

7. Распространение констант

7.1 Глобальное распространение констант

7.1.1 Постановка задачи

- ◇ **Распространение констант** — это оптимизирующее преобразование, заменяющее **выражения**, которые при выполнении всякий раз вычисляют одну и ту же константу, самой этой **константой**.
- ◇ Преобразование удобно выполнять над процедурой в SSA-форме.

7.1 Глобальное распространение констант

7.1.1 Постановка задачи

- ◇ **Распространение констант** – это оптимизирующее преобразование, заменяющее **выражения**, которые при выполнении всякий раз вычисляют одну и ту же константу, самой этой **константой**.
- ◇ Преобразование удобно выполнять над процедурой (программой) в SSA-форме.
- ◇ Распространение констант является прямой задачей потока данных.

7.1 Глобальное распространение констант

7.1.1 Постановка задачи

- ◇ **Распространение констант** – это оптимизирующее преобразование, заменяющее **выражения**, которые при выполнении всякий раз вычисляют одну и ту же константу, самой этой **константой**.
- ◇ Преобразование удобно выполнять над процедурой (или файлом, содержащим несколько процедур) в *SSA*-форме.
- ◇ Распространение констант является прямой задачей потока данных.
- ◇ Выполняя глобальный анализ потока данных (методом итераций) рассматриваем в процедуре все *SSA*-имена v и каждому из них *сопоставляем константу*; это может быть:
 - ◇ специальная константа *Undef* – значение переменной не определено;
 - ◇ специальная константа *NAC* (*not-a-constant*) – переменная не может быть константой;
 - ◇ одна из множества рассматриваемых констант.

7.1 Глобальное распространение констант

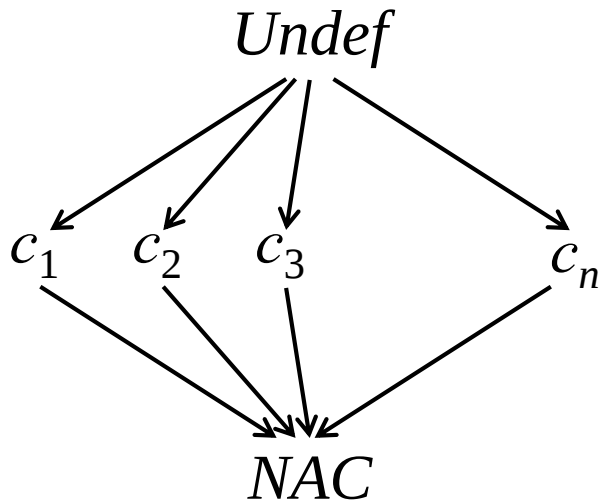
7.1.2 Константы *NAC* и *Undef*

- ◇ ***SSA*-имени v сопоставляется константа *NAC* если установлено, что:**
 - ◇ v может быть присвоено входное значение, т.е.
 - ◆ v — имя формального параметра процедуры или
 - ◆ v — имя целевой переменной функции ввода;
 - ◇ при вычислении v используется *NAC*
 - ◇ на различных путях вычисления v могут быть присвоены разные значения.
- ◇ ***SSA*-имени v сопоставляется константа *Undef*, если не найдено ни одного определения v , достигающего рассматриваемой точки. На самом деле всем *SSA*-именам сначала сопоставляется константа *Undef*.**

7.1 Глобальное распространение констант

7.1.2 Полурешетка значений констант

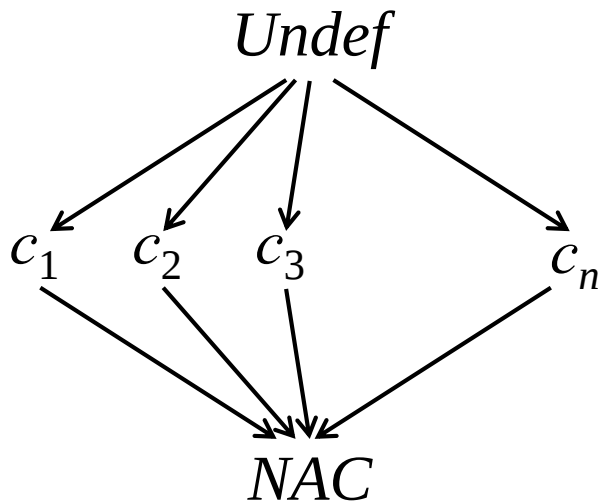
- ◇ Множество значений, сопоставляемых каждой переменной, (включая специальные значения *Undef* и *NAC*) можно представить как полурешетку (см. рисунок).



7.1 Глобальное распространение констант

7.1.2 Полурешетка значений констант

- ◇ Множество значений, сопоставляемых каждой переменной, (включая специальные значения *Undef* и *NAC*) можно представить как полурешетку (см. рисунок).

