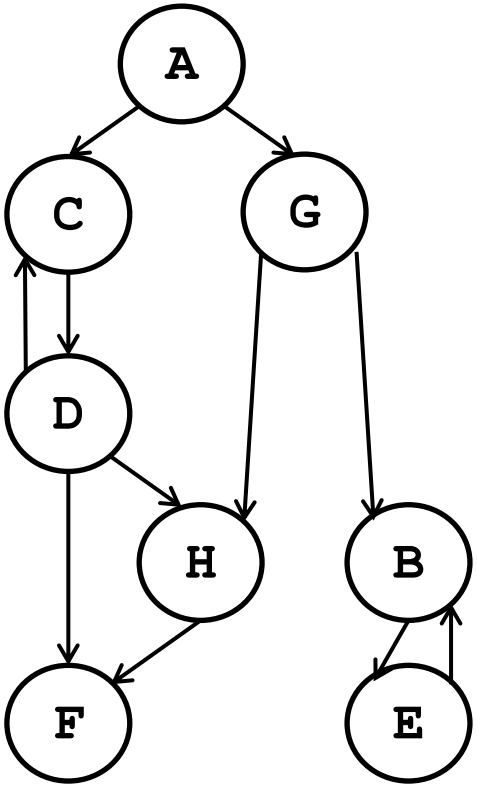
2. Построение множеств Input и Output

2.1 Нумерация вершин ГПУ

2.1.1 Остовное дерево

◊ Необходимо перенумеровать вершины ГПУ. Для этого построим остовное дерево ГПУ с корнем в вершине **`Entry`** и обойдем его «сначала в глубину», используя «обратную нумерацию» (остовное дерево с так пронумерованными вершинами называется «глубинным остовным деревом» – $DFST$).

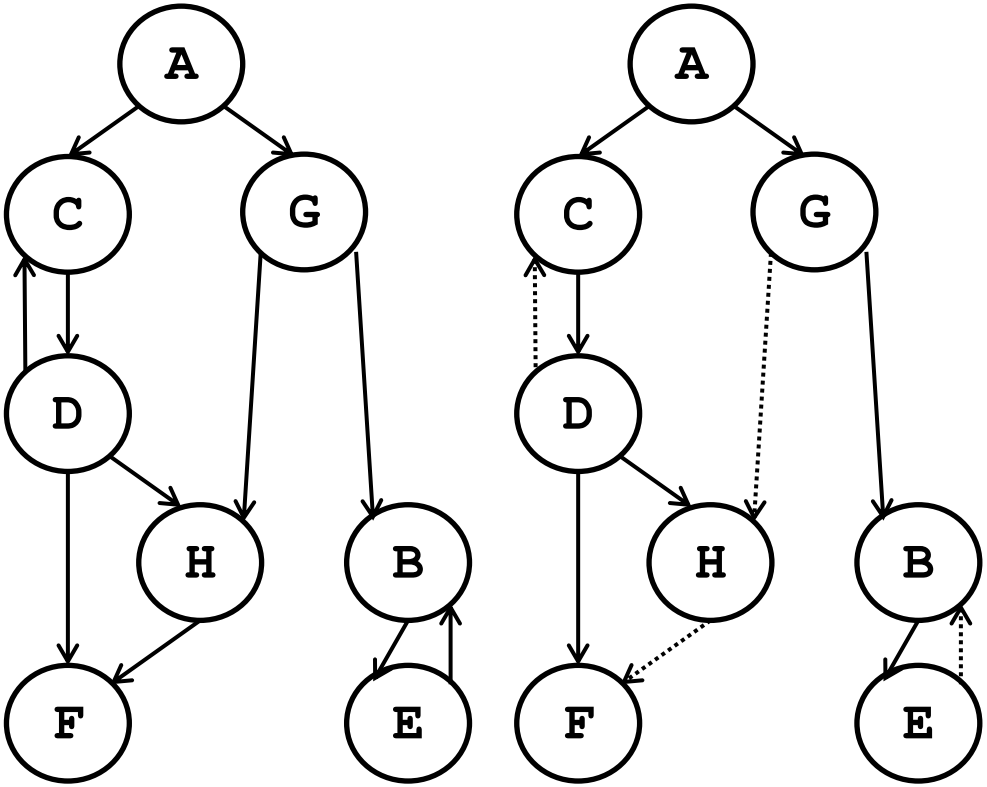
◊ **Пример 1**.



2.1 Нумерация вершин ГПУ

2.1.1 Остовное дерево

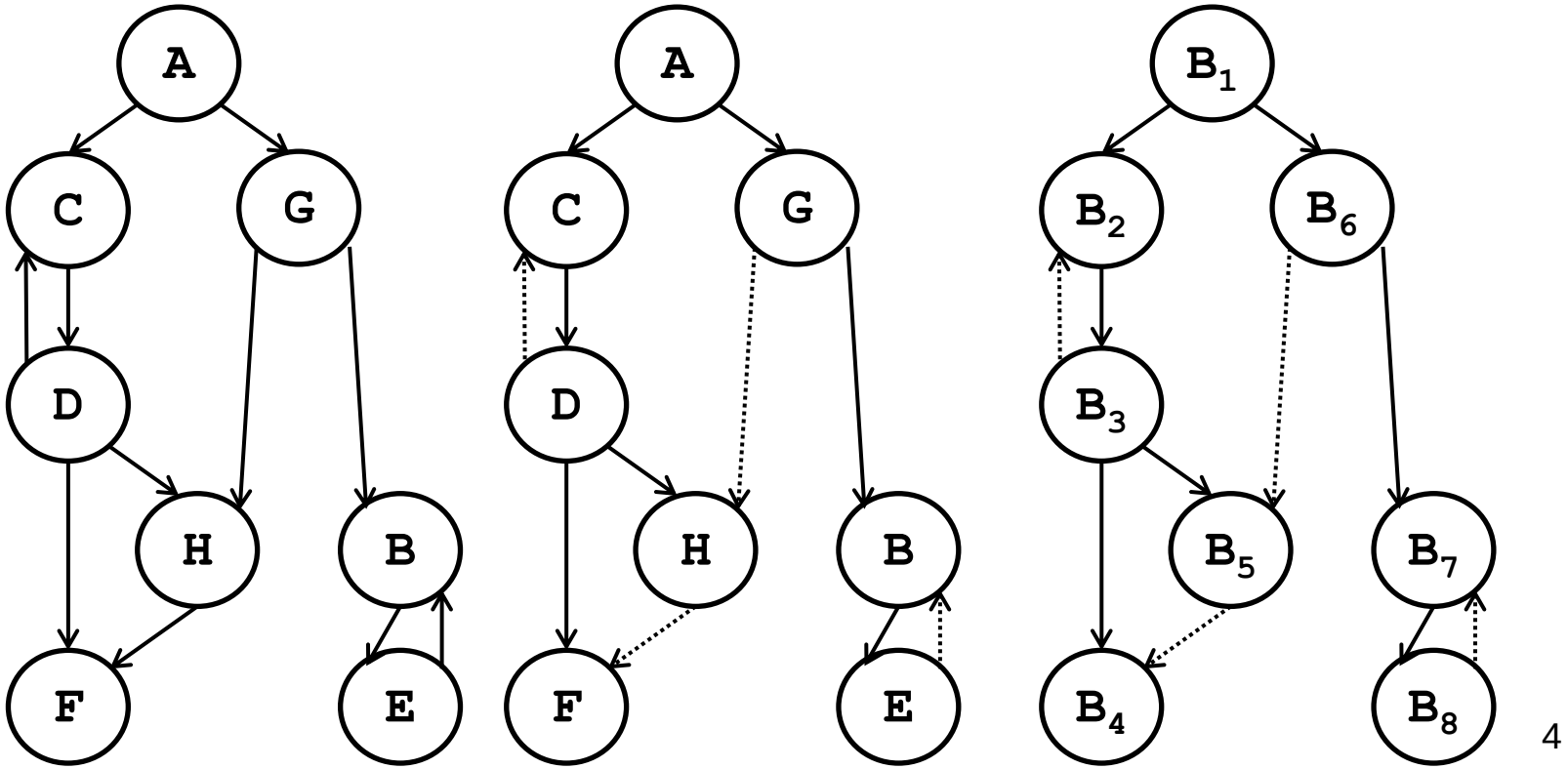
◊ **Пример 1**.



