Максим Исаев

Темы 6–10. Теоретическое задание

Задание (1): Предложите расширение алгоритма каскадного анализа общих подвыражний:

1. Расширьте алгоритм поддержкой инструкций копирования и коммутативных операций. Для коммутативных операций, записи в таблицы рекомендуется хранить в каноническом виде (например, аргументы должны быть упорядочены каким-то образом). Приведите псевдокод расширенной версии алгоритма.
2. Примените расширенный алгоритм к следующему блоку:

```
c = a - b
d = a × b
e = a
h = a - b
a = d
f = b × e
g = e - b
d = a + d
a = b
u = e - a
v = c × h
c = e - a
u = c × u
h = e - a
c = g × v
v = d + f
f = h × u
a = v + c
b = a × f
```

(a) Выпишите таблицу значений
(b) Нарисуйте граф, соответствующий таблице

Решение:

1.

```python
T = empty
N = 0

for 'a = rhs' in the block:
    if rhs is variable: # handle copy instruction
        if rhs in T:
            T[a] = T[rhs]
        else:
            N = N + 1
            T[a] = N
        continue

    b, op, c = rhs.parse()
    if b in T:
        nb = T[b]
    else:
        N = N + 1
        nb = N
        T[b] = nb
    if c in T:
        nc = T[c]
    else:
        N = N + 1
        nc = N
        T[c] = nc

    if op is commutative: # handle commutative operation
        nb, nc = min(nb, nc), max(nb, nc) # canonical order

    if (nb, op, nc) in T:
        m = T[(nb, op, nc)]
        T[a] = m
        mark 'a = b <op> c' as a common subexpression
    else:
        N = N + 1
        T[(nb, op, nc)] = N
        T[a] = N
```

2.

Примечание: В задании в первой строке ‘c = a - b’ используется кириллическая буква ‘c’, вместо латинской. Я предполагаю, что это опечатка и поэтому трактую её как латинскую ‘c’.

~~a~~	↦ 1
~~b~~	↦ 2
(1, −, 2)	↦ 3
~~c~~	↦ 3
(1, ×, 2)	↦ 4
~~d~~	↦ 4
e	↦ 1
~~h~~	↦ 3
~~a~~	↦ 4
~~f~~	↦ 4
g	↦ 3
(4, +, 4)	↦ 5
d	↦ 5
~~a~~	↦ 2
~~u~~	↦ 3
(3, ×, 3)	↦ 6
~~v~~	↦ 6
~~c~~	↦ 3
u	↦ 6
h	↦ 3
(3, ×, 6)	↦ 7
c	↦ 7
(4, +, 5)	↦ 8
v	↦ 8
f	↦ 7
(7, +, 8)	↦ 9
a	↦ 9
(7, ×, 9)	↦ 10
b	↦ 10

Figure 1: Таблица значений

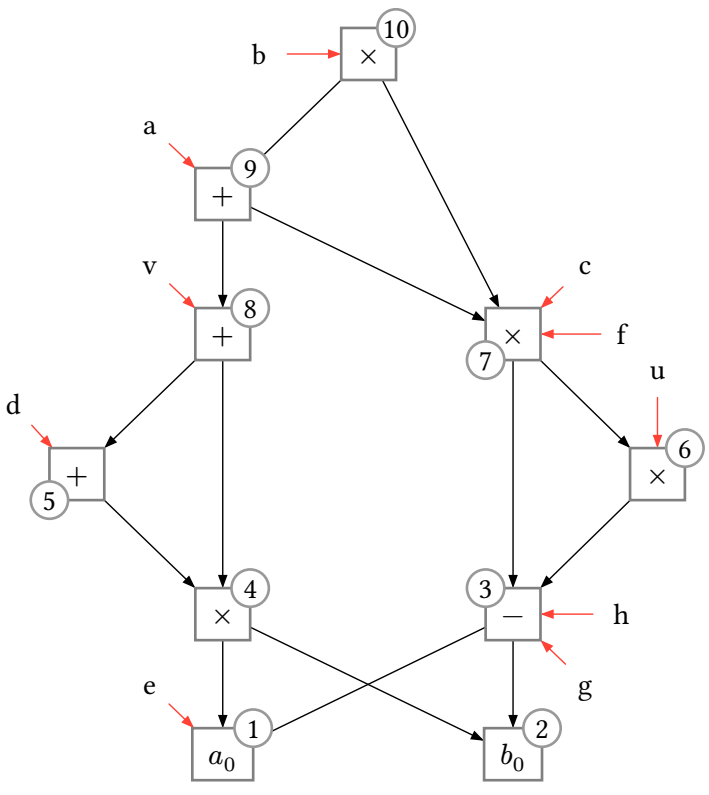

Figure 2: Граф, соответствующий таблице

□

Задание (2): Оптимизируйте следующую программу, используя SSA-представление

```
m = 30
i = 0
j = 1
n = m
while n > i:
  k = j
  if n > m:
    p = n
    while q > p:
      p = p * 2
      q = q - 1
  else:
    while i > 0:
      if m < 20:
        j = j + p
        i = i - q
        m = n
      else:
        j = j + 1
        i = i - 1
  i = k
  n = n - 1
print(j)
```