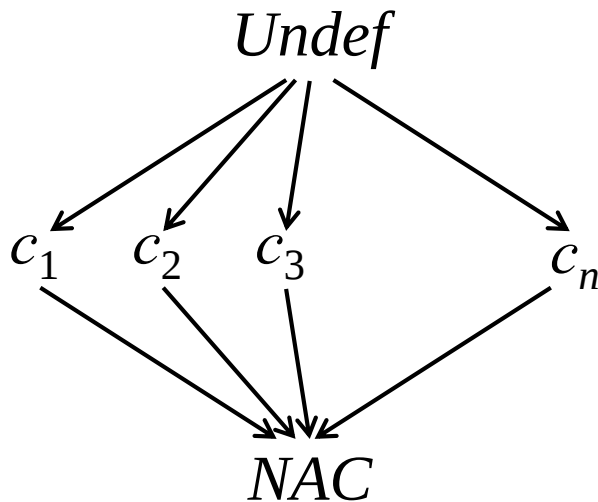


- ◇ верхний элемент $\top = Undef$
- ◇ нижний элемент $\perp = NAC$
- ◇ Значения констант (c_i) не сравнимы между собой, но все они «меньше» *Undef* и «больше» *NAC*:
 $\forall i \ NAC \leq c_i \leq Undef$

7.1 Глобальное распространение констант

7.1.2 Полурешетка значений констант

- ◇ Множество значений, сопоставляемых каждой переменной, (включая специальные значения *Undef* и *NAC*) можно представить как полурешетку (см. рисунок).



- ◇ для всех значений v :

$$Undef \wedge v = v$$

$$NAC \wedge v = NAC$$

- ◇ в частности

$$NAC \wedge Undef = NAC$$

- ◇ по определению

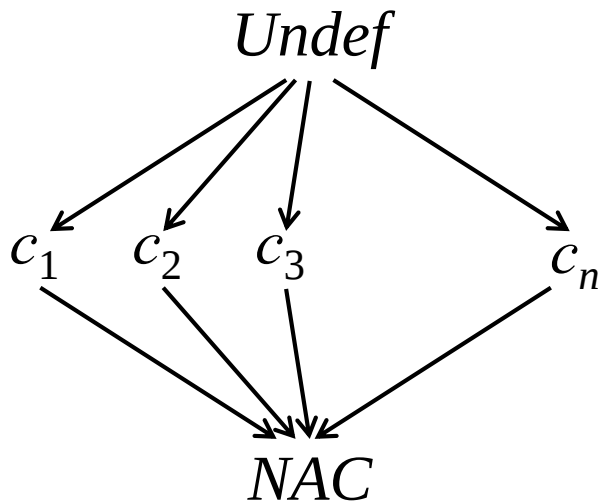
для любых двух констант c_1 и c_2

$$c_1 \wedge c_2 = \begin{cases} c_1 & \text{если } c_1 = c_2 \\ NAC & \text{если } c_1 \neq c_2 \end{cases}$$

7.1 Глобальное распространение констант

7.1.2 Полурешетка значений констант

- ◇ Множество значений, сопоставляемых каждой переменной, (включая специальные значения *Undef* и *NAC*) можно представить как полурешетку (см. рисунок).



- ◇ для всех значений v :

$$Undef \wedge v = v$$

$$NAC \wedge v = NAC$$

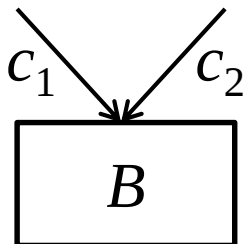
- ◇ в частности

$$NAC \wedge Undef = NAC$$

- ◇ по определению

для любых двух констант c_1 и c_2

$$c_1 \wedge c_2 = \begin{cases} c_1 & \text{если } c_1 = c_2 \\ NAC & \text{если } c_1 \neq c_2 \end{cases}$$



7.1 Глобальное распространение констант

7.1.3 Операция сбора в структуре распространения констант

- ◇ Значение потока данных в структуре распространения констант представляет собой отображение

$$v: \mathcal{N} \rightarrow C,$$

где $\mathcal{N}$ – множество *SSA*-имен

C – полурешетка констант

Значение, соответствующее *SSA*-имени n в отображении v , обозначается как $v(n)$

- ◇ Полурешетка значений потока данных представляет собой декартово произведение k полурешеток C , где k – количество переменных в анализируемой процедуре:

$$v \wedge v' = v'' \quad \Leftrightarrow \quad \forall n: v''(n) = v(n) \wedge v'(n).$$

$$v \leq v' \quad \Leftrightarrow \quad \forall n: v(n) \leq v'(n)$$

7.1 Глобальное распространение констант

7.1.4 Передаточные функции структуры распространения констант

- ◇ Множество передаточных функций F_{Const} состоит из функций вида

$$f: \mathcal{V} \rightarrow \mathcal{V},$$

где $\mathcal{V}$ – множество отображений $v: \mathcal{N} \rightarrow C$.