2.1 Нумерация вершин ГПУ

2.1.1 Остовное дерево

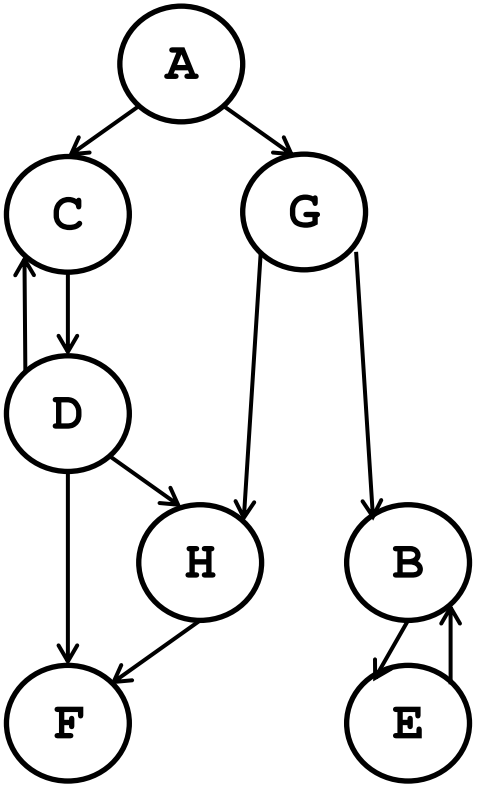
◊ **Пример 1**.

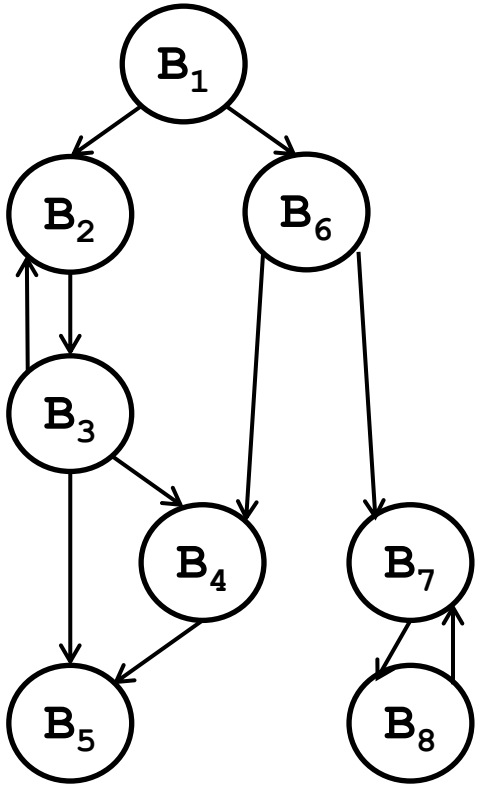


2.1 Нумерация вершин ГПУ

2.1.1 Остовное дерево

◊ **Пример 1**.