1. Переведите программу в SSA-представление
   (a) Постройте граф потока управления (control-flow graph, CFG). Убедитесь, что граф обладает свойством уникального последователя или предшественника.
   (b) Постройте дерево доминаторов.
   (c) Распишите значения границ доминирования для каждого узла в графе потока управления.
   (d) Напишите в какие блоки и для каких переменных необходимо вставить ϕ-функции. Аргументируйте решение.
   (e) Вставьте ϕ-функции и переименуйте переменные должным образом. Покажите финальный граф потока управления для SSA-представления.
2. Примените агрессивное устранение мёртвого кода:
   (a) Постройте таблицу для выполняемых присваиваний $\mathcal{E}$ и возможных значений $\mathcal{V}$.
   (b) Постройте граф управления для SSA-представления, полученный после устранения мёртвого кода.
   (c) Постройте граф управления для SSA-представления, полученный после устранения лишних пустых блоков и лишних ϕ-функций.
3. Постройте граф зависимости управления:
   (a) Постройте дерево пост-доминаторов.
   (b) Постройте границы пост-доминирования.
   (c) Постройте граф зависимости управления.
4. Постройте граф управления для SSA-представления, полученный после агрессивного устранения мёртвого кода.
5. Переведите программу из SSA-представления:
   (a) Постройте граф конфликтов (интерференции) для переменных в SSA-представлении.
   (b) Постройте полученный граф потока управления оптимизированной программы: замените ϕ-функции на копирующие инструкции (в предшествующих блоках), переименуйте переменные (согласно графу конфликтов), и уберите лишние копирующие инструкции.

Решение:

1. (a)

