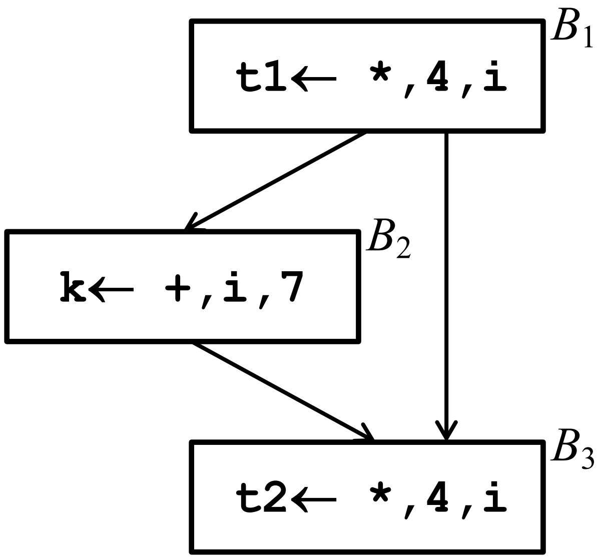
3. Анализ потока данных

3.1 *Доступные выражения*

3.1.1 Определение

◊ Выражение e *доступно* в точке p, если e вычисляется на любом пути от $Entry$ до p причем переменные, входящие в состав e не переопределяются между последним таким вычислением и точкой p.

◊ **Пример**. Пусть точка p – вход в блок B_3.



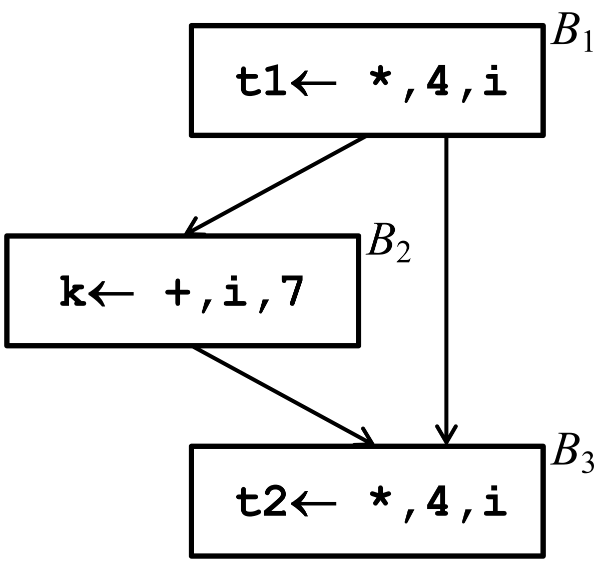
На рисунке присваивание *не убивает* выражение `*,4,i`. Поэтому `t2←*,4,i` можно заменить на `t2← t1`

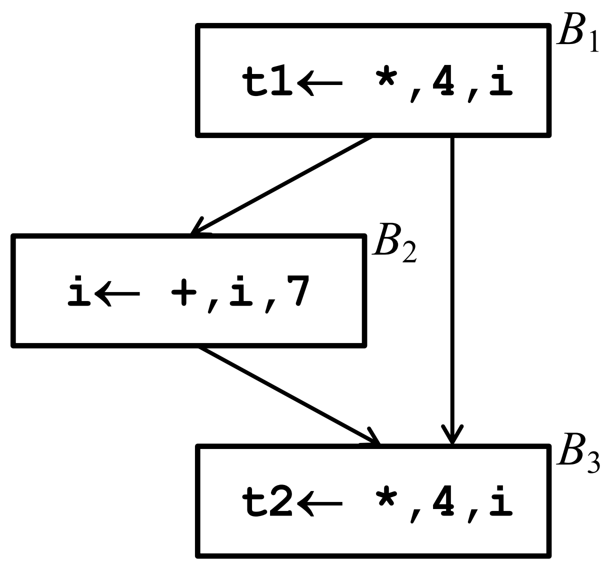
3.1 Доступные выражения

3.1.1 Определение

◊ Выражение *e доступно* в точке *p*, если *e* вычисляется на любом пути от *Entry* до *p* причем переменные, входящие в состав *e* не переопределяются между последним таким вычислением и точкой *p*.

◊ **Пример**. Пусть точка *p* – вход в блок B_3.





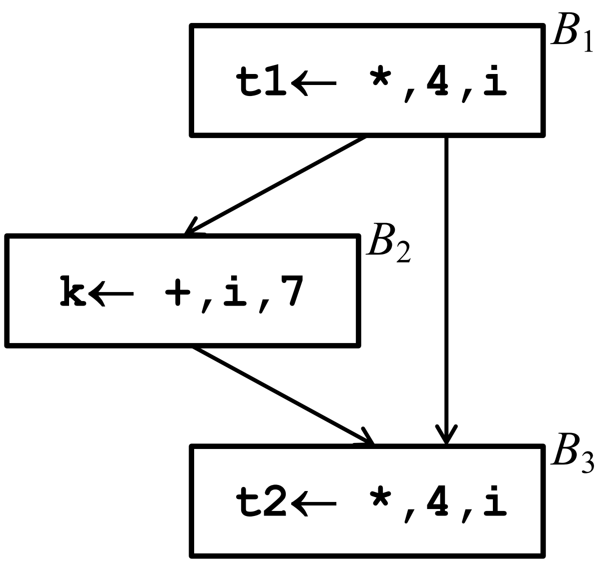
На рисунке присваивание *не убивает* выражение **`*,4,i`**. Поэтому **`t2←*,4,i`** можно заменить на **`t2←t1`**	Присваивание *убивает* выражение **`*,4,i`**. Замена **`t2←*,4,i`** на **`t2←t1`** будет некорректной.

3.1 Доступные выражения

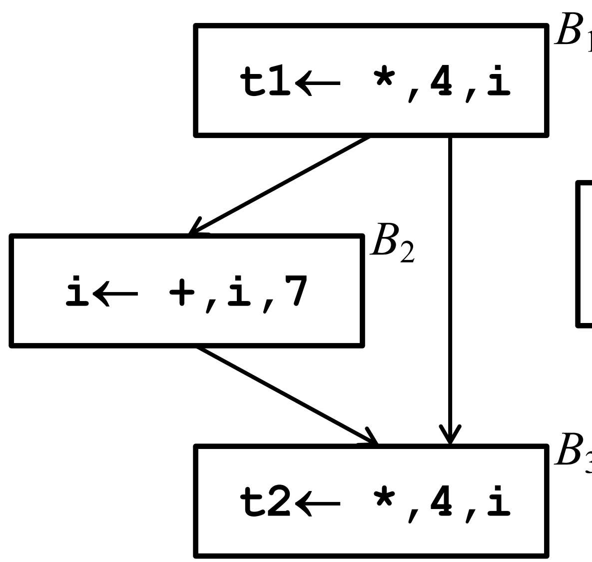
3.1.1 Определение

◊ Выражение *e доступно* в точке p, если e вычисляется на любом пути от $Entry$ до p причем переменные, входящие в состав e не переопределяются между последним таким вычислением и точкой p.

