	b_0	c_0	d_0	i_0
	a_1	b_1	c_1	d_1	i_1
	a_2		c_2		

Дерево доминаторов



Выход из B_7

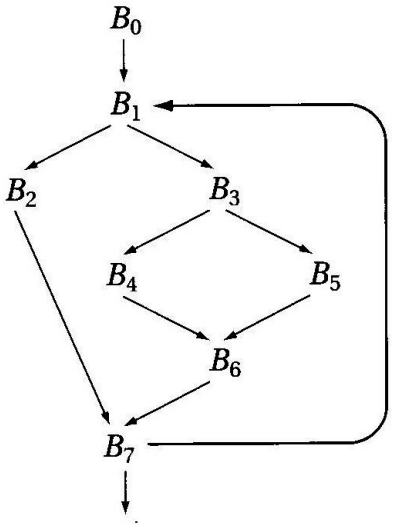
Глобальные переменные	a	b	c	d	I
Счетчики	5	5	7	7	3
Стеки	a_0	b_0	c_0	d_0	I_0
	a_1	b_1	c_1	d_1	I_1
	a_2	b_4	c_2	D_6	I_2
	a_4		c_6		

4.4 Построение частично усеченной SSA-формы
4.4.8. Переименование переменных

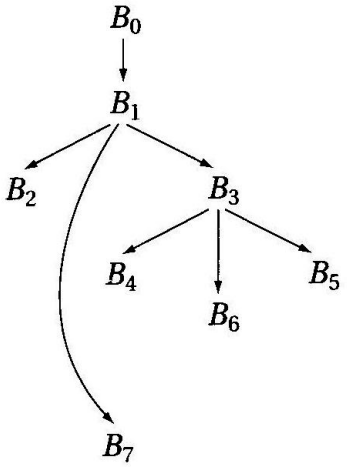
После окончания работы алгоритма получим

B_0	$i_0 \leftarrow 1$	B_5	$c_4 \leftarrow$
B_1	$a_1 \leftarrow \varphi(a_0, a_4)$ $b_1 \leftarrow \varphi(b_0, b_4)$ $c_1 \leftarrow \varphi(c_0, c_6)$ $d_1 \leftarrow \varphi(d_0, d_6)$ $i_1 \leftarrow \varphi(i_0, i_2)$ $a_2 \leftarrow$ $b_2 \leftarrow$	B_7	$a_4 \leftarrow \varphi(a_2, a_3)$ $b_4 \leftarrow \varphi(b_2, b_3)$ $c_6 \leftarrow \varphi(c_3, c_5)$ $d_6 \leftarrow \varphi(d_2, d_5)$ $y \leftarrow +, a_4, b_4$ $z \leftarrow +, c_6, d_6$ $i_2 \leftarrow +, i_1, 1$
B_2	$b_2 \leftarrow$ $c_3 \leftarrow$ $d_2 \leftarrow$	B_3	$a_3 \leftarrow$ $d_3 \leftarrow$
B_4	$d_4 \leftarrow$	B_6	$c_5 \leftarrow \varphi(c_2, c_4)$ $d_5 \leftarrow \varphi(d_4, d_3)$ $b_3 \leftarrow$

Граф потока управления



Дерево доминаторов



4.5 Восстановление кода из SSA-формы

4.5.1. Постановка задачи

- ◊ Поскольку современные процессоры не выполняют ϕ-функций, компилятор должен перевести программу в SSA-форме в выполняемый код.
- ◊ Рассмотрение примеров наводит на мысль, что **компилятору достаточно попросту опустить индексы имен и удалить ϕ-функции**.

  Такой подход был бы правильным, если бы при построении SSA-формы использовался простейший алгоритм.
- ◊ Однако, если переименования выполняются рассмотренным алгоритмом, простой процесс переименования **может привести к некорректному коду**.

4.5 Восстановление кода из SSA-формы

4.5.1. Постановка задачи

◊ **Пример**. Внизу слева показан базовый блок из четырех команд и его оптимизация с помощью ЛНЗ над исходными именами.
Справа показан тот же самый пример с использованием SSA-имен.

Вследствие того, что исходное имя a имеет разные SSA-имена a_0 и a_1, удалось оптимизировать последнее присваивание.

Заметим однако, что если у имен опустить индексы, получится некорректный код: c будет присвоено значение 17.

Исходные имена		***SSA*-имена**	
a ← +, x, y	a ← +, x, y	a_0 ← +, x_0, y_0	a_0 ← +, x_0, y_0
b ← +, x, y	b ← a	b_0 ← +, x_0, y_0	b_0 ← a_0
a ← 17	a ← 17	a_0 ← 17	a_1 ← 17
c ← +, x, y	c ← +, x, y	c_0 ← +, x_0, y_0	c_0 ← a_0

4.5 Восстановление кода из SSA-формы

4.5.2. Замена ϕ-функций группами команд копирования

◊ Можно оставить SSA-имена неизменными, заменив каждую ϕ-функцию группой команд копирования (по одной для каждого входного ребра), предоставив разбираться с именами оптимизирующему преобразованию «Распространение копий».