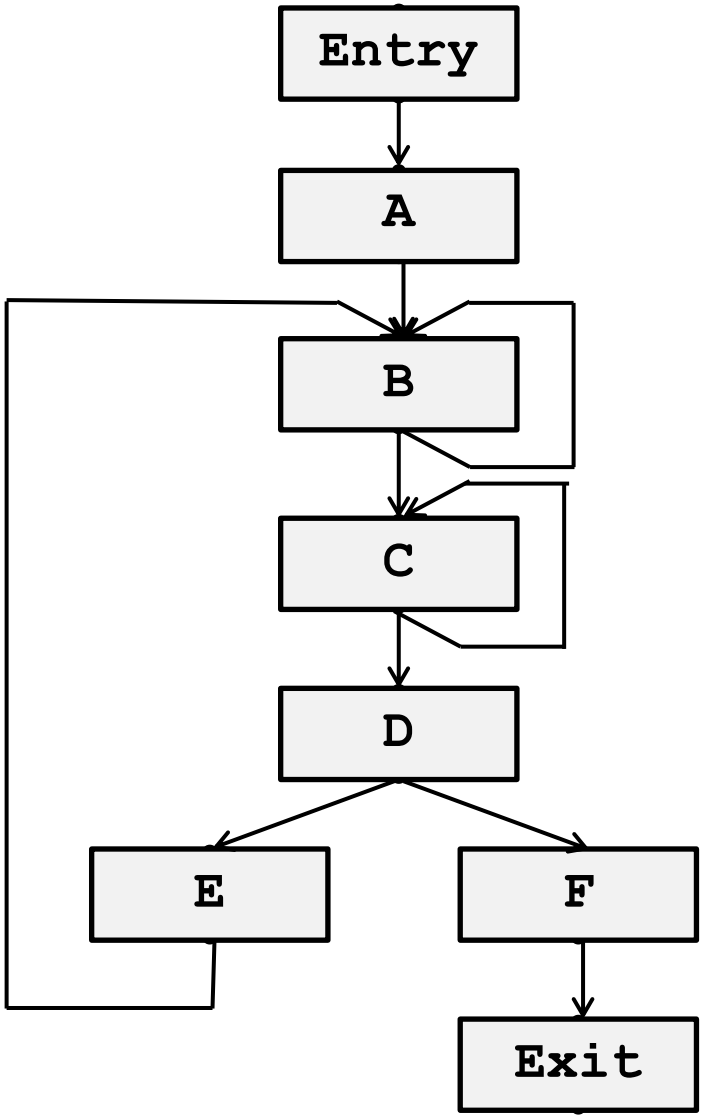


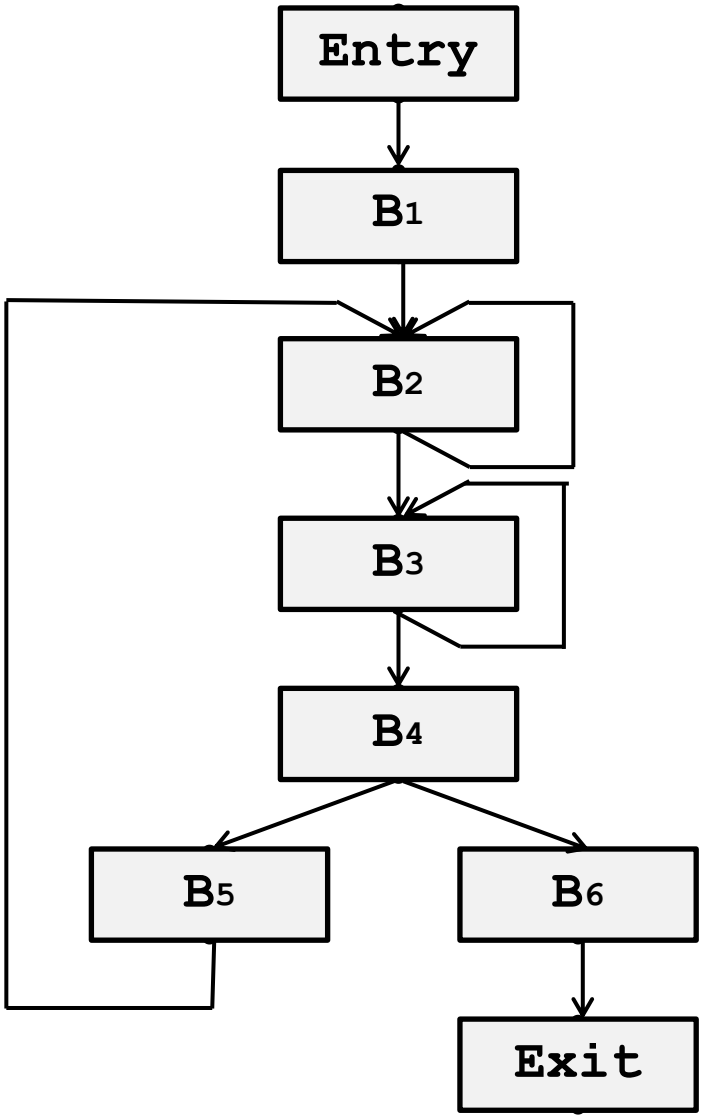


2.1 Нумерация вершин ГПУ

2.1.1 Остовное дерево

◊ **Пример 2**





2.1 Нумерация вершин ГПУ

2.1.2 Алгоритм построения глубинного остовного дерева и нумерации вершин ГПУ

Алгоритм

◊ **Вход**: ГПУ $G = \langle N, E \rangle$ с корнем $Entry \in N$

◊ **Выход**: глубинное остовное дерево графа G ($T_{DFS}(G)$) и нумерация узлов графа G, соответствующая упорядочению в глубину.

◊ **Метод**: все узлы $n \in N$ помечаются как nv (*not visited*)

вызывается рекурсивная процедура **`search(n0)`**

когда процедура **`search`** завершится, будут построены:

- массив номеров узлов dfn
- множество T ребер глубинного остовного дерева $T_{DFS}(G)$

2.1 Нумерация вершин ГПУ

2.1.2 Алгоритм построения глубинного остовного дерева и нумерации вершин ГПУ

Рекурсивный алгоритм построения *DFST*

◊ Функция **main()** :

```
main() {
   T = ∅;
   for all n ∈ N n.vst = nv;
   c = |N|;
   DFST(n0);
}
```

2.1 Нумерация вершин ГПУ

2.1.2 Алгоритм построения глубинного остовного дерева и нумерации вершин ГПУ

◊ Функция **DFST()**:

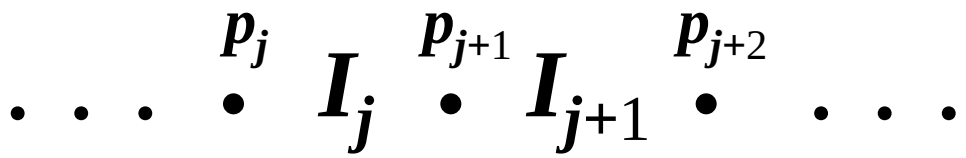
```
void DFST(n) {
   Отмечаем n как v;
   for all s ∈ Succ(n)
         if (s.vst == nv){
               T ∪= {n→s};
               DFST(s);
         }
   dfn[c] = n;
   c--;
}
```

◊ **Замечание.** Для каждой вершины $n \in N$ определено множество $Succ(n)$, содержащее все вершины $s \in N$, в которые входят дуги, выходящие из n.

2.2 Анализ потока данных

2.2.1 Поток данных

- *Состояние программы* – множество значений всех переменных программы, включая переменные в кадрах стека времени выполнения, находящихся ниже текущей вершины стека
- *Точки программы* (…, p_j, p_{j+1}, p_{j+2}, …) расположены между ее *инструкциями* (…, I_j, I_{j+1}, …)



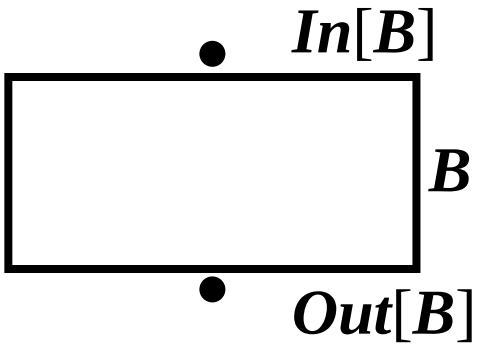
- Инструкция программы I_j описывается парой состояний:
  - состоянием в *точке программы* p_j *перед* инструкцией I_j
  - состоянием в *точке программы* p_{j+1} *после* инструкции I_j.