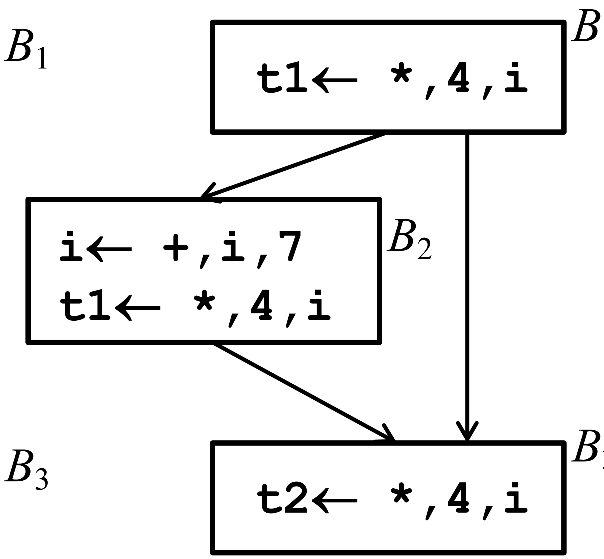
◊ **Пример**. Пусть точка p – вход в блок B_3.



Присваивание *не убивает* выражение **`*,4,i`**. Замена **`t2←*,4,i`** на **`t2←t1`** корректна.



Присваивание *убивает* выражение **`*,4,i`**. Замена **`t2←*,4,i`** на **`t2←t1`** некорректна.



Перевычисление в B_2 выражения **`*,4,i`** обеспечивает коррект-ность замены **`t2←*,4,i`** на **`t2←t1`**

3.1 Доступные выражения

3.1.1 Определение

- ◊ Для каждого базового блока B определим
  - ◊ множество e_kill_B выражений, *убиваемых* в блоке B

    (пример – присваивание **`i ← +,i,7`** в блоке B_2 *убивает* выражение **`*,4,i`**)
  - ◊ множество e_gen_B выражений, *порождаемых* в блоке B

    (пример – присваивание **`t1 ← *,4,i`** в блоке B_2 *порождает* выражение **`*,4,i`**).

3.1 Доступные выражения

3.1.2 Уравнения потока данных

- Для того, чтобы найти доступные выражения, можно использовать метод, напоминающий метод вычисления достигающих определений.
  - U – множество всех выражений программы.
  - $In[B]$ – множество выражений из U, доступных на входе в B,
  - Граничное условие:

$$In[Entry] = \varnothing$$

  - Система уравнений:

$$Out[B] = e\_gen_B \cup (In[B] - e\_kill_B)$$

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

$$In[B_i] = \bigcap_{P \in Pred(B_i)} (e\_gen_P \cup (In[P] - e\_kill_P))$$

3.1 Доступные выражения

3.1.3 Итеративный алгоритм вычисления доступных выражений

- **Вход**: граф потока, в котором для каждого блока B вычислены множества e_gen_B и e_kill_B
- **Выход**: множества выражений, доступных на входе ($In[B]$) каждого базового блока B графа потока.
- **Метод**: выполнить следующую программу

$In[Entry] = \varnothing$;

for (каждый базовый блок B, отличный от $Entry$) $In[B]$ = U;

while (внесены изменения в Out) {

for (каждый базовый блок B, отличный от $Entry$) {

$$In[B] = \bigcap_{P \in Pred(B)} \left(e\_gen_B \cup (In[P] - e\_kill_B)\right)$$

}

}

3.1 Доступные выражения

3.1.3 Итеративный алгоритм вычисления доступных выражений

- **Метод**: выполнить следующую программу

```
In[Entry] = ∅;
WorkList = ∅;
for (каждый базовый блок B,отличный от Entry){
поместить B в WorkList;
In[B] = U;/* Каждому In[B] присваивается значение
                его нулевой итерации */
};
do {              /* основной цикл*/
  Выбрать из очереди WorkList очередной блок B
  Вычислить InNew[B], используя уравнение
```

$$InNew[B] = \bigcap_{P \in Pred(B)} \left(e\_gen_B \cup (In[B] - e\_kill_B) \right)$$

```
   if (InNew[B] ≠ In[B]) {
     In[B] = InNew[B];
     Поместить потомков B в конец очереди WorkList
   }
} while |WorkList|>0;
```

3.2 Полурешетки

3.2.1 Анализ потока данных

◊ При анализе потока данных рассматриваются множества переменных для описания состояния и такие операции как *объединение* ($\cup$) и *пересечение* ($\cap$) множеств.

3.2 Полурешетки

3.2.1 Анализ потока данных

◊ При анализе потока данных рассматриваются множества переменных для описания состояния и такие операции как *объединение* ($\cup$) и *пересечение* ($\cap$) множеств.

◊ Свойства операций $\cup$ и $\cap$:

$A \cup A = A$	$A \cap A = A$	*(идемпотентность)*
$A \cup B = B \cup A$	$A \cap B = B \cap A$	*(коммутативность)*
$A \cup (B \cup C) = (A \cup B) \cup C$	$A \cap (B \cap C) = (A \cap B) \cap C$	*(ассоциативность)*
Если $A \cup B = A$, то $B \subseteq A$	Если $A \cap B = A$, то $B \supseteq A$	отношение частичного порядка, связанное с операцией

3.2 Полурешетки

3.2.1 Анализ потока данных

◊ При анализе потока данных рассматриваются множества переменных для описания состояния и такие операции как *объединение* ($\cup$) и *пересечение* ($\cap$) множеств.

◊ Свойства операций $\cup$ и $\cap$:

$A\cup A = A$	$A\cap A = A$	*(идемпотентность)*
$A\cup B = B\cup A$	$A\cap B = B\cap A$	*(коммутативность)*
$A\cup(B\cup C) = (A\cup B)\cup C$	$A\cap (B\cap C) = (A\cap B) \cap C$	*(ассоциативность)*
Если $A\cup B = A$, то $B\subseteq A$	Если $A\cap B = A$, то $B\supseteq A$	отношение частичного порядка, связанное с операцией

◊ Если множество U содержит все элементы, то любое множество $A \subseteq U$ и $A \cup U = U \cup A = U$

◊ Для пересечения ($\cap$) роль U играет $\varnothing$: любое множество $A \supseteq \varnothing$ и $A \cap \varnothing = \varnothing \cap A = \varnothing$

3.2 Полурешетки

3.2.2 Определение полурешетки

◊ Полурешетка – это абстрактная алгебраическая структура, над элементами которой определена абстрактная операция $\wedge$ (мы будем называть ее «сбор»), обладающая свойствами операций $\cup$ и $\cap$.

◊ **Определение.** *Полурешетка* представляет собой множество L, на котором определена бинарная операция «*сбор*» $\wedge$, такая, что для всех x, y и $z \in L$:

$$x \wedge x = x \qquad (\textit{идемпотентность})$$

$$x \wedge y = y \wedge x \qquad (\textit{коммутативность})$$

$$x \wedge (y \wedge z) = (x \wedge y) \wedge z \qquad (\textit{ассоциативность})$$

Полурешетка имеет *верхний элемент* (или *верх*) $\top \in L$ такой, что для всех $x \in L$ выполняется $\top \wedge x = x$

Полурешетка может иметь (но необязательно) *нижний элемент* (или *низ*) $\bot \in L$: $\forall x \in L \;\; \bot \wedge x = \bot$

3.2 Полурешетки

3.2.3 Полурешеточное отношение частичного порядка $\leq$

◊ Для всех пар $x, y \in L$ определим отношение $\leq$:

$x \leq y$ тогда и только тогда, когда $x \wedge y = x$

$$x \cap y = x \quad \Rightarrow \quad x \subseteq y$$
$$x \cup y = x \quad \Rightarrow \quad y \subseteq x$$

Т.е. отношение $\leq$ имеет противоположный смысл для $\cap$ и $\cup$, являясь обобщением $\subseteq$ или $\supseteq$ для данной задачи анализа потока данных.