- ◇ F_{Const} содержит тождественную функцию i :

$$\forall v \in \mathcal{V} \ i(v) = v$$

- ◇ F_{Const} содержит константную передаточную функцию c :

$$\forall v \in \mathcal{V} \ c(v) = v_0$$

где

$$\forall v \in \mathcal{V} \ v_0(v) = Undefined$$

В частности, $c(Entry) = v_0$ означает, что перед началом выполнения программы не определена ни одна переменная.

7.1 Глобальное распространение констант

7.1.4 Передаточные функции структуры распространения констант

- ◇ Пусть $f_s \in \mathcal{F}_{Const}$ – передаточная функция для инструкции s и пусть v и $v' \in \mathcal{V}$ – значения потока данных в точках до и после s : $v' = f_s(v)$.
- ◇ Опишем f_s через v и v' .
 - (1) Если s – не содержит присваивания какому-либо SSA -имени, то
 f_s – тождественная функция i , т.е. для всех v $v'(n) = v(n)$
 - (2) Если s – содержит присваивание SSA -имени x , то
для $n \neq x$ $v'(n) = v(n)$,
для $n = x$ $v'(n) = v'(x)$

7.1 Глобальное распространение констант

7.1.4 Передаточные функции структуры распространения констант

◇ Описание f_s через m и m'

(2) Пусть s – присваивание переменной x и пусть $v = x$
 $m' = m'(x)$ определяется следующим образом:

(a) Если s – присваивание константы c : $x = c$, то
 $m'(x) = c$

(b) Если s – присваивание выражения: $x = y + z$, то

$$m'(x) = \begin{cases} m(y) + m(z), & \text{если } m(y) \text{ и } m(z) \text{ константы} \\ NAC, & \text{если } m(y) = NAC \text{ или } m(z) = NAC \\ Undef & \text{в остальных случаях} \end{cases}$$

(c) Если s – присваивание любого другого выражения (например, вызова функции или указателя), то
 $m'(x) = NAC$.

7.1 Глобальное распространение констант

7.1.5 Монотонность структуры распространения констант

◇ **Утверждение.** Структура распространения констант **МОНОТОННА**:

если для m_1, m_2, m'_1, m'_2 выполняются соотношения:

$$m_1 \leq m_2, \quad m'_1 = f_s(m_1) \quad m'_2 = f_s(m_2)$$

то $m'_1 \leq m'_2$

◇ **Доказательство.** Рассмотрим определение f_s через m и m' .

Случай 1): $f_s \equiv i \Rightarrow m'_1 = m_1, m'_2 = m_2 \Rightarrow m'_1 \leq m'_2$

Случай 2а): $\forall m: f_s(m) = c \Rightarrow m'_1 = m'_2 = c \Rightarrow m'_1 \leq m'_2$

Случай 2b): на следующем слайде

Случай 2с): $\forall m: f_s(m) = NAC \Rightarrow m'_1 = m'_2 = NAC \Rightarrow m'_1 \leq m'_2$

7.1 Глобальное распространение констант

7.1.5 Монотонность структуры распространения констант

◇ Доказательство утверждения (случай 2b)

Для присваивания $x = y + z$ передаточная функция f_s описывается формулой

$$m'(x) = \begin{cases} m(y) + m(z), & \text{если } m(y) \text{ и } m(z) \text{ константы} \\ NAC, & \text{если } m(y) = NAC \text{ или } m(z) = NAC \\ Undef & \text{в остальных случаях} \end{cases}$$

которой соответствует таблица справа.

Из таблицы видно, что во всех случаях

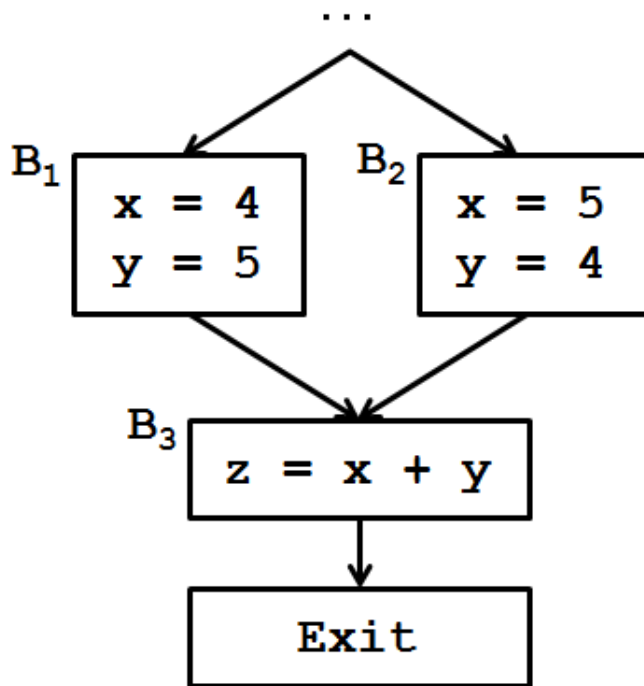
$$m'(x) \leq m(x)$$

$m(y)$	$m(z)$	$m'(x)$
$Undef$	$Undef$	$Undef$
	c_2	$Undef$
	NAC	NAC
c_1	$Undef$	$Undef$
	c_2	$c_1 + c_2$
	NAC	NAC
NAC	$Undef$	NAC
	c_2	NAC
	NAC	NAC