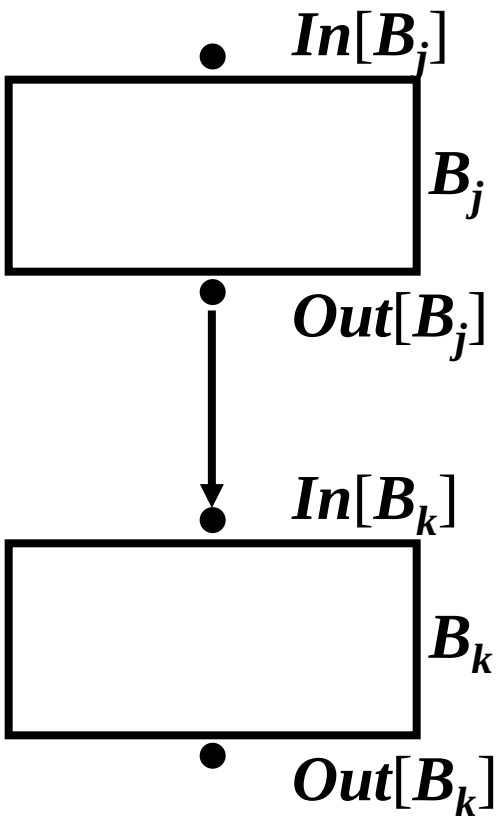
2.2 Анализ потока данных

2.2.1 Поток данных

Базовый блок B описывается парой состояний:

- состоянием $\boldsymbol{In}[B]$ в точке входа в B (перед первой инструкцией),
- состоянием $\boldsymbol{Out}[B]$ в точке выхода из B (после последней инструкции)





С дугой от блока B_j к блоку B_k связаны две точки программы :

- точка выхода из блока B_j (ей соответствует состояние $\boldsymbol{Out}[B_j]$)
- точка входа в блок B_k (ей соответствует состояние $\boldsymbol{In}[B_k]$)

2.2 Анализ потока данных

2.2.3 Передаточные функции инструкций

- Соотношение f_{I_J} между значениями данных до и после инструкции I_j называется *передаточной функцией* инструкции I_j.
- Передаточные функции работают **в прямом** и **обратном** направлениях:
  - **В задаче прямого обхода**: $Out[I_j] = f_{I_j}(In[I_j])$
  - **В задаче обратного обхода**: $In[I_j] = f^b_{I_j}(Out[I_j])$

- Если I_j и I_{j+1} – *последовательные* инструкции блока B, то
  - **В задаче прямого обхода** $In[I_{j+1}] = Out[I_j]$
  - **В задаче обратного обхода** $Out[I_{j-1}] = In[I_j]$

2.2 Анализ потока данных

2.2.3 Передаточные функции базовых блоков

◊ Рассмотрим базовый блок

$$B = \langle P, Input, Output \rangle,$$

где $P = I_1, \ldots, I_n$ (в указанном порядке)

◊ По определению

$$In[B] = In[I_1],\ Out[B] = Out[I_n].$$

◊ Передаточная функция f_B блока B по определению равна композиции передаточных функций его инструкций $I_1, \ldots, I_n$

$$f_B(x) = f_{I_n}(f_{I_{n-1}}(\ldots f_{I_1}(x)\ldots)) = (f_{I_1} \circ f_{I_2} \circ \ldots \circ f_{I_n})(x)$$

или

$$f_B = f_{I_1} \circ f_{I_2} \circ \ldots \circ f_{I_n}$$

2.2.4 Передаточные функции базовых блоков

◊ **При прямом обходе**:

Соотношение между потоком данных при выходе из блока B и потоком данных при входе в него имеет вид

$$Out[B] = f_B(In[B])$$

◊ **При обратном обходе**:

Соотношение между потоком данных при входе в блок B и потоком данных при выходе из него имеет вид

$$In[B] = f_B^b(Out[B])$$

2.3 Достигающие определения

2.3.1 Терминология

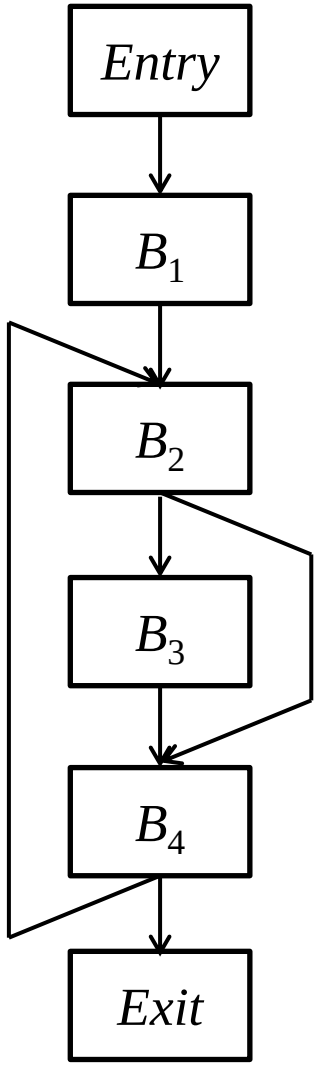
- ◊ *Определением переменной* x называется инструкция, которая присваивает значение переменной x.
- ◊ *Использованием переменной* x является инструкция, одним из операндов которой является переменная x.
- ◊ Каждое новое определение переменной x *убивает* ее предыдущее определение.
- ◊ Определение d *достигает* точки p, если существует путь от точки, непосредственно следующей за d, к точке p, такой, что вдоль этого пути d остается живым.

2.3 Достигающие определения

2.3.2 Пример

B_1
`i ← -, m, 1`
`j ← n`
`a ← u1`
	B_2
	`i ← +, i, 1`
	`j ← -, j, 1`

B_3 `a ← u2`	B_4 `i ← u3`



- Начало блока B_2 достигается определениями:
  - (i, B_1), (j, B_1), (a, B_1),
  - (j, B_2) (других определений j на пути от (j, B_2) до начала блока B_2 нет)
  - (a, B_3)
  - (i, B_4)
- (i, B_2) **не достигает** начала B_2, так как его убивает (i, B_4)
- (j, B_2) убивает (j, B_1), не позволяя ему достичь блоков B_3 и B_4

2.3 Достигающие определения

2.3.3 Передаточные функции для достигающих определений

◊ Рассмотрим инструкцию I

$$d\text{: } \mathtt{u = v + w}$$

расположенную между точками p_1 и p_2 программы.

- ⧫ Пусть

  $\boldsymbol{x}$ – множество определений, достигающих точки p_1

  gen_I – множество определений, порождаемых инструкцией I

  $kill_I$ – множество определений, убиваемых инструкцией I

  $\boldsymbol{y}$ – множество определений, достигающих точки p_2

- ⧫ $gen_I = \{d\}$

- ⧫ для определения $kill_I$ нужно иметь **все другие определения** $\mathtt{u}$, т.е. всю процедуру (а иногда и всю программу)

2.3 Достигающие определения

2.3.3 Передаточные функции для достигающих определений

◊ Рассмотрим инструкцию I

$$d\text{: } \mathbf{u = v + w}$$

расположенную между точками p_1 и p_2 программы.

- По определению передаточной функции

$$\boldsymbol{y} = f_I(\boldsymbol{x})$$

- Инструкция I сначала убивает все предыдущие определения **u** , а потом порождает d – новое определение **u**. Следовательно

$$\boldsymbol{y} = gen_I \cup (\boldsymbol{x} - kill_I)$$

- Следовательно, передаточная функция f_I инструкции I может быть записана в виде:

$$f_I(x) = gen_I \cup (x - kill_I)$$

2.3 Достигающие определения

2.3.4. Передаточные функции вида *gen-kill*

◊ **Определение**.