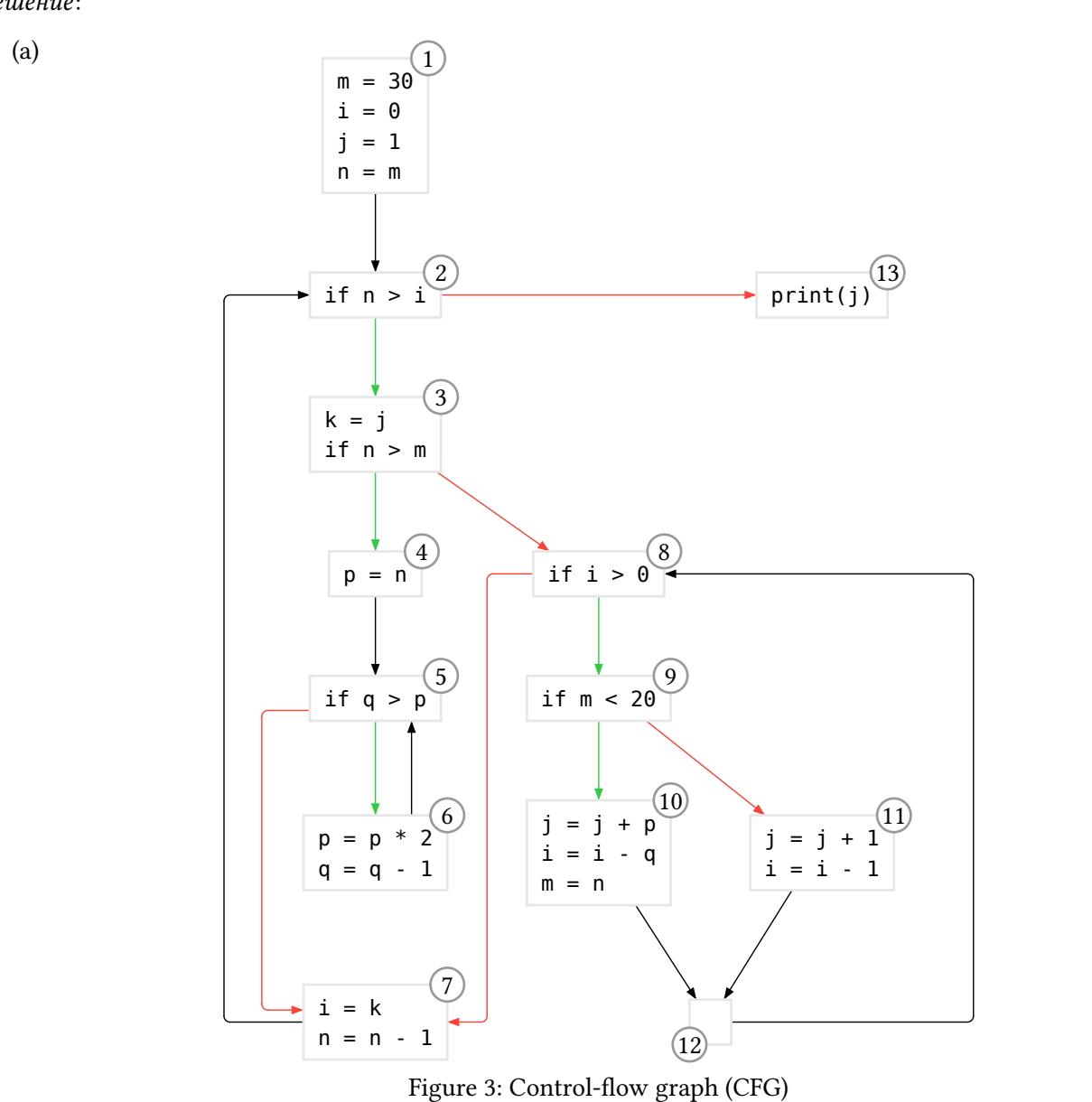

Figure 3: Control-flow graph (CFG)

Данных граф обладает свойством уникального предшественника и уникального последователя, однако мы временно нарушим это свойство для уменьшения количества узлов и вернёт нужные веришны на этапе перевода из SSA-представления обратно.