3.2 Полурешетки

3.2.3 Полурешеточное отношение частичного порядка $\leq$

◊ Для всех пар $x, y \in L$ определим отношение $\leq$:

$x \leq y$ тогда и только тогда, когда $x \wedge y = x$

◊ Отношение $\leq$ является отношением частичного порядка.

(1) *Рефлексивность* $\leq$ следует из идемпотентности $\wedge$:

$x \leq x \;\Leftrightarrow\; x \wedge x = x$

(2) *Антисимметричность* $\leq$ следует из коммутативности $\wedge$:

пусть $x \leq y$ и $y \leq x$; тогда по определению операции $\leq$:

$x = x \wedge y = y \wedge x = y$

(3) *Транзитивность* $\leq$ следует из ассоциативности $\wedge$:

пусть $x \leq y$ и $y \leq z$; тогда по определению $\leq$

$x \wedge y = x$ и $y \wedge z = y$;

$(x \wedge z) = ((x \wedge y) \wedge z) = (x \wedge (y \wedge z)) = (x \wedge y) = x$,

$(x \wedge z) = x \;\Leftrightarrow\; x \leq z$.

3.2 Полурешетки

3.2.4 Наибольшая нижняя граница

◊ **Определение**. Пусть $\langle L, \leq\rangle$ – частично упорядоченное множество (в частности, полурешетка).

Наибольшей нижней границей $inf\,(x, y)$ элементов x и $y \in L$ называется элемент $g \in L$, такой, что $g \leq x;\ g \leq y;$ и если $z \in L$, такой, что $z \leq x$ и $z \leq y$, то $z \leq g$.

◊ **Утверждение.** Если x и $y \in L$, где $\langle L, \wedge\rangle$ – полурешетка, то $inf\,(x, y) = x \wedge y$.

Доказательство. Пусть $g = x \wedge y$. Тогда

$g \wedge x = ((x \wedge y) \wedge x) = (x \wedge (y \wedge x)) = (x \wedge (x \wedge y)) =$
$= ((x \wedge x) \wedge y) = (x \wedge y) = g$, откуда следует $g \leq x$.

Точно таким же образом доказывается, что $g \leq y$.

Пусть $z \in L$, $z \leq x$ и $z \leq y$. Докажем, что $z \leq g$. Имеем:

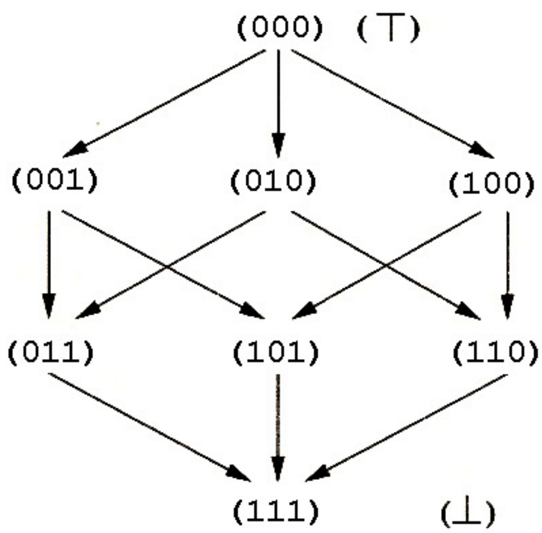
$(z \wedge g) = (z \wedge (x \wedge y)) = ((z \wedge x) \wedge y)$; $z \leq x \Rightarrow (z \wedge x) = z$
$\Rightarrow (z \wedge g) = (z \wedge y) = z. \Rightarrow (z \wedge g) = z \Rightarrow z \leq g.$

3.2 Полурешетки

3.2.5. Диаграммы полурешеток

◊ *Диаграмма полурешетки* $\langle L, \wedge \rangle$ представляет собой граф, узлами которого являются элементы L, а ребра направлены от x к y, если $y \leq x$.

◊ **Пример**. На рисунке – диаграмма полурешетки $\langle U, \cup \rangle$, $|U| = 8$: элемент множества U представляется битовым 3-вектором.



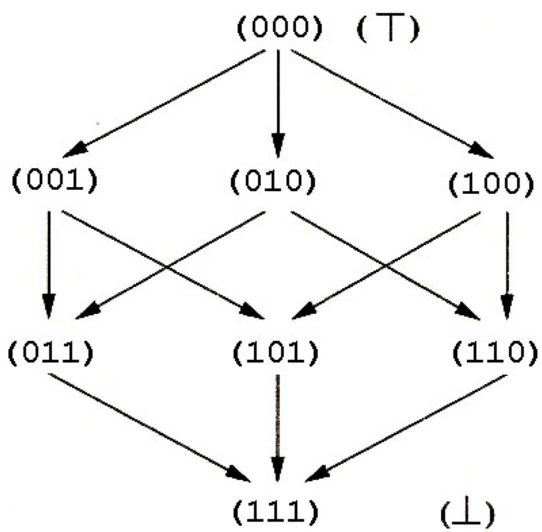
(по определению $\leq$:
x ≤ *y* <=> *x* ∧ *y* = *x)*
В частности, если сбор – это объединение:
x ∪ *y* = *x* => *x* ⊇ *y*
т. е. дуги направлены от *большего* к *меньшему* в смысле *«полурешеточного»* отношения, а в «обычном» смысле получается наоборот (но так будет только для ∪) .

3.2 Полурешетки

3.2.5. Диаграммы полурешеток

◊ *Диаграмма полурешетки* $\langle L, \wedge \rangle$ представляет собой граф, узлами которого являются элементы L, а ребра направлены от x к y, если $y \le x$.

◊ **Пример**. На рисунке – диаграмма полурешетки $\langle U, \cup \rangle$, $|U| = 8$: элемент множества U представляется битовым 3-вектором.



◊ **Замечание.** Определение элемента $\perp$ внизу диаграммы: для любого $x \in L$: $\perp \wedge x = \perp$
Этот элемент называется «низом», так как по определению отношения $\le$ для любого $x \in L$ $\perp \le x$.
Полурешетка с операцией $\wedge$ не обязательно содержит $\perp$.