Передаточные функции, определяемые соотношением

$$f(x) = gen \cup (x - kill)$$

будем называть передаточными функциями вида gen-$kill$.

◊ **Утверждение 1**.

Композиция двух функций вида gen-$kill$ является функцией вида gen-$kill$.

2.3 Достигающие определения

2.3.4. Передаточные функции вида *gen-kill*

◊ **Утверждение 2**.

Пусть базовый блок B содержит n инструкций, каждая из которых имеет передаточную функцию $f_i(x) = gen_i \cup (x - kill_i)$ $i = 1, 2, ..., n$. Тогда передаточная функция для базового блока B может быть записана как

$$f_B(x) = gen_B \cup (x - kill_B) \quad ,$$

где

$$kill_B = kill_1 \cup kill_2 \cup ... \cup kill_n$$

$$gen_B = gen_n \cup (gen_{n-1} - kill_n) \cup (gen_{n-2} - kill_{n-1} - kill_n) \cup ...$$

$$\cup (gen_1 - kill_2 - kill_3 - ... - kill_n)$$

2.3 Достигающие определения

2.3.5. Передаточные функции вида *gen-kill*

◊ Если какая-либо переменная определяется в блоке B несколько раз, то в gen_B войдет только ее последнее определение,
т.е.
только последнее определение переменной будет действительно вне блока.

2.3 Достигающие определения

2.3.6. Система уравнений

◊ Таким образом, для каждого базового блока B_i можно выписать уравнение

$$Out[B_i] = f_B(In[B_i])$$

или в случае анализа достигающих определений

$$Out[B_i] = gen_B \cup (In[B_i] - kill_B)$$

◊ Если граф потока содержит n базовых блоков,

получится n уравнений относительно $2 \cdot n$ неизвестных

$In[B_i]$ и $Out[B_i]$, $i = 1, 2, \ldots, n$.

◊ Еще n уравнений получится с помощью сбора вкладов путей.

2.3 Достигающие определения

2.3.6 Сбор вкладов путей

◊ Определение достигает точки программы, тогда и только тогда, когда существует по крайней мере один путь, вдоль которого эта точка может быть достигнута.

Следовательно, с одной стороны

$$\forall P \in Pred(B) : Out[P] \subseteq In[B]$$

откуда

$$\bigcup_{P \in Pred(B)} Out[P] \subseteq In[B]$$

а с другой стороны

$$In[B] \subseteq \bigcup_{P \in Pred(B)} Out[P]$$

Таким образом

$$In[B] = \bigcup_{P \in Pred(B)} Out[P]$$

2.3 Достигающие определения

2.3.7 Итеративный алгоритм для вычисления достигающих определений

◊ Получается система уравнений

$$Out_{RD}[B_i] = gen_{B_i} \cup (In_{RD}[B_i] - kill_{B_i})$$

$$In_{RD}[B_i] = \bigcup_{P \in Pred(B_i)} Out_{RD}[P]$$

($i = 1, 2, \ldots, n$).

(RD - *Reaching definitions*)

◊ $In_{RD}[B]$ – множество переменных, определенных на входе в блок B

◊ $Out_{RD}[B]$ – множество переменных, определенных на выходе из блока B

◊ В дальнейшем индекс RD будет опускаться

2.3.7 Итеративный алгоритм для вычисления достигающих определений

◊ Полученную систему уравнений

$$Out[B_i] = gen_{B_i} \cup (In[B_i] - kill_{B_i})$$

$$In[B_i] = \bigcup_{P \in Pred(B_i)} Out[P]$$

($i = 1, 2, \ldots, n$) будем решать методом итераций.

◊ При этом потребуется граничное условие («самый первый $Out[B]$»)

$$Out[Entry] = \varnothing$$

◊ В качестве начальных приближений $Out[B_i]$ можно взять пустые множества: $(Out[B_i])^0 = \varnothing$.

2.3 Достигающие определения

2.3.7 Итеративный алгоритм для вычисления достигающих определений

◊ ***Алгоритм «Достигающие определения»***

- **Вход**: граф потока, в котором для каждого базового блока B_i вычислены множества $kill_{B_i}$ и gen_{B_i}
- **Выход**: множества достигающих определений для входа и выхода каждого блока B_i графа потока: $In[B_i]$ и $Out[B_i]$ $(i = 1, 2, \ldots, n)$
- **Метод**: Используется метод итераций с начальным приближением $(Out[B_i])^0 = \varnothing$.

  Итерации продолжаются до тех пор, пока множества $(In[B_i])^r$ и $(Out[B_i])^r$ (r – номер итерации) не перестанут изменяться.

2.3 Достигающие определения

2.3.7 Итеративный алгоритм для вычисления достигающих определений

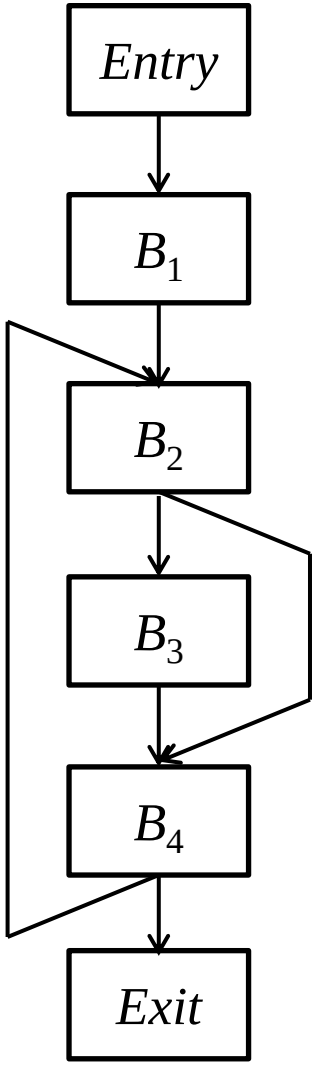
```
1)  Out[Entry] = ∅;
3)  change = true;
2)  /*установка начального значения множества Out */
4)  for (каждый базовый блок B, отличный от Entry)
        Out [B] = ∅;
5)      /* основной цикл*/
6)      while (change) do {
7)              change = false;
8)              for (каждый базовый блок B, отличный от Entry) {
9)              /* вычисление новых значений Out[B] и In[B] */
10)                 OutNew[B] = gen_B ∪ (In[B] − kill_B)
11)                 In[B] = ⋃_{P∈Pred[B]} Out[P]
12)             if (OutNew[B] ≠ Out[B]){
13)                     Out[B] = OutNew[B];
14)                     change = true;
15)             }
16)     }
17) }
```