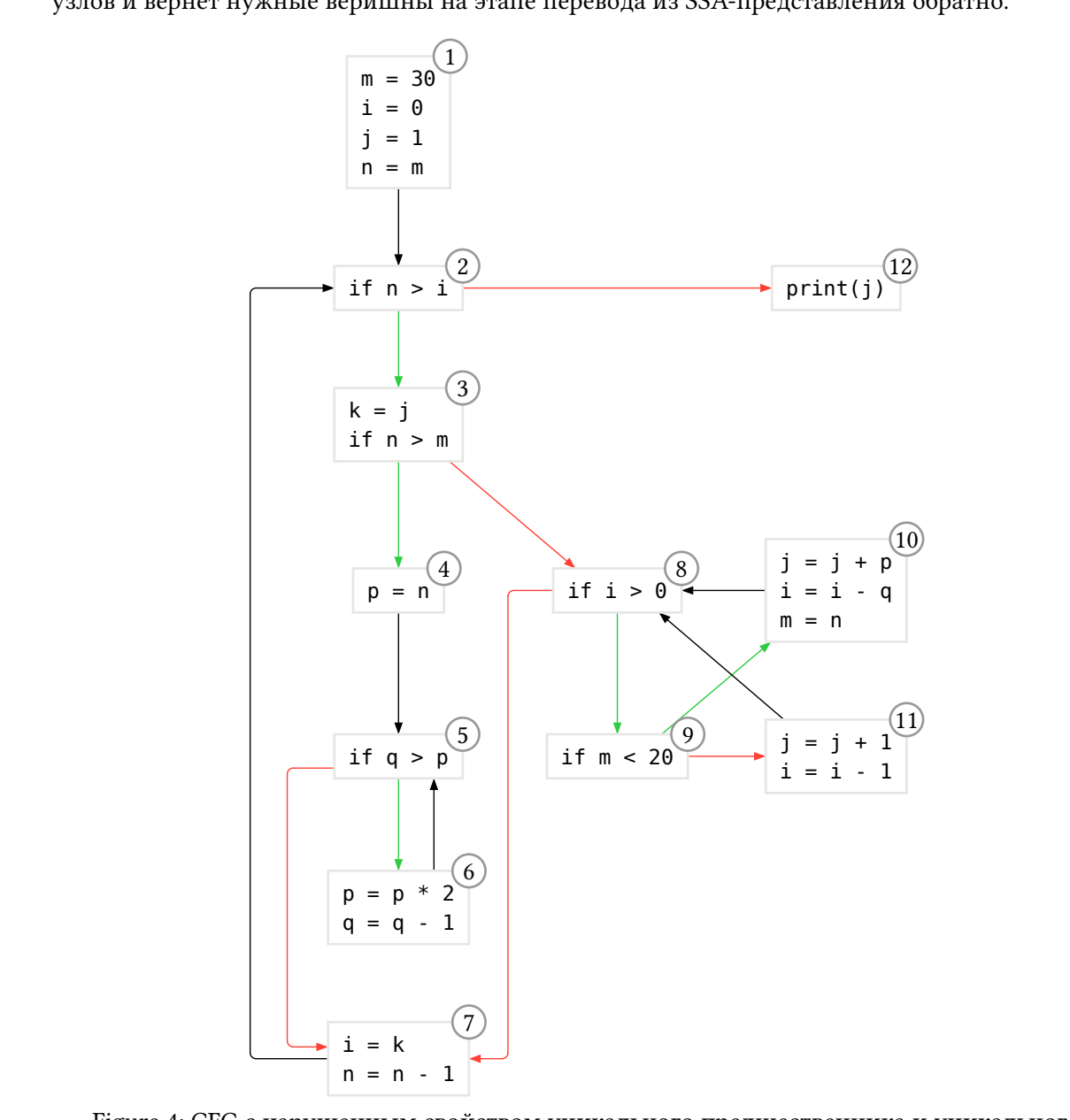

Figure 4: CFG с нарушенным свойством уникального предшественника и уникального последователя

(b)

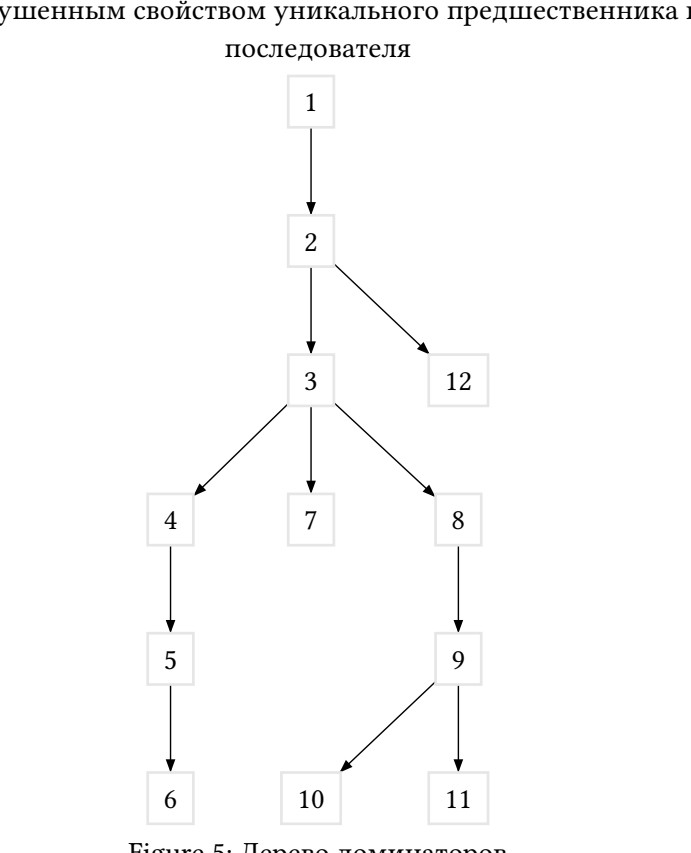

Figure 5: Дерево доминаторов

(c)

Block	Dominance Frontier
1	{}
2	{2}
3	{2}
4	{7}
5	{5, 7}
6	{5}
7	{2}
8	{7, 8}
9	{8}
10	{8}
11	{8}
12	{}

(d) Для каждой переменной посмотрим блоки, где она присваивается и найдём границы доминирования этих блоков.

Var	Blocks with definition
n	{1, 7, 2}
m	{1, 10, 8, 7, 2}
i	{1, 7, 10, 11, 2, 8}
j	{1, 10, 11, 8, 7, 2}
p	{4, 6, 7, 5, 2}
q	{6, 5, 7, 2}

Расставим ϕ-функции в блоки, которые входят в границы доминирования **Blocks with definition**.