3.2 Полурешетки

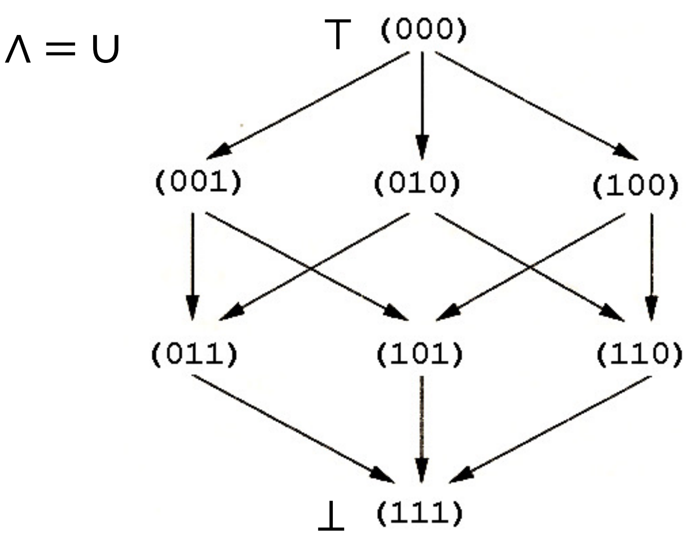
3.2.5.1. Примеры полурешеток

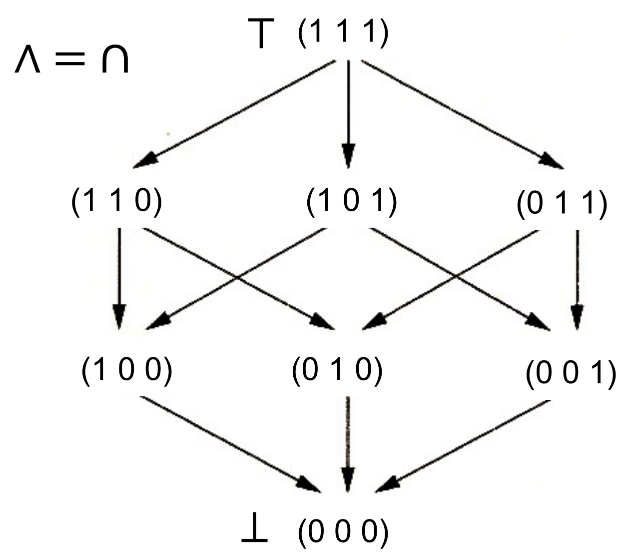
◊ *Достигающие определения и живые переменные:*

$\wedge = \cup$ (сбор $-$ объединение), $\perp = (\,1\ 1\ 1\,)$, $\top = (\,0\ 0\ 0\,)$

◊ *Доступные выражения:*

$\wedge = \cap$ (сбор $-$ пересечение), $\perp = (\,0\ 0\ 0\,)$, $\top = (\,1\ 1\ 1\,)$





«⊥» соответствует консервативному решению, при котором оптимизацию делать нельзя (все определения достигают точки, все переменные живы, ни одно выражение не доступно)

3.2.6 Наименьшая верхняя граница

- В частично упорядоченном множестве $\langle L, \leq \rangle$ можно по аналогии с $inf\,(x, y)$ определить наименьшую верхнюю границу $sup\,(x, y)$ как такой $b \in L$, что $x \leq b, y \leq b$, и если для $z \in L$ $x \leq z$ и $y \leq z$, то $b \leq z$.
  - В полурешетке $\langle L, \wedge \rangle$ $inf\,(x, y)$ существует для любой пары элементов x и $y \in L$, а $sup\,(x, y)$ нет.
  - В частично упорядоченном множестве $\langle L, \leq \rangle$, в котором для любых x и $y \in L$ существует $sup\,(x, y)$, можно определить бинарную операцию $\vee$ (объединение) $x \vee y = sup\,(x, y)$.
  - Множество, в котором определены обе операции – $\wedge$ (сбор) и $\vee$ (объединение), – называется *решеткой*.

3.3 Структура потока данных

3.3.1 Определение

◊ **Определение**. *Структурой потока данных* называется четверка

$$\langle D, F, L, \wedge \rangle ,$$

где

- D – направление анализа (*Forward* или *Backward*),
- F – семейство передаточных функций,
- L – поток данных (множество элементов полурешетки),
- $\wedge$ - реализация операции сбора.

◊ **Примеры.**

1 Структура потока данных для анализа **достигающих определений**: $\langle$***Forward***, $\mathcal{GK}$, ***Def***, $\cup\rangle$,

где $\mathcal{GK}$ – семейство передаточных функций вида *gen-kill*,

Def – множество определений переменных.

3.3 Структура потока данных

3.3.1 Определение

◊ **Примеры.**

2 Структура потока данных для анализа **живых переменных**:

$\langle \mathbf{\mathit{Backward}}, \mathcal{LV}, \mathbf{\mathit{Var}}, \cup \rangle$,

где $\mathcal{LV}$ – семейство передаточных функций вида,

$$f(x) = use \cup (x - def)$$

Def – множество определений переменных.

3 Структура потока данных для анализа **доступных выражений**:

$\langle \mathbf{\mathit{Forward}}, \mathbf{\mathit{AE}}, \mathbf{\mathit{Expr}}, \cap \rangle$,

где $\mathcal{AE}$ – семейство передаточных функций вида

$$f(x) = e\_gen \cup (x - e\_kill),$$

$Expr$ – множество выражений программы.

3.3 Структура потока данных

3.3.2 Замкнутость

- **Определение**. Семейство передаточных функций F называется *замкнутым*, если:
  - F содержит тождественную функцию I: $\forall x \in L$: $I(x) = x$.
  - F замкнуто относительно композиции:
    $\forall\, f, g \in F \Rightarrow h(x) = g(f(x)) \in F$.

- **Утверждение 1**. Семейство передаточных функций F, используемое при анализе достигающих определений (передаточные функции вида *gen-kill*), является замкнутым.

3.3 Структура потока данных

3.3.2 Замкнутость

◊ **Утверждение 1**. Семейство передаточных функций F, используемое при анализе достигающих определений (передаточные функции вида $gen\text{-}kill$), является замкнутым.

1) Замкнутость относительно композиции уже установлена.

2) Тождественная функция $I(x) = x$ является функцией вида $gen\text{-}kill$ с $gen = kill = \varnothing$.

3.3 Структура потока данных

3.3.2 Замкнутость

- **Утверждение 1**. Семейство передаточных функций F, используемое при анализе достигающих определений (передаточные функции вида $gen\text{-}kill$), является замкнутым.

  1) Замкнутость относительно композиции уже установлена.

  2) Тождественная функция $I(x) = x$ является функцией вида $gen\text{-}kill$ с $gen = kill = \varnothing$.

- **Утверждение 2**. Семейство передаточных функций, используемое при анализе живых переменных является замкнутым.

- **Утверждение 3**. Семейство передаточных функций, используемое при анализе доступных выражений является замкнутым.

3.3 Структура потока данных

3.3.3 Монотонные структуры

◊ **Определение 1**. Структура потока данных $\langle D, F, L, \wedge\rangle$ называется *монотонной,* если $\forall x, y \in L, \forall f \in F\ (x \le y) \Rightarrow f(x) \le f(y)$.

◊ **Определение 2**. Структура потока данных $\langle D, F, L, \wedge\rangle$ называется *монотонной,* если $\forall x, y \in L, \forall f \in F\ \ f(x \wedge y) \le f(x) \wedge f(y)$.

3.3 Структура потока данных

3.3.3 Монотонные структуры

- **Определение 1**. Структура потока данных $\langle D, F, L, \wedge\rangle$ называется *монотонной,* если $\forall x, y \in L, \forall f \in F\ (x \le y) \Rightarrow f(x) \le f(y)$.
- **Определение 2**. Структура потока данных $\langle D, F, L, \wedge\rangle$ называется *монотонной,* если $\forall x, y \in L, \forall f \in F\ \ f(x \wedge y) \le f(x) \wedge f(y)$.
- **Утверждение.** Определения 3.3.3.1 и 3.3.3.2 эквивалентны.

3.3 Структура потока данных

3.3.3 Монотонные структуры

- **Определение 1**. Структура потока данных $\langle D, F, L, \wedge\rangle$ называется *монотонной*, если $\forall x, y \in L, \forall f \in F\ x \leq y \Rightarrow f(x) \leq f(y)$.