7.1 Глобальное распространение констант

7.1.6 Недистрибутивность структуры распространения констант

◇ Для установления отсутствия дистрибутивности у передаточной функции структуры распространения констант рассмотрим пример



Пример.

Итеративный алгоритм не может обнаружить, что в программе на рисунке слева значение $x + y$ на входе в блок B_3 всегда равно 9

И x , и y на входе в B_3 имеют значение *NAC*

Этот результат безопасен (консервативен), но неточен: не удастся отследить зависимость: когда x равно 4, y равно 5, а когда x равно 5, y равно 4.

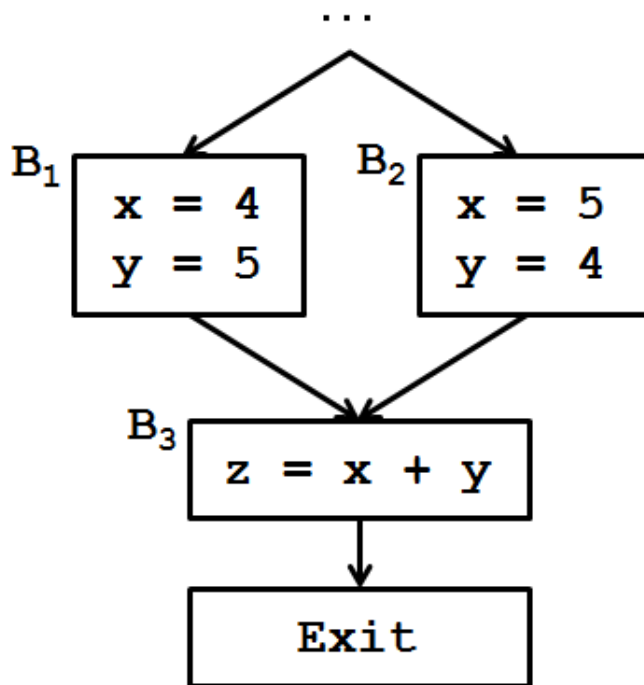
7.1 Глобальное распространение констант

7.1.6 Недистрибутивность структуры распространения констант

- ◇ Таким образом, структура распространения констант не дистрибутивна, т.е. *MFP*-решение безопасно, но необязательно равно *MOP*-решению и может оказаться «меньше» (грубее) него.

Пример.

Итеративный алгоритм не сможет обнаружить, что в программе на рисунке слева значение $\mathbf{x} + \mathbf{y}$ на входе в блок B_3 всегда равно **9**, так как и $\mathbf{x}$, и $\mathbf{y}$ на входе в B_3 имеют значение **НАС**