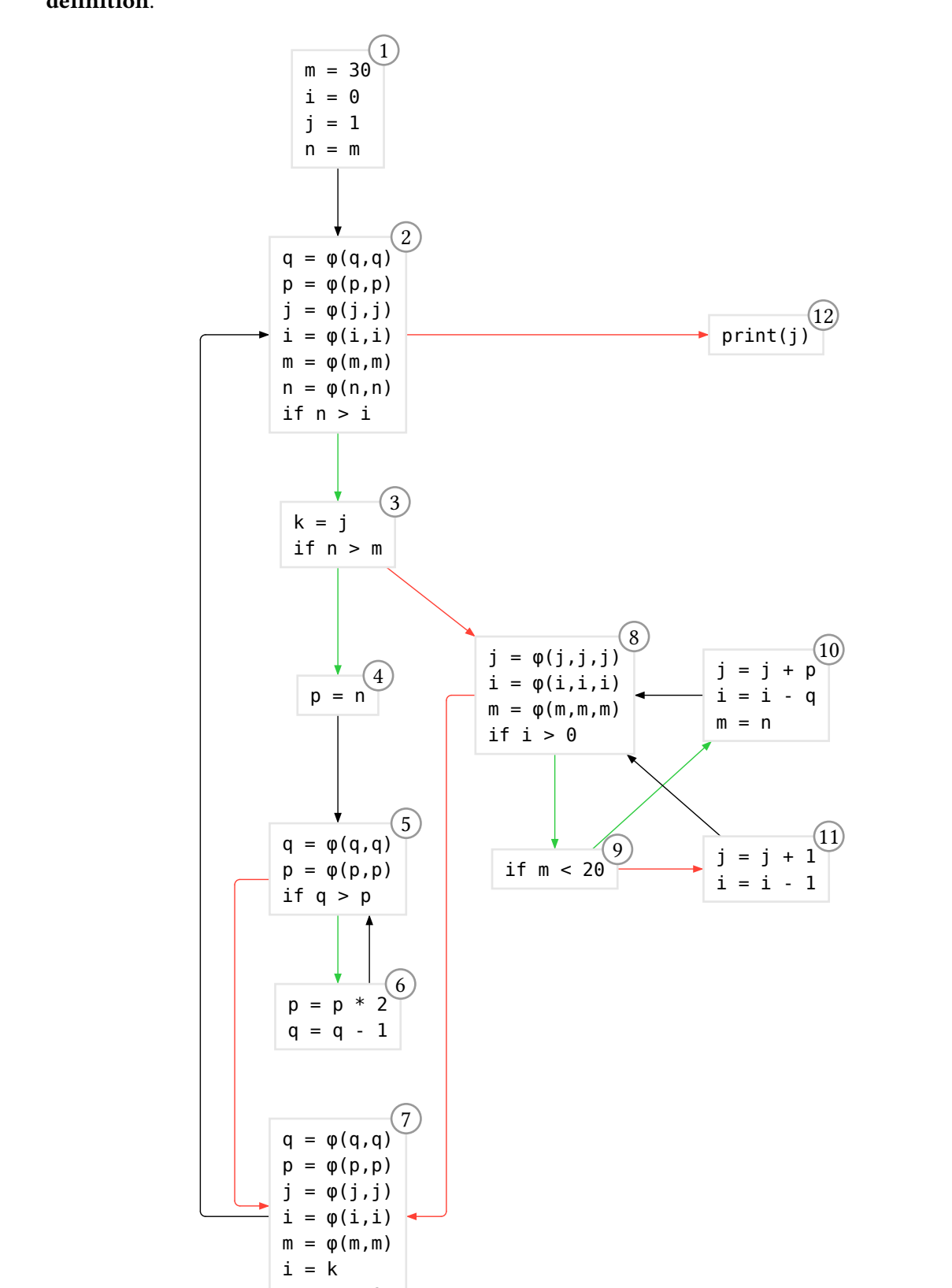

Figure 6: CFG с вставленными ϕ-функциями

(e)