- **Определение 2**. Структура потока данных $\langle D, F, L, \wedge\rangle$ называется *монотонной*, если $\forall x, y \in L, \forall f \in F\ \ f(x \wedge y) \leq f(x) \wedge f(y)$.

- **Утверждение.** Определения 3.2.3.1 и 3.2.3.2 эквивалентны.

  - Из определения 1 следует определение 2:

    $x \wedge y = inf(x, y) \Rightarrow x \wedge y \leq x$ и $x \wedge y \leq y \Rightarrow (1) \Rightarrow$

    $f(x \wedge y) \leq f(x)\ u\ f(x \wedge y) \leq f(y) \Rightarrow$

    $f(x \wedge y) = inf(f(x), f(y)) = f(x) \wedge f(y)$

  - Из определения 2 следует определение 1:

    $x \leq y \Rightarrow x \wedge y = x \Rightarrow_{(\text{опр2})} f(x) \leq f(x) \wedge f(y),$

    $f(x) \wedge f(y) = inf(f(x), f(y)) \leq f(y); \Rightarrow f(x) \leq f(y)$

3.3 Структура потока данных

3.3.4 Дистрибутивные структуры

◊ **Определение**. Структура потока данных $\langle D, F, L, \wedge\rangle$ называется *дистрибутивной,* если

$$\forall x,\, y \in L,\, \forall f \in F: \quad f(x \wedge y) = f(x) \wedge f(y).$$

3.3 Структура потока данных

3.3.4 Дистрибутивные структуры

◊ **Определение**. Структура потока данных $\langle D, F, L, \wedge\rangle$ называется *дистрибутивной,* если

$$\forall x, y \in L, \forall f \in F: \quad f(x \wedge y) = f(x) \wedge f(y).$$

◊ **Утверждение.** Если структура потока данных $\langle D, F, L, \wedge\rangle$ дистрибутивна, то она монотонна.

$$\boldsymbol{a} = \boldsymbol{b} \Rightarrow_{\textit{идемпотентность}} a \wedge b = a \Rightarrow \boldsymbol{a} \leq \boldsymbol{b}$$

$$f(x \wedge y) = f(x) \wedge f(y) \Rightarrow f(x \wedge y) \leq f(x) \wedge f(y)$$

◊ Обратное утверждение неверно.

В качестве доказательства можно привести пример монотонной структуры потока данных, которая не дистрибутивна.

3.3 Структура потока данных

3.3.5. Дистрибутивность структуры достигающих определений

◊ **Утверждение.** Структура достигающих определений $RD = \langle Forward,\ Gen\text{-}kill,\ \varnothing,\ \cup \rangle$ дистрибутивна

◊ $y, z \in RD, f(x) = G \cup (x - K) \in Gen\text{-}kill.$

Докажем, что

$$G \cup ((y \cup z) - K) = (G \cup (y - K)) \cup (G \cup (z - K))$$

(1) $y, z \in G$: равенство выполняется, так как G входит и в левую, и в правую его части.

(2) $y, z \notin G$: G можно исключить из равенства:

$$(y \cup z) - K = (y - K)) \cup (z - K)$$

Это равенство проверяется при помощи диаграмм Венна.

3.3 Структура потока данных

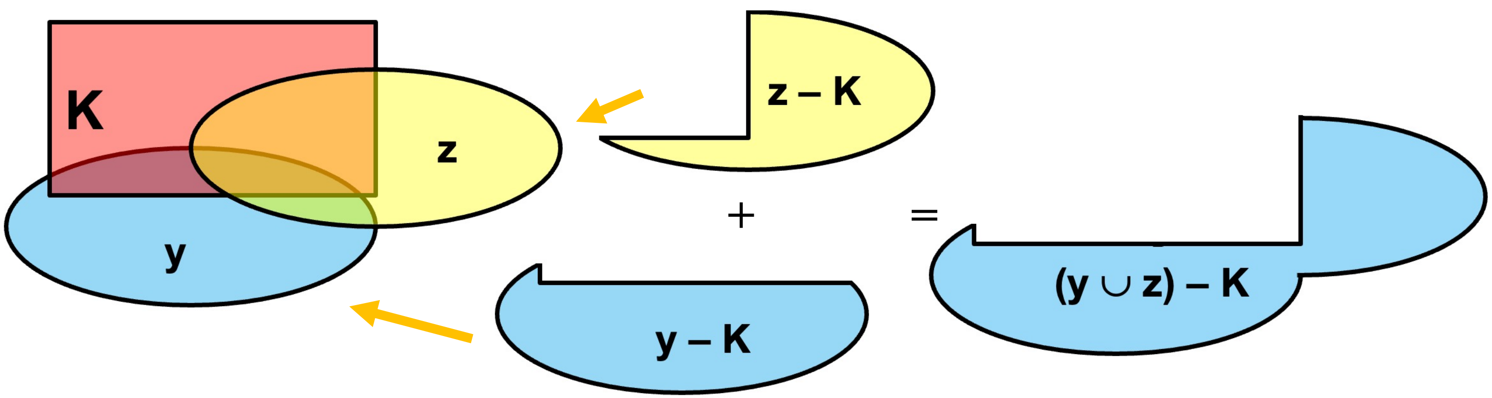
3.3.5. Дистрибутивность структуры достигающих определений

◊ **Утверждение.** Структура достигающих определений $RD = \langle Forward, Gen\text{-}kill, \varnothing, \cup \rangle$ дистрибутивна

◊ $y, z \in RD, f(x) = G \cup (x - K) \in Gen\text{-}kill.$
Докажем, что
$G \cup ((y \cup z) - K) = (G \cup (y - K)) \cup (G \cup (z - K))$

(2) $y, z \notin G$: G можно исключить из равенства:
$(y \cup z) - K = (y - K) \cup (z - K)$

Это равенство проверяется при помощи диаграмм Венна.



3.3 Структура потока данных

3.3.6. Дистрибутивность структур живых переменных и доступных выражений

◊ **Следствие 1.** Структура живых переменных

$LV = \langle Backward, Def\text{-}use, \varnothing, \cup\rangle$

дистрибутивна

◊ $f(x) \in Def\text{-}use$ алгебраически подобна функции класса $Gen\text{-}kill$.

◊ **Следствие 2.** Структура доступных выражений

$AE = \langle Forward, Gen\text{-}kill, U, \cap\rangle$

дистрибутивна

3.4 Обобщенный итеративный алгоритм

3.4.1 Описание алгоритма

◊ **Алгоритм.** Итеративное решение задачи анализа потока данных

- **Вход**: граф потока управления,
  структура потока данных $\langle D, F, L, \wedge\rangle$,
  передаточная функция $f_B \in F$
  константа из $l \in L$ для граничного условия
- **Выход**: значения из L для $In[B]$ и $Out[B]$ для каждого блока B в графе потока.
- **Метод**: если $D = Forward$ выполнить программу (3.4.2);
  если $D = Backward$ выполнить программу (3.4.3).