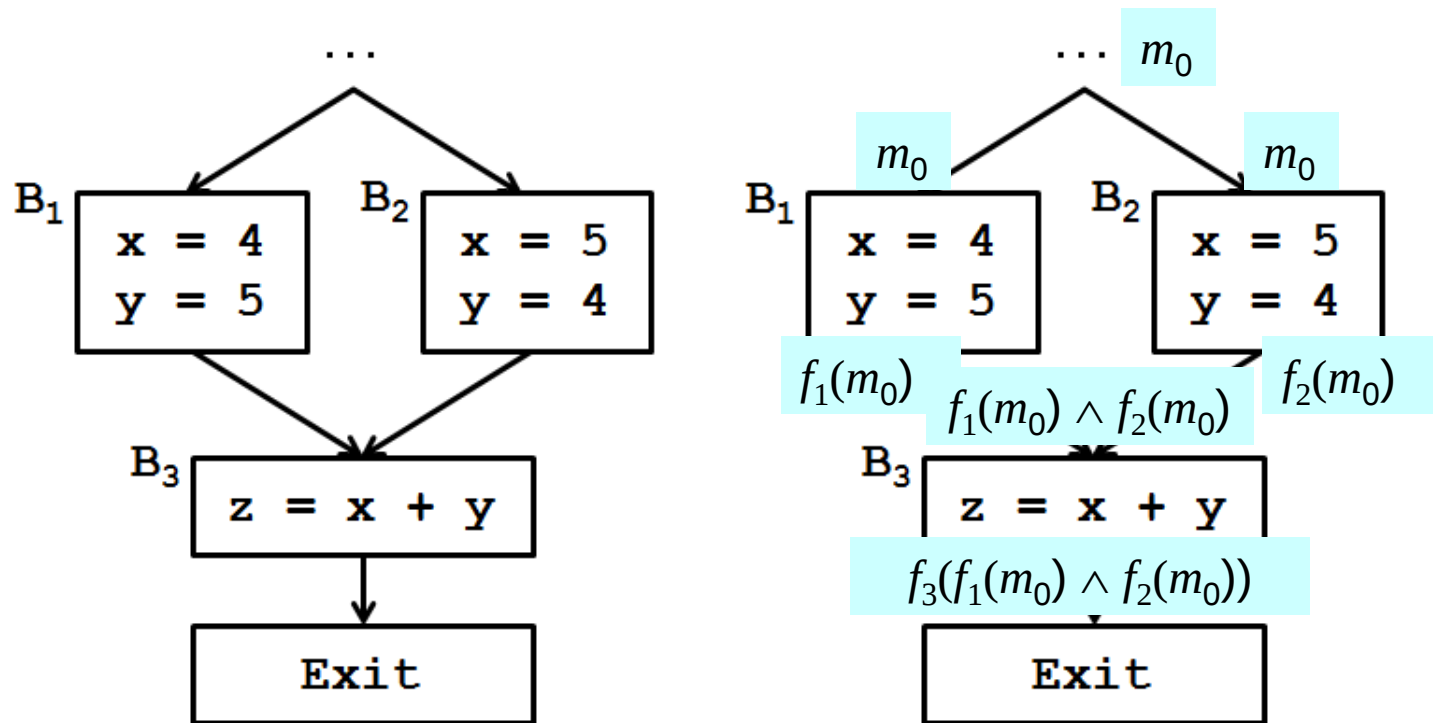
Этот результат безопасен (консервативен), но неточен: из-за недистрибутивности структуры распространения констант итеративный алгоритм не может отследить зависимость: когда $\mathbf{x}$ равно 4, $\mathbf{y}$ равно 5, а когда $\mathbf{x}$ равно 5, $\mathbf{y}$ равно 4.

7.1 Глобальное распространение констант

7.1.6 Недистрибутивность структуры распространения констант

◇ Окончание примера.

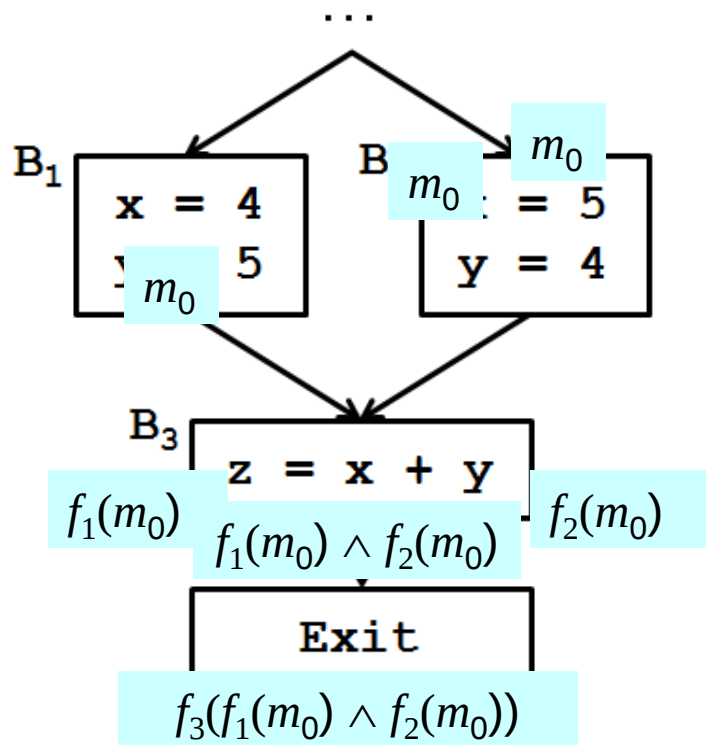
Пусть f_1 , f_2 и f_3 – передаточные функции блоков B_1 , B_2 и B_3 соответственно, а m_0 – состояние на входе в блоки B_1 и B_2 . Тогда



7.1 Глобальное распространение констант

7.1.6 Недистрибутивность структуры распространения констант

◇ Пример. Таким образом, как видно из таблицы



m	$m(x)$	$m(y)$	$m(z)$
m_0	<i>Undef</i>	<i>Undef</i>	<i>Undef</i>
$f_1(m_0)$	4	5	<i>Undef</i>
$f_2(m_0)$	5	4	<i>Undef</i>
$f_1(m_0) \wedge f_2(m_0)$	<i>NAC</i>	<i>NAC</i>	<i>Undef</i>
$f_3(f_1(m_0) \wedge f_2(m_0))$	<i>NAC</i>	<i>NAC</i>	<i>NAC</i>
$f_3(f_1(m_0))$	4	5	9
$f_3(f_2(m_0))$	5	4	9
$f_3(f_1(m_0)) \wedge f_3(f_2(m_0))$	<i>NAC</i>	<i>NAC</i>	9