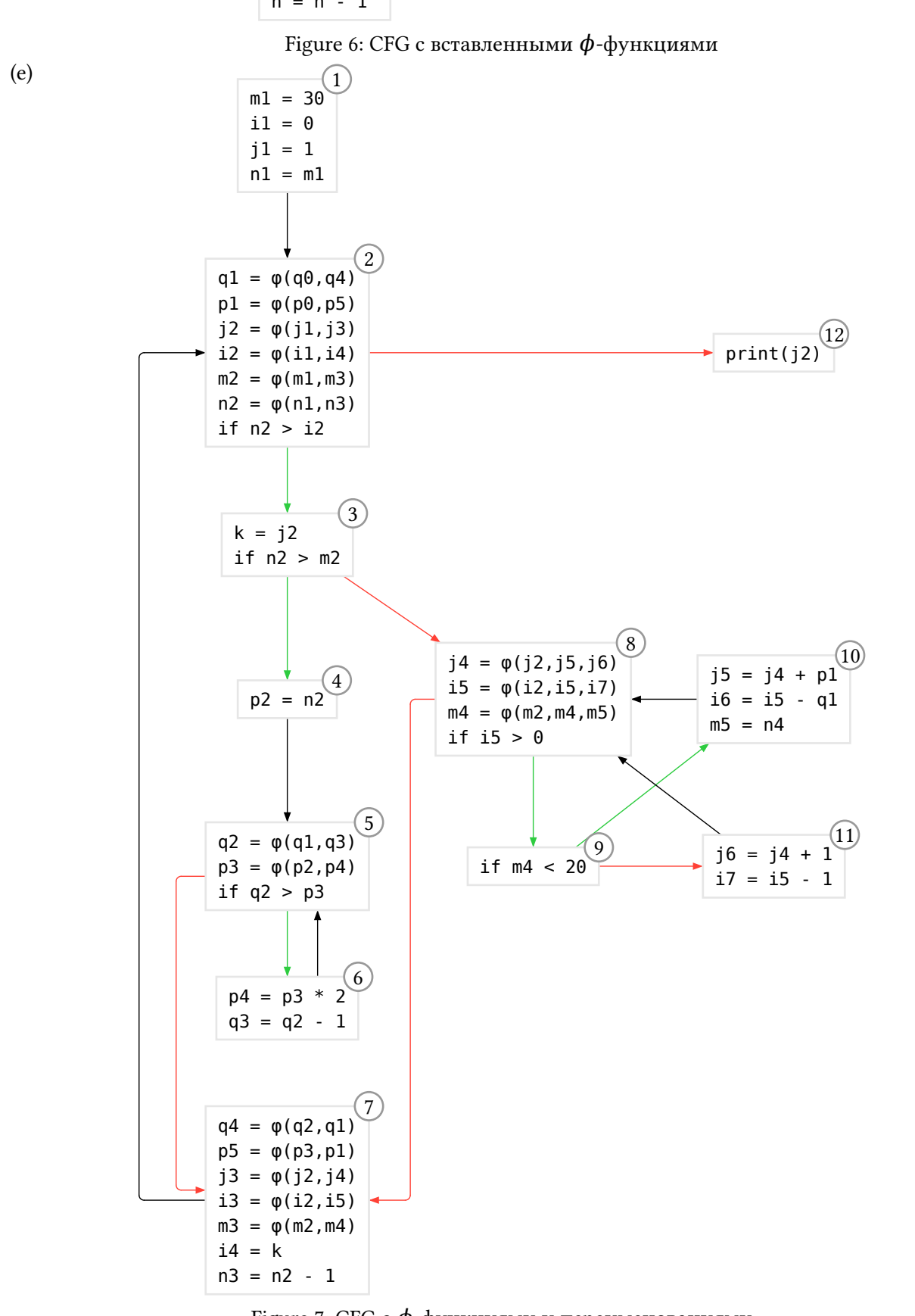

Figure 7: CFG с ϕ-функциями и переименованиями

2. (a)

Block	1	2	3	4	5	6	7	8	9	10	11	12
$\mathcal{E}$	true	true	true	true	true	true	true	true	true	false	true	true

Table 1: Выполняемые присваивания

Var	m1	m2	m3	m4	m5	n1	n2	n3	i1	i2	i3	i4	i5	i6	i7	j1	j2	j3	j4	j5	j6
$\mathcal{V}$	30	30	30	30	⊥	30	⊤	⊤	0	⊤	⊤	⊤	⊤	⊥	⊤	1	⊤	⊤	⊤	⊥	⊤

Table 2: Возможные значения

Var	p0	p1	p2	p3	p4	p5	q0	q1	q2	q3	q4	k
$\mathcal{V}$	⊤	⊤	⊤	⊤	⊤	⊤	⊤	⊤	⊤	⊤	⊤	⊤

Table 3: Возможные значения

(b)