2.3 Достигающие определения

2.3.8 Пример

B_1
`i ← -, m, 1`
`j ← n`
`a ← u1`
	B_2
	`i ← +, i, 1`
	`j ← -, j, 1`

B_3 `a ← u2`	B_4 `i ← u3`

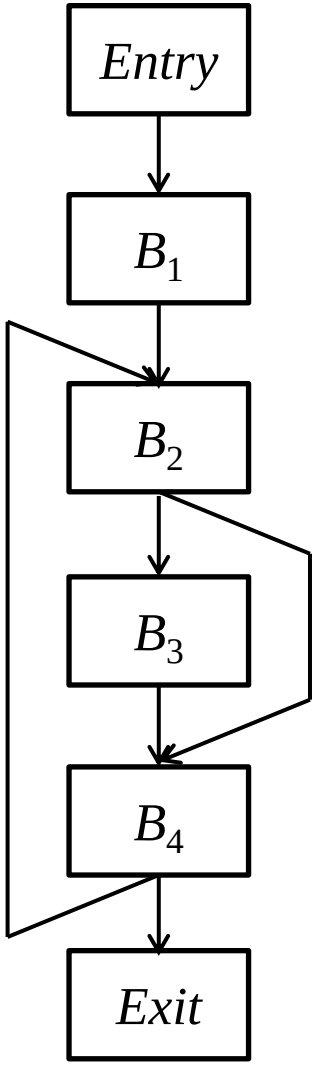


2.3 Достигающие определения

2.3.8 Пример

B_1
`i ← -, m, 1`
`j ← n`
`a ← u1`
	B_2
	`i ← +, i, 1`
	`j ← -, j, 1`

B_3 `a ← u2`	B_4 `i ← u3`



Вычислим множества gen и $kill$ для каждого базового блока

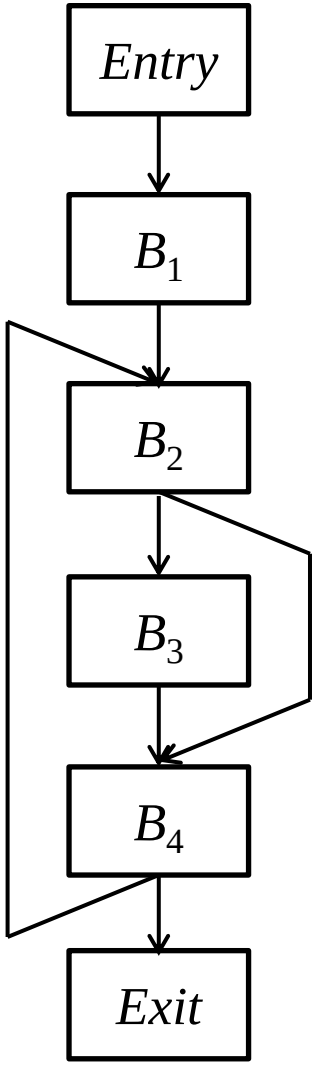
Для блока B_1 $gen_{B_1} = \{d_1, d_2, d_3\}$ $kill_{B_1} = \{d_4, d_5, d_6, d_7\}$

2.3 Достигающие определения

2.3.8 Пример

B_1
`i ← -, m, 1`
`j ← n`
`a ← u1`
	B_2
	`i ← +, i, 1`
	`j ← -, j, 1`

B_3 `a ← u2`	B_4 `i ← u3`



Вычислим множества *gen* и *kill* для каждого базового блока

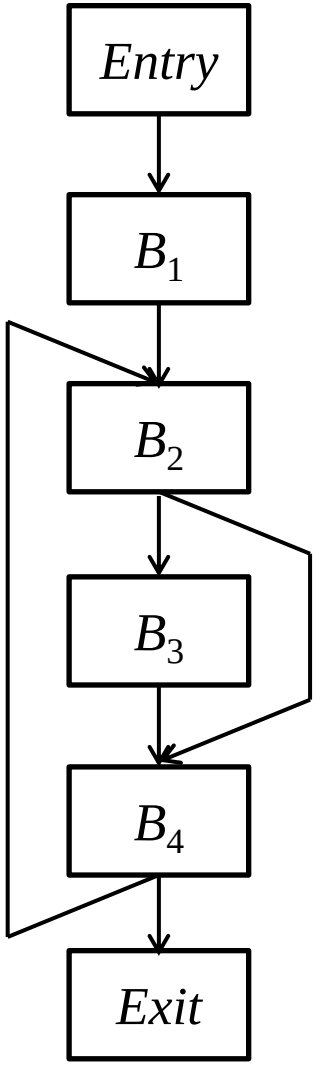
Результаты вычислений сведены в таблицу

B	gen_B	$kill_B$
B_1	$\{(i,B_1), (j,B_1), (a,B_1)\}$ = **(1110000)**	$\{(i,B_2), (j,B_2), (a,B_3), (i,B_4)\}$ = **(0001111)**
B_2	$\{(i,B_2), (j,B_2)\}$ = **(0001100)**	$\{(i,B_1), (j,B_1), (i,B_4)\}$ = **(1100001)**
B_3	$\{(a,B_3)\}$ = **(0000010)**	$\{(a,B_1)\}$ = **(0010000)**
B_4	$\{(i,B_4)\}$ = **(0000001)**	$\{(i,B_1), (i,B_4)\}$ = **(1001000)**

2.3 Достигающие определения

2.3.8 Пример

B	gen_B	$kill_B$
B_1	$\{(i,B_1), (j,B_1), (a,B_1)\}$ = **(1110000)**	$\{(i,B_2), (j,B_2), (a,B_3), (i,B_4)\}$ = **(0001111)**
B_2	$\{(i,B_2), (j,B_2)\}$ = **(0001100)**	$\{(i,B_1), (j,B_1), (i,B_4)\}$ = **(1100001)**
B_3	$\{(a,B_3)\}$ = **(0000010)**	$\{(a,B_1)\}$ = **(0010000)**
B_4	$\{(i,B_4)\}$ = **(0000001)**	$\{(i,B_1), (i,B_4)\}$ = **(1001000)**



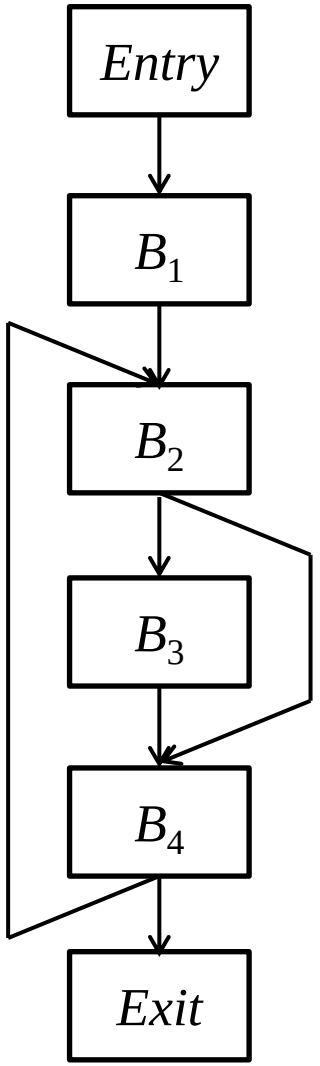
Начальное приближение

$(Out[B_i])^0 = \varnothing$= **(0000000)** , i = 1,2,3,4.