$f_3(f_1(m_0) \wedge f_2(m_0)) = \text{NAC}$, а $f_3(f_1(m_0)) \wedge f_3(f_2(m_0)) = 9$,

откуда следует **недистрибутивность** структуры:

$$f_3(f_1(m_0) \wedge f_2(m_0)) \neq f_3(f_1(m_0)) \wedge f_3(f_2(m_0))$$

7.1 Глобальное распространение констант

7.1.7 Интерпретация значения *Undef*

- ◇ Значение *Undef* используется в итеративном алгоритме:
 - (1) для инициализации входного узла (граничное условие).
 - (2) для инициализации внутренних точек программы на нулевой итерации
- ◇ Смысл случаев (1) и (2):
 - ◇ **Случай (1):** в начале выполнения программы значения ее переменных не определены.
В конце итеративного процесса переменные на выходе из *Entry* сохраняют значение *Undef*, поскольку *Out[Entry]* не изменяется.

7.1 Глобальное распространение констант

7.1.7 Интерпретация значения *Undef*

◇ Смысл случаев (1) и (2):

◇ **Случай (2):** из-за недостатка информации в начале итерационного процесса решение аппроксимируется верхним элементом *Undef*.

В конце итерационного процесса эти *Undef* сохранятся только в тех точках, для которых не найдется определений соответствующих переменных ни на одном пути, ведущем к этим точкам.

Если же найдется хотя бы один путь, содержащий определение переменной, достигающее рассматриваемой точки программы, то эта переменная не будет иметь значение *Undef*.

7.1 Глобальное распространение констант

7.1.7 Интерпретация значения *Undef*

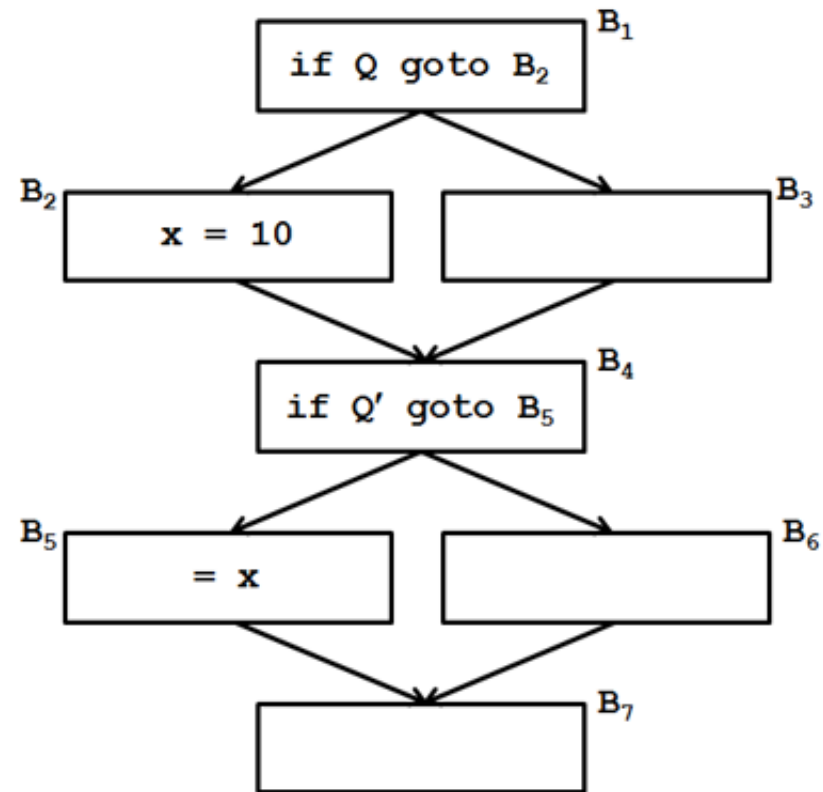
- ◇ Если все определения какой-либо переменной, достигающие некоторой точки программы, имеют одно и то же константное значение (но не *Undef* и не *NAC*), то эта переменная рассматривается как константа, **даже если может оказаться, что она не определена на одном из путей.**
- ◇ Если предположить, что оптимизируемая программа корректна, то итеративный алгоритм может найти большее количество констант, т.к. он может считать константами и те переменные, которые на части путей имеют значения *Undef*.
Такой подход правомерен для языков программирования, которые исходят из того, что неопределенная (*Undef*) переменная может иметь любое значение.
Если семантика языка программирования требует, чтобы все неопределенные переменные принимали некоторое конкретное значение, необходимо это ограничение включить в формулировку задачи анализа потока данных.