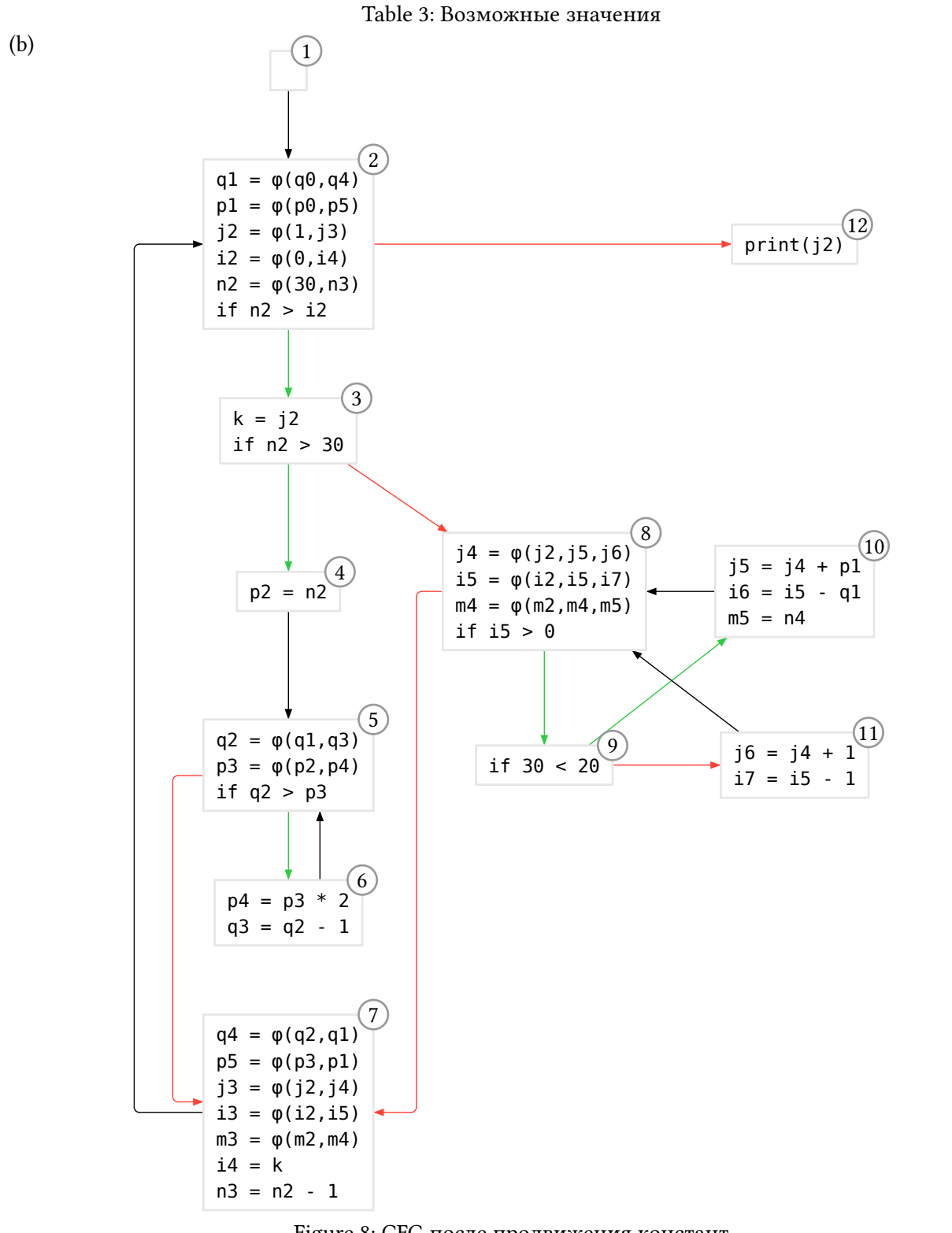

Figure 8: CFG после продвижения констант

(c)