2.3 Достигающие определения

2.3.8 Пример

B	gen_B	$kill_B$
B_1	$\{(i,B_1), (j,B_1), (a,B_1)\}$ = **(1110000)**	$\{(i,B_2), (j,B_2), (a,B_3), (i,B_4)\}$ = **(0001111)**
B_2	$\{(i,B_2), (j,B_2)\}$ = **(0001100)**	$\{(i,B_1), (j,B_1), (i,B_4)\}$ = **(1100001)**
B_3	$\{(a,B_3)\}$ = **(0000010)**	$\{(a,B_1)\}$ = **(0010000)**
B_4	$\{(i,B_4)\}$ = **(0000001)**	$\{(i,B_1), (i,B_4)\}$ = **(1001000)**



Начальное приближение

$(Out[B_i])^0 = \varnothing$= **(0000000)** , i = 1,2,3,4.

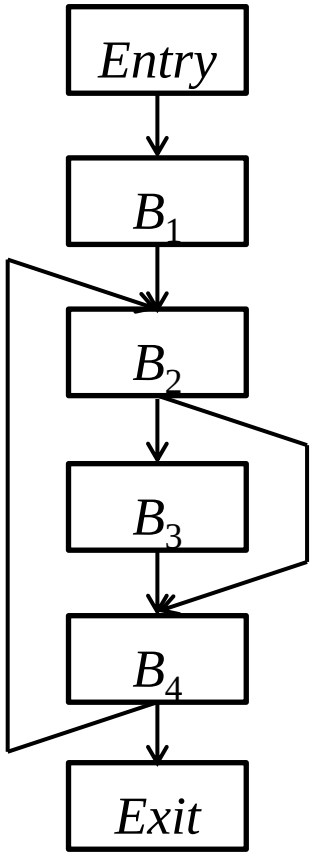
$(In[B_i])^0$ вычисляется по формуле $(In[B])^0 = \bigcup_{P \in Pred[B]} (Out[P])^0$

Например, $(In[B_2])^0 = (Out[B_1])^0 \cup (Out[B_4])^0 = \varnothing \cup \varnothing = \varnothing$

2.3 Достигающие определения

2.3.8 Пример

B	gen_B	$kill_B$
B_1	$\{(i,B_1), (j,B_1), (a,B_1)\}$ = **(1110000)**	$\{(i,B_2), (j,B_2), (a,B_3), (i,B_4)\}$ = **(0001111)**
B_2	$\{(i,B_2), (j,B_2)\}$ = **(0001100)**	$\{(i,B_1), (j,B_1), (i,B_4)\}$ = **(1100001)**
B_3	$\{(a,B_3)\}$ = **(0000010)**	$\{(a,B_1)\}$ = **(0010000)**
B_4	$\{(i,B_4)\}$ = **(0000001)**	$\{(i,B_1), (i,B_4)\}$ = **(1001000)**



Следующее приближение

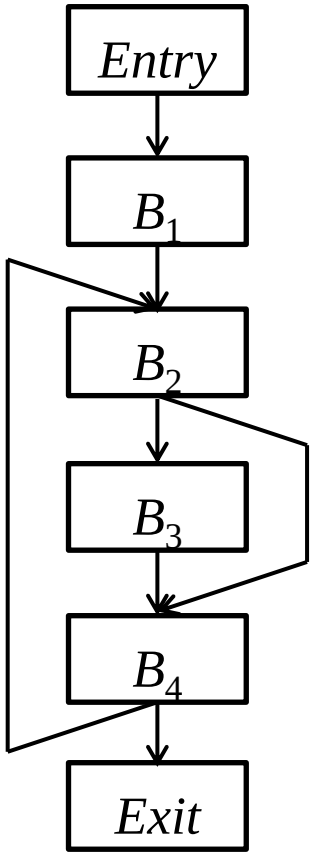
$(In[B_1])^1 = Out[Entry] = \varnothing =$ **(0000000)**

$(Out[B_1])^1 = gen_{B_1} \cup ((In(B_1))^1 - kill_{B_1}) = gen_{B_1} =$**(1110000)**

2.3 Достигающие определения

2.3.8 Пример

B	gen_B	$kill_B$
B_1	$\{(i,B_1), (j,B_1), (a,B_1)\}$ = **(1110000)**	$\{(i,B_2), (j,B_2), (a,B_3), (i,B_4)\}$ = **(0001111)**
B_2	$\{(i,B_2), (j,B_2)\}$ = **(0001100)**	$\{(i,B_1), (j,B_1), (i,B_4)\}$ = **(1100001)**
B_3	$\{(a,B_3)\}$ = **(0000010)**	$\{(a,B_1)\}$ = **(0010000)**
B_4	$\{(i,B_4)\}$ = **(0000001)**	$\{(i,B_1), (i,B_4)\}$ = **(1001000)**



Следующее приближение

$(In[B_1])^1 = Out[Entry] = \varnothing =$ **(0000000)**

$(Out[B_1])^1 = gen_{B_1} \cup ((In(B_1))^1 - kill_{B_1}) = gen_{B_1} =$ **(1110000)**

$(In[B_2])^1 = (Out[B_1])^1 \cup (Out[B_4])^0 =$ **(1110000)** $\vee$ **(1100001)** = **(1110000)**

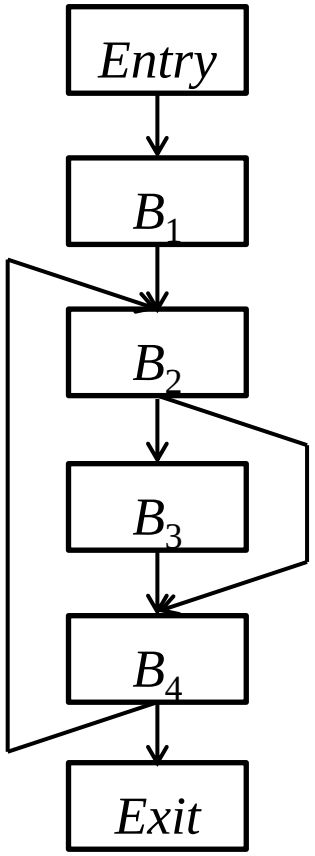
$(Out[B_2])^1 = gen_{B_2} \cup ((In(B_2))^1 - kill_{B_2}) =$ **(0001100)** $\vee$ **((1110000)** − **(1100001)** =

= **(0011100)**.