7.1 Глобальное распространение констант

7.1.7 Интерпретация значения *Undef*

◇ На рисунке справа значения x на выходе из блоков B_2 и B_3 – соответственно 10 и *Undef*. Из $Undef \wedge 10 = 10$ следует, что значение x на входе в B_4 равно 10, и в B_5 можно заменить x на 10.

◇ Если выполнение пойдет по пути $B_1 \rightarrow B_3 \rightarrow B_4 \rightarrow B_5$, то блока B_5 будет достигать значение $x = Undef$, и замена x на 10 будет корректной не для всех исходных языков.



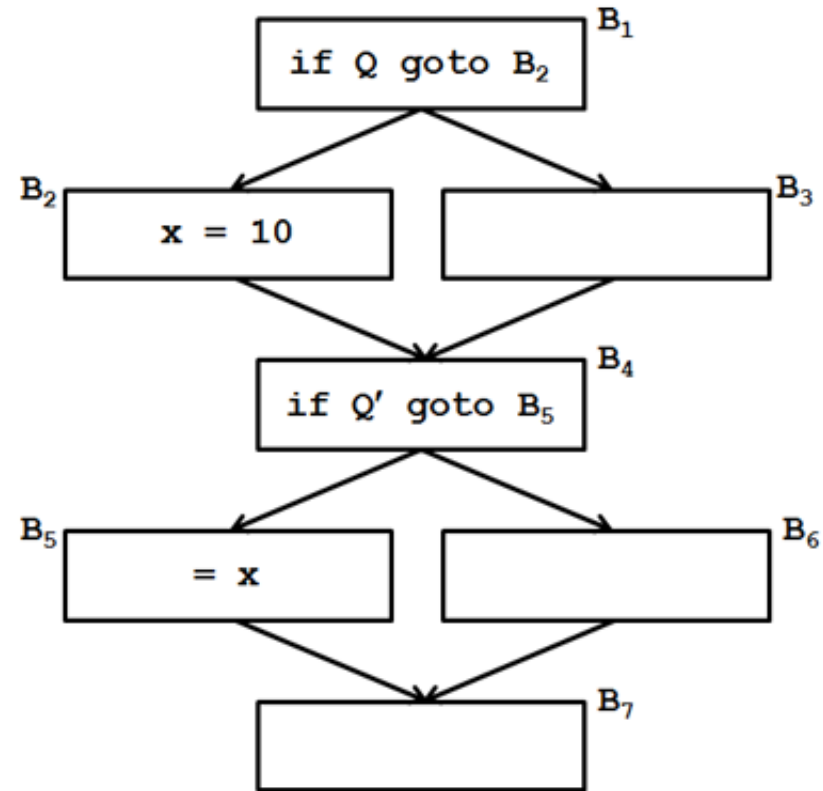
7.1 Глобальное распространение констант

7.1.7 Интерпретация значения *Undef*

Пример.

- ◇ Даже если можно доказать, что предикат Q не может иметь значение *false*, когда предикат Q' имеет значение *true*, и поэтому путь $B_1 \rightarrow B_3 \rightarrow B_4 \rightarrow B_5$ никогда не будет пройден, следует понимать, что **такой вывод находится за пределами возможностей анализа потоков данных.**

- ◇ Таким образом, если считать, что исходная программа корректна и что все ее переменные определяются до их использования, то корректно считать, что x при входе в B_5 определена, и ее значение равно 10.



7.2 Алгоритм глобального распространения констант

7.2.1 Фаза инициализации

// Initialization Phase

WorkList = $\emptyset$;

for each SSA-name **v**

initialize **Value(v)** by rules specified in the text;

if **Value(v)** $\neq$ **Undef** then

WorkList = **WorkList** $\cup$ {**v**};

На фазе инициализации устанавливаются начальные значения *SSA*-имен и формируется начальное состояние **WorkList**.

Для каждого *SSA*-имени **v** анализируется операция, определяющая **v** и устанавливающая значение **Value(v)** в соответствии со следующими простыми правилами (следующий слайд).

7.2 Алгоритм глобального распространения констант

7.2.1 Фаза инициализации

// Initialization Phase

WorkList = $\emptyset$;

for each SSA-name v

initialize Value(v) by rules specified in the text;

if Value(v) $\neq$ Undef then

WorkList WorkList $\cup$ { v };

1. v определяется ϕ -функцией, Value(v) = Undef.
2. v равно одной из констант c_i , Value(v) = c_i .
3. v может быть присвоено входное значение, Value(v) = NAC.
4. Значение v неизвестно, Value(v) = Undef.