3.4 Обобщенный итеративный алгоритм

3.4.2 Решение задачи потока данных ($D = Forward$)

$Out[Entry] = l;$

for ($each\ B \neq Entry$) $Out[B] = \top;$

while (внесены изменения в Out)

for ($each\ B \neq Entry$) {

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

}

3.4 Обобщенный итеративный алгоритм

3.4.3 Решение задачи потока данных ($D = Backward$)

$In[Exit] = l;$

for $(each\ B \neq Exit)\ \ In[B] = \top;$

while (внесены изменения в In**)**

for $(each\ B \neq Exit)$ {

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

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

}

3.5 Свойства итеративного алгоритма

3.5.1 Сходимость к решению

◊ **Утверждение.** Если обобщенный итеративный алгоритм сходится, то получающийся результат является решением уравнений потоков данных.

◊ $D = Forward$:

Если после очередной итерации цикла **`while`** хотя бы для одного B уравнение $Out[B] = f_B(In[B])$ не удовлетворяется, то для этого B

$$OutNew[B] \neq Out[B].$$

Следовательно, в множество $Out[B]$ будет внесено изменение и $change$ не позволит выйти из цикла.

3.5 Свойства итеративного алгоритма

3.5.1 Сходимость к решению

◊ **Утверждение.** Если обобщенный итеративный алгоритм сходится, то получающийся результат является решением уравнений потоков данных.

- $D = Backward$**:**

  Аналогичные рассуждения, только вместо множества $Out[B]$ рассматривается множество $In[B]$.

3.5 Свойства итеративного алгоритма

3.5.2 Максимальная фиксированная точка

◊ **Определение**. *Максимальная фиксированная точка* системы уравнений

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

представляет собой решение $\{In[B_i]^{max}, Out[B_i]^{max}\}$ этой системы, обладающее тем свойством, что для любого другого решения $\{In[B_i], Out[B_i]\}$ выполняются условия

$$In\ [B_i] \le In[B_i]^{max} \text{ и } Out[B_i] \le Out[B_i]^{max}$$

где $\le$ – полурешеточное отношение частичного порядка.

3.5 Свойства итеративного алгоритма

3.5.3 Монотонность итераций

◊ **Утверждение 1.** Пусть $In[B]^i$ и $Out[B]^i$ – значения $In[B]$ и $Out[B]$ после i-ой итерации. Если структура потока данных монотонна, то

$$In[B]^{i+1} \le In[B]^i \text{ и } Out[B]^{i+1} \le Out[B]^i$$

◊ Индукция по i.

3.5 Свойства итеративного алгоритма

3.5.3 Монотонность итераций

◊ **Утверждение 1.** Пусть $In[B]^i$ и $Out[B]^i$ – значения $In[B]$ и $Out[B]$ после i-ой итерации. Если структура потока данных монотонна, то

$$In[B]^{i+1} \le In[B]^i \text{ и } Out[B]^{i+1} \le Out[B]^i$$

- Индукция по i.
- ***Основание***:

$$In[B]^1 \le In[B]^0 \quad \text{и}$$

$$Out[B]^1 \le Out[B]^0,$$

так как $\forall B \ne Entry$: $In[B]^0 = \top$ и $Out[B]^0 = \top$.

3.5 Свойства итеративного алгоритма

3.5.3 Монотонность итераций

◊ **Утверждение 1.** Пусть $In[B]^i$ и $Out[B]^i$ – значения $In[B]$ и $Out[B]$ после i-ой итерации. Если структура потока данных монотонна, то

$$In[B]^{i+1} \leq In[B]^i \text{ и } Out[B]^{i+1} \leq Out[B]^i$$

- Индукция по i.
- ***Шаг***: Пусть $In[B]^k \leq In[B]^{k-1}$ и $Out[B]^k \leq Out[B]^{k-1}$

3.5 Свойства итеративного алгоритма

3.5.3 Монотонность итераций

◊ **Утверждение 1.** Пусть $In[B]^i$ и $Out[B]^i$ – значения $In[B]$ и $Out[B]$ после i-ой итерации. Если структура потока данных монотонна, то

$$In[B]^{i+1} \le In[B]^i \text{ и } Out[B]^{i+1} \le Out[B]^i$$

- Индукция по i.
- ***Шаг***: Пусть $In[B]^k \le In[B]^{k-1}$ и $Out[B]^k \le Out[B]^{k-1}$

$$In[B]^{k+1} = \bigwedge_{P \in Pred(B)} Out[P]^k \le \bigwedge_{P \in Pred(B)} Out[P]^{k-1} = In[B]^k$$

так как операция сбора монотонна.

3.5 Свойства итеративного алгоритма

3.5.3 Монотонность итераций

◊ **Утверждение 1.** Пусть $In[B]^i$ и $Out[B]^i$ – значения $In[B]$ и $Out[B]$ после i-ой итерации. Если структура потока данных монотонна, то

$$In[B]^{i+1} \le In[B]^i \text{ и } Out[B]^{i+1} \le Out[B]^i$$

- Индукция по i.
- ***Шаг***: Пусть $In[B]^k \le In[B]^{k-1}$ и $Out[B]^k \le Out[B]^{k-1}$

$$In[B]^{k+1} = \bigwedge_{P \in Pred(B)} Out[P]^k \le \bigwedge_{P \in Pred(B)} Out[P]^{k-1} = In[B]^k$$

так как операция сбора монотонна.

$$Out[B]^{k+1} = f_B(In[B]^{k+1}) \le f_B(In[B]^k) = Out[B]^k,$$

так как передаточная функция $f_B(x)$ монотонна.

3.5 Свойства итеративного алгоритма

3.5.3 Монотонность итераций

◊ **Утверждение 1.** Пусть $In[B]^i$ и $Out[B]^i$ – значения $In[B]$ и $Out[B]$ после i-ой итерации. Если структура потока данных монотонна, то

$$In[B]^{i+1} \le In[B]^i \text{ и } Out[B]^{i+1} \le Out[B]^i$$

◊ **Следствие**. Для любого i

$$In[B] \le In[B]^i$$

$$Out[B] \le Out[B]^i$$

3.5 Свойства итеративного алгоритма

3.5.3 Монотонность итераций

◊ **Утверждение 2.** Если структура потока данных монотонна, то решение системы уравнений (1), найденное с помощью итеративного алгоритма, является максимальной фиксированной точкой этой системы.

◊ Требуется доказать, что для всех j

$$In[B_j] \le In[B_j]^{IA},\ Out[B_j] \le Out[B_j]^{IA},$$

где $\{In[B_j]^{IA}, Out[B_j]^{IA}\}$ – решение системы (1), найденное с помощью итеративного алгоритма, $\{In[B_j], Out[B_j]\}$ – любое другое решение этой системы.

Аналогично доказательству предыдущего утверждения.

3.5 Свойства итеративного алгоритма

3.5.4. Сходимость итеративного алгоритма

◊ **Определение 1.** *Восходящей цепочкой* в частично упорядоченном множестве $(L, \leq)$ называется последовательность его элементов, в которой

$$x_1 < x_2 < \ldots < x_n.$$

◊ **Определение 2.** *Высотой* полурешетки называется наибольшее количество отношений $<$ в восходящих цепочках.

◊ **Утверждение.** Если полурешетка структуры монотонна и имеет конечную высоту, то итеративный алгоритм гарантированно сходится после количества итераций, не превышающего произведения высоты полурешетки на количество базовых блоков.