2.3 Достигающие определения

2.3.8 Пример

B	gen_B	$kill_B$
B_1	$\{(i,B_1), (j,B_1), (a,B_1)\}$ = (1110000)	$\{(i,B_2), (j,B_2), (a,B_3), (i,B_4)\}$ = (0001111)
B_2	$\{(i,B_2), (j,B_2)\}$ = (0001100)	$\{(i,B_1), (j,B_1), (i,B_4)\}$ = (1100001)
B_3	$\{(a,B_3)\}$ = (0000010)	$\{(a,B_1)\}$ = (0010000)
B_4	$\{(i,B_4)\}$ = (0000001)	$\{(i,B_1), (i,B_4)\}$ = (1001000)



Следующее приближение

$(In[B_1])^1 = Out[Entry] = \varnothing =$ (0000000)

$(Out[B_1])^1 = gen_{B_1} \cup ((In[B_1])^1 - kill_{B_1}) = gen_{B_1} =$ (1110000)

$(In[B_2])^1 = (Out[B_1])^1 \cup (Out[B_4])^0 =$ (1110000)$\vee$(1100001) = (1110000)

$(Out[B_2])^1 = gen_{B_2} \cup ((In[B_2])^1 - kill_{B_2}) =$ (0001100)$\vee$((1110000)$-$(1100001)=

= (0011100)

Дальнейшие вычисления ведутся аналогично.

2.3 Достигающие определения

2.3.8 Пример

B	$(Out[B])^0$	$(In[B])^1$	$(Out[B])^1$	$(In[B])^2$	$(Out[B])^2$	$(In[B])^3$	$(Out[B])^3$
B_1	0000000	0000000	1110000	0000000	1110000	0000000	1110000
B_2	0000000	1110000	0011100	1110111	0011110	1110111	0011110
B_3	0000000	0011100	0001110	0011110	0011110	0011110	0011110
B_4	0000000	0011111	0010111	0011110	0010111	0011110	0010111
Exit	0000000	0010111	0010111	0010111	0010111	0010111	0010111

Процесс завершается, так как $((In[B])^3, (Out[B])^3)$ совпадают с $((In[B])^2, (Out[B])^2)$

2.3.9 Множества *Input* для базовых блоков

◊ Множество $Input[B]$ для базового блока B – это множество $In_{RD}[B]$, которое строится при исследовании достигающих определений

2.4 Живые переменные

2.4.1 Множества *Output* для базовых блоков

◊ Множества *Output* для базовых блоков строятся как результат анализа, позволяющего выявить *живые переменные*, т.е. переменные, используемые в базовых блоках, в которые управление попадает после выхода из исследуемого базового блока.

◊ Анализ очень похож на предыдущий, но ГПУ просматривается не с начала, а с конца: от *Exit* к *Entry*.

2.4.2. Определение

◊ Цель анализа – для определения переменной x в точке p программы выяснить, будет ли указанное значение x использоваться вдоль какого-нибудь пути, начинающемся в точке p.

- ◈ Если да – переменная x *жива* (активна) в точке p,
- ◈ если нет – переменная x *мертва* (неактивна) в точке p.

2.4 Живые переменные

2.4.3 Уравнения потока данных

◊ $In_{LV}[B]$ – множество переменных, **живых** на входе в блок B

$Out_{LV}[B]$ – множество переменных, **живых** на выходе из блока B .

- def_B – множество переменных, определяемых в блоке B до их использования в этом блоке

  (любая переменная из def_B мертва на входе в блок B)

- use_B – множество переменных, используемых в блоке B до их определения в этом блоке

  (любая переменная из use_B жива на входе в блок B)

2.4 Живые переменные

2.4.3 Уравнения потока данных

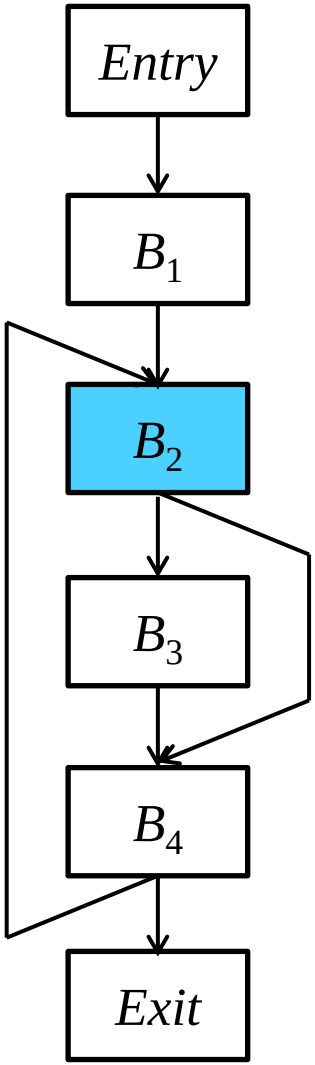
◊ **Пример**

(1) в блоке B_2 переменные i и j используются до их переопределения, следовательно,

$$use_{B_2} = \{(i, B_1), (i, B_4), (j, B_1)\} = \mathbf{(1100001)}$$

(2) в блоке B_2 определяются новые значения переменных i и j, так что

$$def_{B_2} = \{(i, B_2), (j, B_2)\} = \mathbf{(0001100)}$$



B_1 i ← −, m, 1 j ← n a ← u1	B_2 i ← +, i, 1 j ← -, j, 1 a
B_3 a ← u2	B_4 i ← u3

2.4 Живые переменные

2.4.3 Уравнения потока данных

◊ Уравнения, связывающие def и use с неизвестными In и Out, определяются следующим образом:

$$In[B] = use_B \cup (Out[B] - def_B)$$

$$Out[B] = \bigcup_{S \in Succ(B)} In[S]$$

◊ К ним добавляется граничное условие

$$In\ [Exit] = \varnothing$$

2.4 Живые переменные

2.4.4 Итеративный алгоритм анализа живых переменных

◊ ***Алгоритм «Живые переменные»***

- **Вход**: граф потока, в котором для каждого блока B вычислены множества def и use
- **Выход**: множества переменных, живых на входе ($In[B]$) и выходе ($Out[B]$) каждого базового блока B графа потока.
- **Метод**: выполнить следующую программу:

```
1)  In[Exit] = ∅;
2)  change = true;
3)  /*установка начального значения множества In*/
4)      for (каждый базовый блок B, отличный от Exit)
            In[B] = ∅;
5)  /* основной цикл*/
6)      while (change) do {
7)     change = false;
8)         for (каждый базовый блок B, отличный от Exit) {
9)              /* вычисление новых значений множеств
                In[B],Out[B] */
10)             In[B] = ⋃ In[S]   (S ∈ Succ(B))
11)             InNew[B] = use_B ∪ (Out[B] − def_B)
12)             if (InNew[B] ≠In[B] ){
13)                 In[B] = InNew[B] ;
14)                 change = true;
15)             }
16)         }
17)     }
```

2.4 Живые переменные

2.4.5 Множества $Output$ для базовых блоков

◊ Множество $Output[B]$ для базового блока B – это множество $Out_{LV}[B]$, которое строится при исследовании живых переменных