В рассматриваемом примере при удалении ϕ-функций будет:

```
B7   a4 ← φ(a2, a3)
     b4 ← φ(b2, b3)
     c6 ← φ(c3, c5)
     d6 ← φ(d2, d5)
     y  ← +, a4, b4
     z  ← +, c6, d6
     i2 ← +, i1, 1
```

```
B6   c5 ← φ(c2, c4)
     d5 ← φ(d4, d3)
     b3 ←
```

```
B2   b2 ←
     c3 ←
     d2 ←
```

```
B7   y  ← +, a4, b4
     z  ← +, c6, d6
     i2 ← +, i1, 1
```

```
B6   b3 ←
     a4 ← a2
     b4 ← b2
     c6 ← c3
     d6 ← d2
```

```
B2   b2 ←
     c3 ←
     d2 ←
     a4 ← a3
     b4 ← b3
     c6 ← c5
     d6 ← d5
```

4.5 Восстановление кода из SSA-формы

4.5.2. Замена ϕ-функций группами инструкций копирования

```
B7   a4 ← φ(a2, a3)
     b4 ← φ(b2, b3)
     c6 ← φ(c3, c5)
     d6 ← φ(d2, d5)
     y  ← +, a4, b4
     z  ← +, c6, d6
     i2 ← +, i1, 1
```

```
B7   y  ← +, a4, b4
     z  ← +, c6, d6
     i2 ← +, i1, 1
```

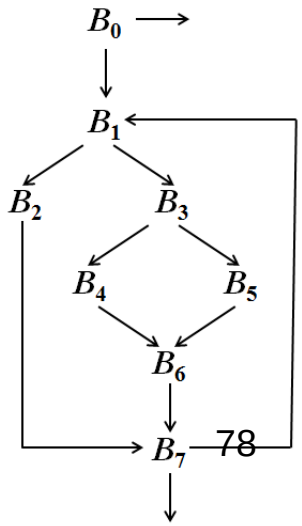
```
B6   c5 ← φ(c2, c4)
     d5 ← φ(d4, d3)
     b3 ←
```

```
B6   b3 ←
     a4 ← a2
     b4 ← b2
     c6 ← c3
     d6 ← d2
```

```
B2   b2 ←
     c3 ←
     d2 ←
```

```
B2   b2 ←
     c3 ←
     d2 ←
     a4 ← a3
     b4 ← b3
     c6 ← c5
     d6 ← d5
```

- ◊ Удаление ϕ-функций из блока B_6 породит инструкции копирования в блоках B_4 и B_5.
- ◊ Удаление ϕ-функций из блока B_1 должно породить инструкции копирования в блоках B_0 и B_7. Но этого делать нельзя!



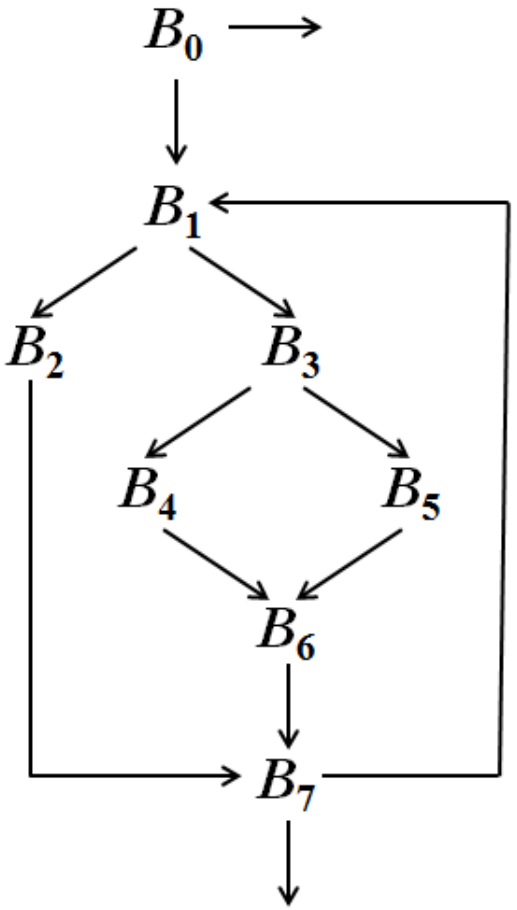
4.5 Восстановление кода из SSA-формы

4.5.2. Замена ϕ-функций группами инструкций копирования

◊ Удаление ϕ-функций из блока B_1 должно породить инструкции копирования в блоках B_0 и B_7. Но этого делать нельзя! Почему?

◊ У блоков B_0 и B_7 по два последующих блока (вторые блоки не попали на схему, так как они находятся вне анализируемого цикла).

Если вставить копирования в конце B_0 и B_7, они будут выполняться не только на ребрах, внутри рассматриваемого цикла, но и на ребрах выводящих из цикла (помечены условием $i > 100$).