3.5 Свойства итеративного алгоритма

3.5.5 Решение, получаемое итеративным алгоритмом (максимальная фиксированная точка)

◊ Итеративный алгоритм

(1) посещает базовые блоки не в порядке их выполнения, а в порядке обхода ГПУ (на каждой итерации каждый узел посещается только один раз)

(2) в каждой точке сбора применяет операцию сбора к значениям потока данных, полученным к этому моменту

(3) иногда в пределах итерации базовый блок B посещается до посещения его предшественников (прямой обход)

3.5 Свойства итеративного алгоритма

3.5.5 Решение, получаемое итеративным алгоритмом (максимальная фиксированная точка)

◊ (4) для процесса итерации *необходимо граничное условие*, так как к блоку $Entry$ передаточная функция неприменима.

(5) в качестве «нулевой итерации» все $Out[B]$ инициализируются значением Т, которое, по определению, «не меньше» всех значений потока, и, следовательно, того значения, которое оно заменяет; при этом монотонность передаточных функций обеспечивает получение результата, «не меньшего», чем искомое решение:

3.5 Смысл решения уравнений потока данных

Без потери общности будем считать, что рассматривается прямая задача (обход графа потока от *Entry* к *Exit*).
Обратная задача (обход графа потока от *Exit* к *Entry*) рассматривается аналогично.

3.5.1 Идеальное решение

◊ Пусть f_{B_k} – передаточная функция блока B_k в графе потока.

Рассмотрим путь $P = Entry \to B_1 \to B_2 \to \ldots \to B_{k\text{-}1} \to B_k$

(Путь P может содержать циклы: базовые блоки могут встречаться в нем по нескольку раз).

По определению *передаточная функция пути P*:

$$f_P = f_{B_1} \circ f_{B_2} \circ \ldots \circ f_{B_{k-1}}$$

(f_{B_k} не является частью композиции, так как путь достигает начала блока B_k, но не его конца). Значение потока данных, создаваемое этим путем, представляет собой $f_P(l_{Entry})$, где l_{Entry} – граничное условие.

3.5 Смысл решения уравнений потока данных

3.5.1 Идеальное решение

◊ Пусть $\mathcal{P} = \{P_1, P_2, \ldots\}$ – множество *всех выполнимых* путей от $Entry$ до B_k

Путь P является *выполнимым* только тогда, когда известно выполнение программы (начальные данные), которое следует в точности по этому пути.

◊ Идеальным решением системы уравнений потока данных для $In[B_k]$ будет:

$$Ideal[B_k] = \bigwedge\nolimits_{P \in \mathcal{P}} f_P(l_{Entry}).$$

◊ Решение $Ideal[B_k]$ названо идеальным, так как

(1) оно наиболее точное и

(2) вычислить его почти никогда не удается, так как поиск всех возможных путей выполнения – задача неразрешимая. Следовательно, требуется поиск приближенного решения.

3.5 Смысл решения уравнений потока данных

3.5.2 Свойства решений уравнений потока данных для монотонных и дистрибутивных структур

◊ Добавление еще одного пути в сбор $\bigwedge_{P \in \mathcal{P}}$ делает решение «меньше» в смысле частичного порядка полурешетки $\leq$

◊ Если просмотрены все выполнимые пути и ни одного лишнего, получается идеальное решение $Ideal$.

◊ Решение Sol_1: $Ideal \leq Sol_1$, получается, когда *просмотрены не все выполнимые пути*. Использовать решение Sol_1 опасно, так как для непросмотренных путей преобразования программы могут оказаться неверными (нарушается консервативность).

3.5 Смысл решения уравнений потока данных

3.5.2 Свойства решений уравнений потока данных для монотонных и дистрибутивных структур

◊ Решение Sol_2: $Sol_2 \leq Ideal$ обладает следующими свойствами:

(1) Sol_2 консервативно: оно содержит все выполнимые пути

(2) Sol_2 неточно: в нем не отсеяны «лишние» пути, т.е. либо не существующие в графе потока, либо существующие, но такие, по которым программа никогда не проследует.

В результате:

(1) Sol_2 может запрещать некоторые из преобразований, разрешенных решением *Ideal.*

(2) все преобразования, которые разрешает $Sol2$, корректны

◊ В абстракции потока данных предполагается, что каждый путь в графе потока может быть пройден.

3.5 Смысл решения уравнений потока данных

3.5.3 Решение сбором по всем путям

◊ *Решение сбором по всем путям* (*MOP-решение*) от $Entry$ до входа в B_k определяется соотношением:

$$MOP[B_k] = \bigwedge_{P \in Q} f_P(l_{Entry}),$$

где $Q = \{P_1, P_2, \ldots\}$ – множество *всех* путей от $Entry$ до входа в блок B_k

◊ Пути, рассматриваемые в *MOP*-решении, – это надмножество всех выполнимых путей: *MOP*-решение собирает значения потоков данных как для всех выполнимых путей, так и для путей, которые не могут быть выполнены. Следовательно, для всех B_k выполняется соотношение

$$MOP[B_k] \leq Ideal[B_k].$$

3.5 Смысл решения уравнений потока данных

3.5.3 Решение сбором по всем путям

◊ **Замечание**. Для прямой задачи $MOP[B_k]$ дает значения для $In[B_k]$. Для обратной задачи $MOP[B_k]$ дает значения для $Out[B_k]$. Если рассматривается обратная задача, *MOP*-решение определяется соотношением :

$$MOP[B_k] = \bigwedge_{P \in Q} f_P(l_{Exit}),$$

где $Q = \{P_1, P_2, \ldots\}$ – множество *всех* путей от выхода из B_k до $Exit$.

◊ Если в анализируемой программе есть циклы, количество всех путей, рассматриваемых при построении *MOP*-решения, становится бесконечным. Поэтому построить *MOP*-решение, как правило, не удается (невозможно учесть бесконечное множество путей).

3.5 Смысл решения уравнений потока данных

3.5.4 Решение, получаемое итеративным алгоритмом (максимальная фиксированная точка)

◊ Итеративный алгоритм

(1) посещает базовые блоки не в порядке их выполнения (как при вычислении *MOP*-решения), а в порядке обхода графа потока (на каждой итерации каждый узел посещается только один раз)

(2) в каждой точке слияния алгоритм применяет операцию сбора к значениям потока данных, полученным к этому моменту

(3) иногда в пределах итерации базовый блок B посещается до посещения его предшественников (прямой обход)

3.5 Смысл решения уравнений потока данных

3.5.4 Решение, получаемое итеративным алгоритмом (максимальная фиксированная точка)

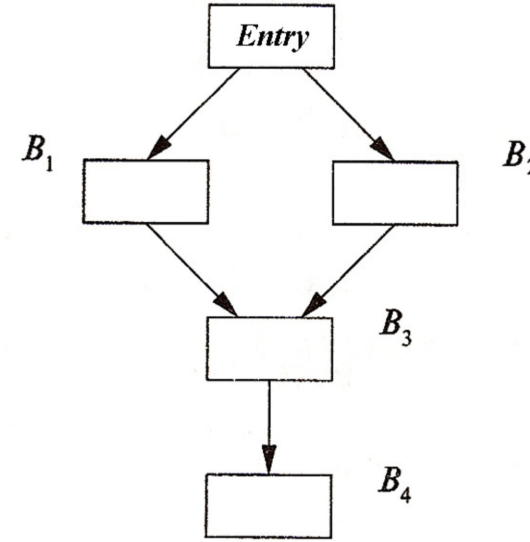
- Следовательно:
  - *необходимо граничное условие*, так как к блоку $Entry$ передаточная функция не применяется.
  - необходима инициализация потока данных для выходов из базовых блоков: $Out[B]$ инициализируется значением Т, которое, как известно, «не меньше» всех значений потока, и, следовательно, того значения, которое оно заменяет.
  - при использовании Т в качестве входных данных монотонность передаточных функций обеспечивает получение результата, «не меньшего», чем искомое решение:

$$MFP \leq MOP$$

3.5 Смысл решения уравнений потока данных

3.5.4 Решение, получаемое итеративным алгоритмом (максимальная фиксированная точка)

◊ **Пример** (см. рисунок)



Требуется вычислить значение $In[B_4]$.