Если Value(v) $\neq$ Undef, алгоритм добавляет v в WorkList

7.2 Алгоритм глобального распространения констант

7.2.2 Фаза распространения

```
// Propagation Phase
// Выполнять итерации (до неподвижной точки)
while (WorkList  $\neq \emptyset$ ;)
  remove some v from WorkList      //Выбрать
                                    произвольное имя
  for each operation op that uses v
    let w be the SSA-name that op defines
    if Value(w)  $\neq$  NAC then        // Recompute and
                                    test for change
      t  $\leftarrow$  Value(w)
      Value(w)  $\leftarrow$  result of interpreting op over
                        lattice values
      if Value(w)  $\neq$  t
        then WorkList  $\leftarrow$  WorkList  $\cup$  {w};
```

7.2 Алгоритм глобального распространения констант

7.2.2 Фаза распространения

Фаза распространения проста:

Из **WorkList** извлекается (с удалением) произвольное SSA-имя **v**.

Алгоритм анализирует каждую операцию **op**,

которая использует **v**

и определяет SSA -имя **w**.

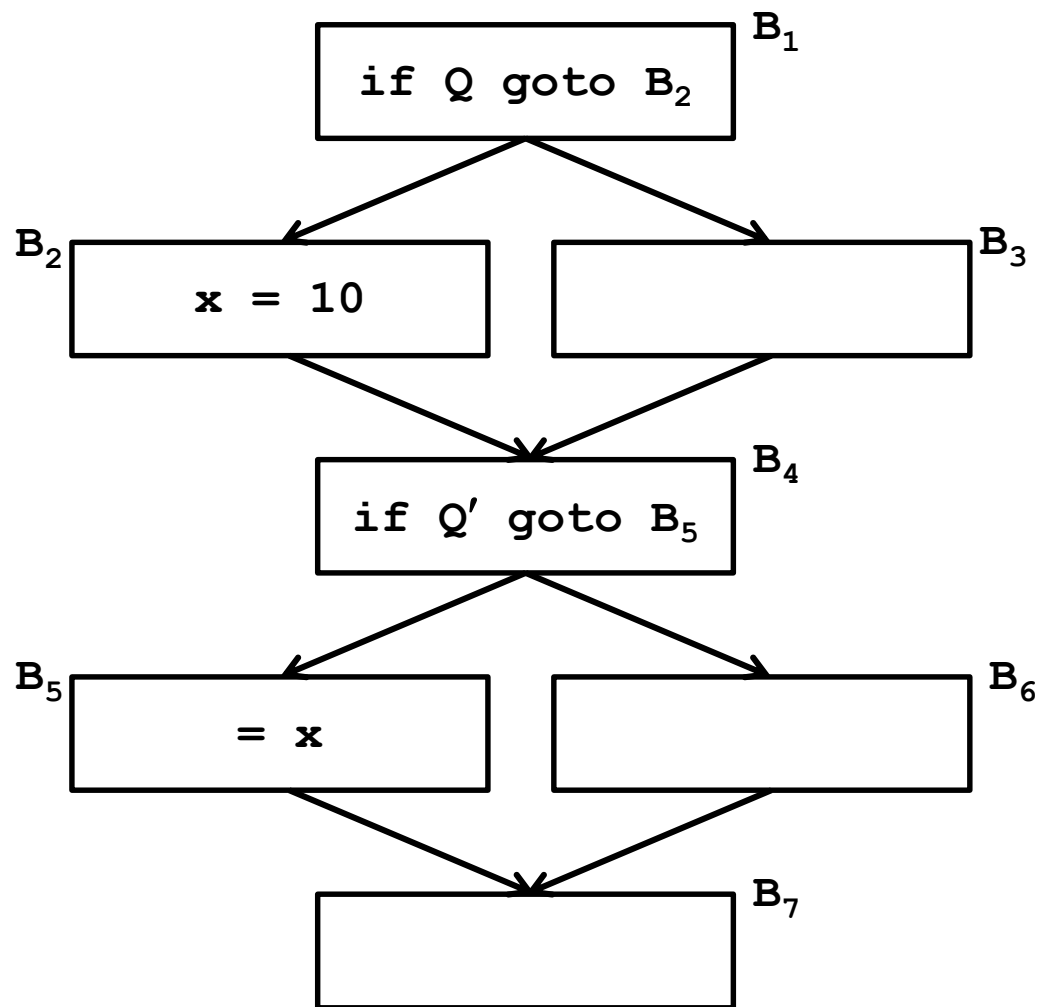
Если **Value (w) = NAC**, дальнейший анализ **w** не имеет смысла.

В противном случае, моделируется выполнение **op** с помощью интерпретации операции над полурешетками значений ее операндов.

Если результат отличен от **Value (w)**, он рассматривается как новое значение **Value (w)**, причем **w** добавляется в **WorkList**.

Алгоритм завершается, когда **WorkList** пуст.

7.1 Глобальное распространение констант



7.1 Глобальное распространение констант

7.1.6 Недистрибутивность структуры распространения констант