4.5 Восстановление кода из SSA-формы

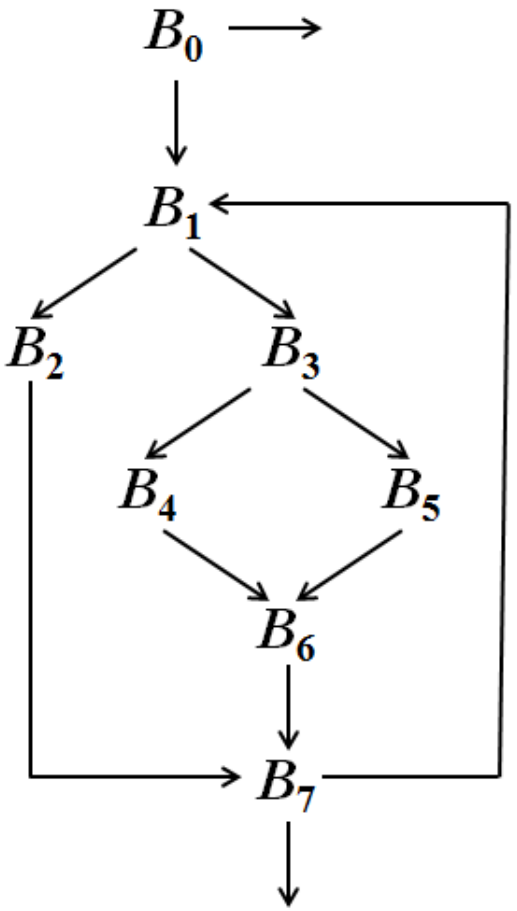
4.5.2. Замена ϕ-функций группами инструкций копирования

- По определению ребро, выходящее из вершины с несколькими потомками, и входящее в вершину с несколькими предками, называется *критическим*.

  Ребра (B_0 B_1) и (B_7 B_1) – критические.

- Критические ребра обычно исключают, разрывая их а вставляя в разрыв дополнительные вершины.

- Разорвем критические ребра и добавим два новых блока B_8 и B_9, поместив в них требуемые инструкции копирования.



4.5 Восстановление кода из SSA-формы

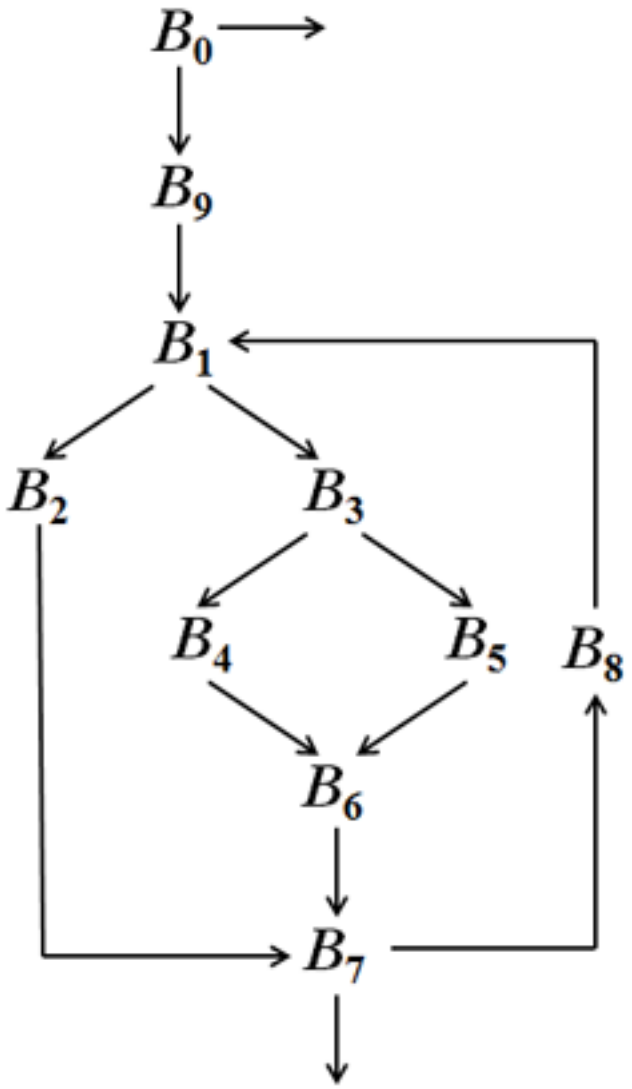
4.5.2. Замена ϕ-функций группами инструкций копирования

- ◊ По определению ребро, выходящее из вершины с несколькими потомками, и входящее в вершину с несколькими предками, называется *критическим*.

  Ребра (B_0 B_1) и (B_7 B_1) – критические.

- ◊ Критические ребра обычно исключают, разрывая их а вставляя в разрыв дополнительные вершины.

- ◊ Разорвем критические ребра и добавим два новых блока B_8 и B_9, поместив в них требуемые инструкции копирования.



4.5 Восстановление кода из SSA-формы

4.5.2. Замена ϕ-функций группами инструкций копирования