(1) *MOP*-решение:

$$MOP[B_4] =$$

$$= ((f_{B_3} \circ f_{B_1}) \wedge (f_{B_3} \circ f_{B_2}))(l_{Entry})$$

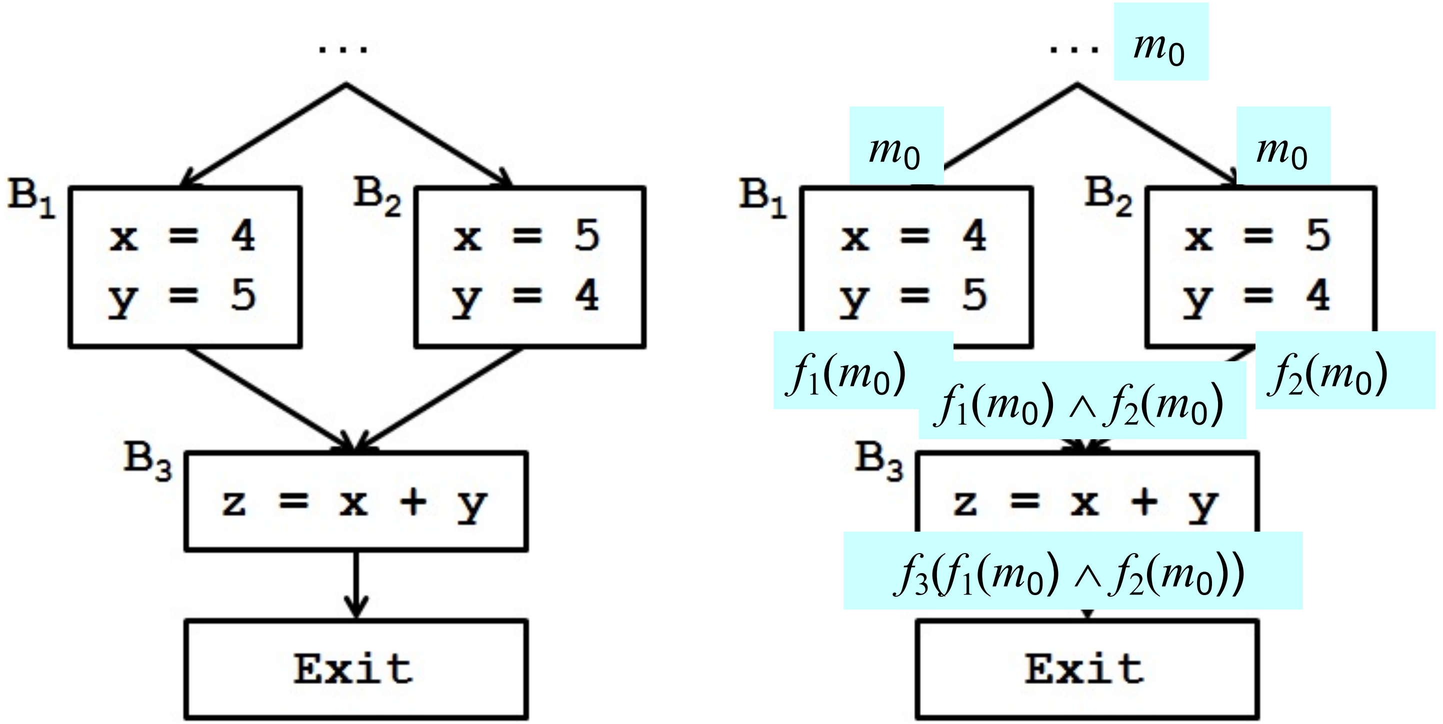
(2) Итеративный алгоритм (узлы посещаются в порядке B_1, B_2, B_3, B_4)

$$In[B_4] = f_{B_3}(f_{B_1}(l_{Entry}) \wedge f_{B_2}(l_{Entry}))$$

Операция сбора $\wedge$ в итеративном алгоритме применяется раньше, чем при вычислении *MOP*-решения

Пример на распространение констант (тема одной из следующих лекций)

Пусть f_1, f_2 и f_3 – передаточные функции блоков B_1, B_2 и B_3 соответственно, а m_0 – состояние на входе в блоки B_1 и B_2. Тогда



3.5 Смысл решения уравнений потока данных

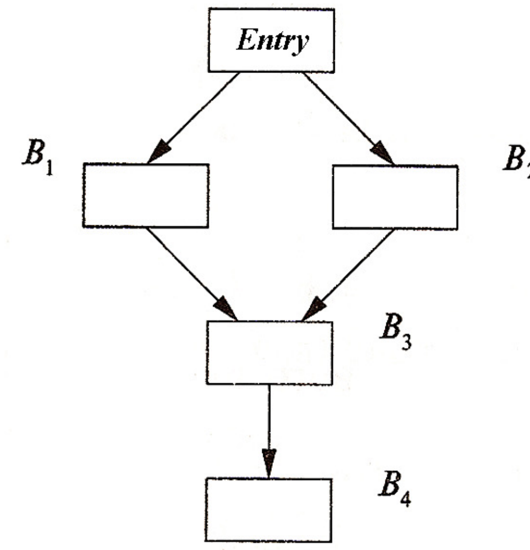
3.5.4 Решение, получаемое итеративным алгоритмом (максимальная фиксированная точка)

◊ Если структура потока данных дистрибутивна, то

$$In[B_4] = MOP[B_4]$$

если же структура потока данных монотонна, но не дистрибутивна, то

$$In[B_4] \leq MOP[B_4]$$



◊ Следовательно, в обоих случаях приближенное решение, представленное максимальной фиксированной точкой (*MFP*-решение), удовлетворяет условию:

$$MFP[B] \leq MOP[B]$$

3.5 Смысл решения уравнений потока данных

3.5.4 Решение, получаемое итеративным алгоритмом (максимальная фиксированная точка)

◊ Если структура потока данных дистрибутивна, то для всех B

$$MFP[B] = MOP[B].$$

◊ В разделах 3.2.5 и 3.2.6 была установлена дистрибутивность структур достигающих определений (RD), живых переменных (LV) и доступных выражений (AE).

Следовательно, в указанных случаях итеративный алгоритм позволяет получить MOP-решение.

3.5 Смысл решения уравнений потока данных

3.5.5. Консервативность *MFP*-решения

◊ **Утверждение.** *MFP*-решение, получаемое итеративным алгоритмом, всегда консервативно

Доказательство

Индукция по i: значения, полученные итеративным алгоритмом после i итераций, не превосходят результата операции сбора по всем путям длины i.

Итеративный алгоритм завершается только тогда, когда он получает такой же ответ, как и при неограниченном количестве итераций.

Следовательно, $MFP \le MOP$.

Но уже установлено: $MOP \le Ideal$.

Следовательно (транзитивность $\le$), $MFP \le Ideal$
и MFP-решение консервативно.