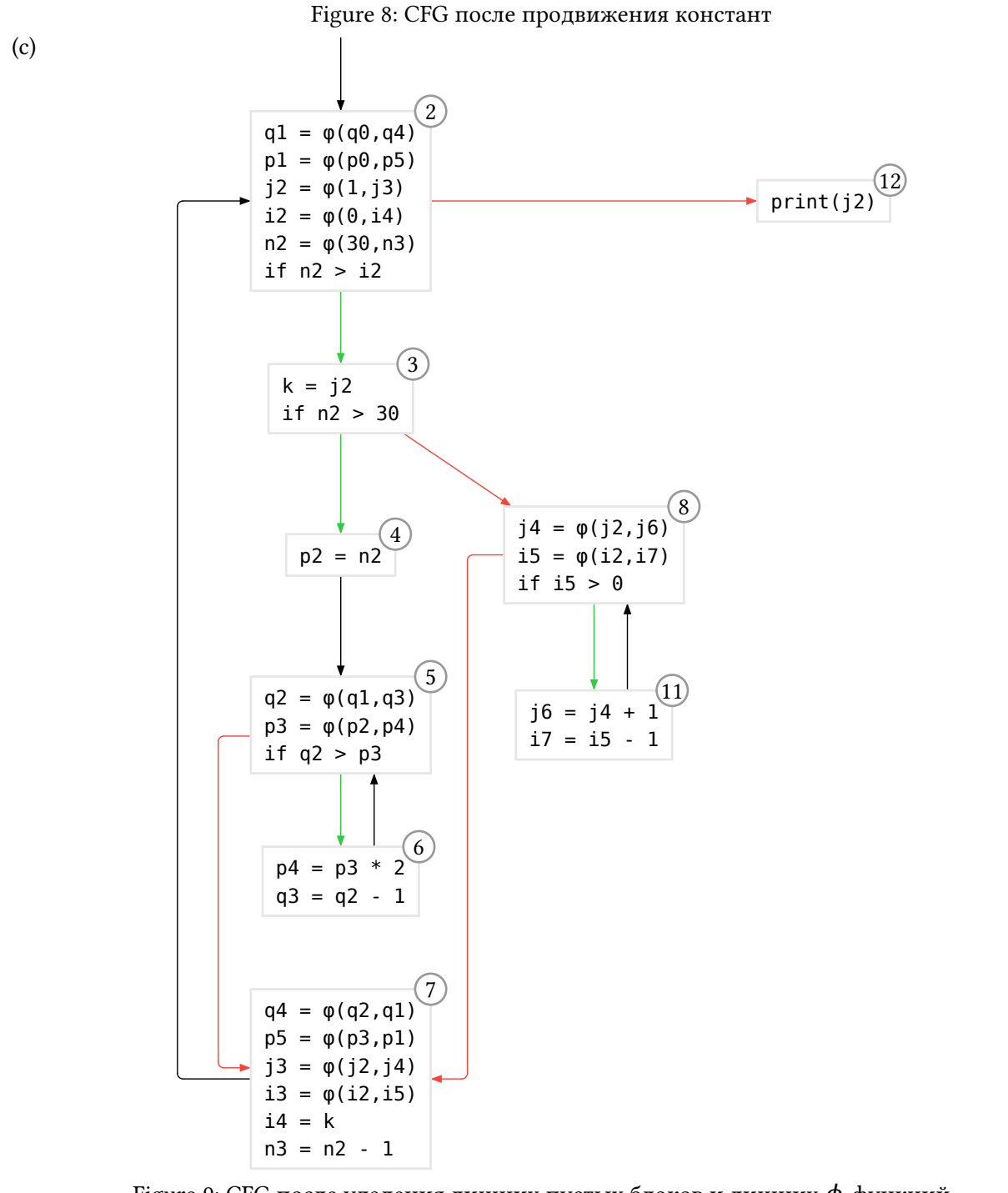

Figure 9: CFG после удаления лишних пустых блоков и лишних ϕ-функций

3. (a)

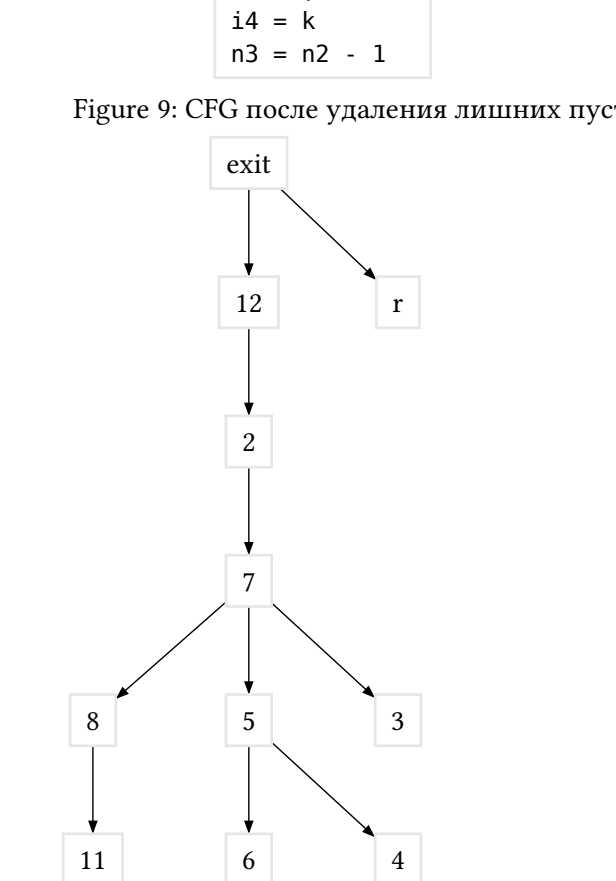

Figure 10: Дерево пост-доминаторов

(b)

Block	Post-Dominance Frontier
1	{}
2	{1, 2}
3	{2}
4	{3}
5	{3, 5}
6	{5}
7	{2}
8	{3, 8}
11	{8}
12	{1}
13	{}

(c)

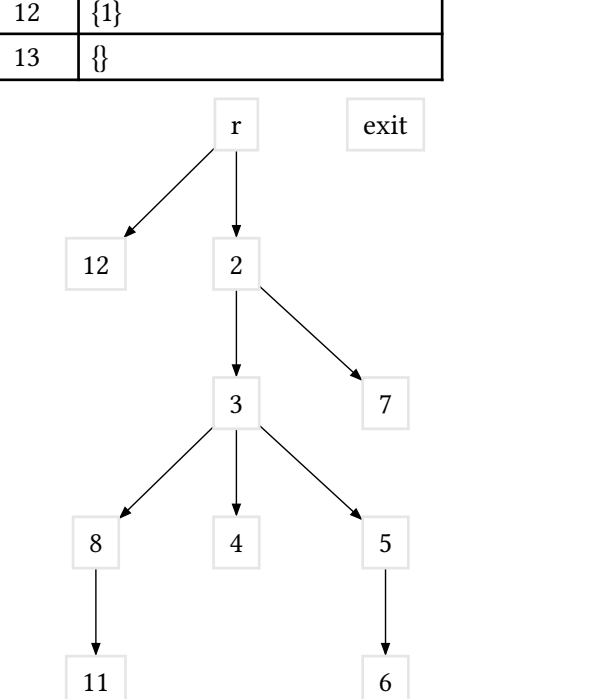

Figure 11: Control Dependence Graph (CDG)

4.

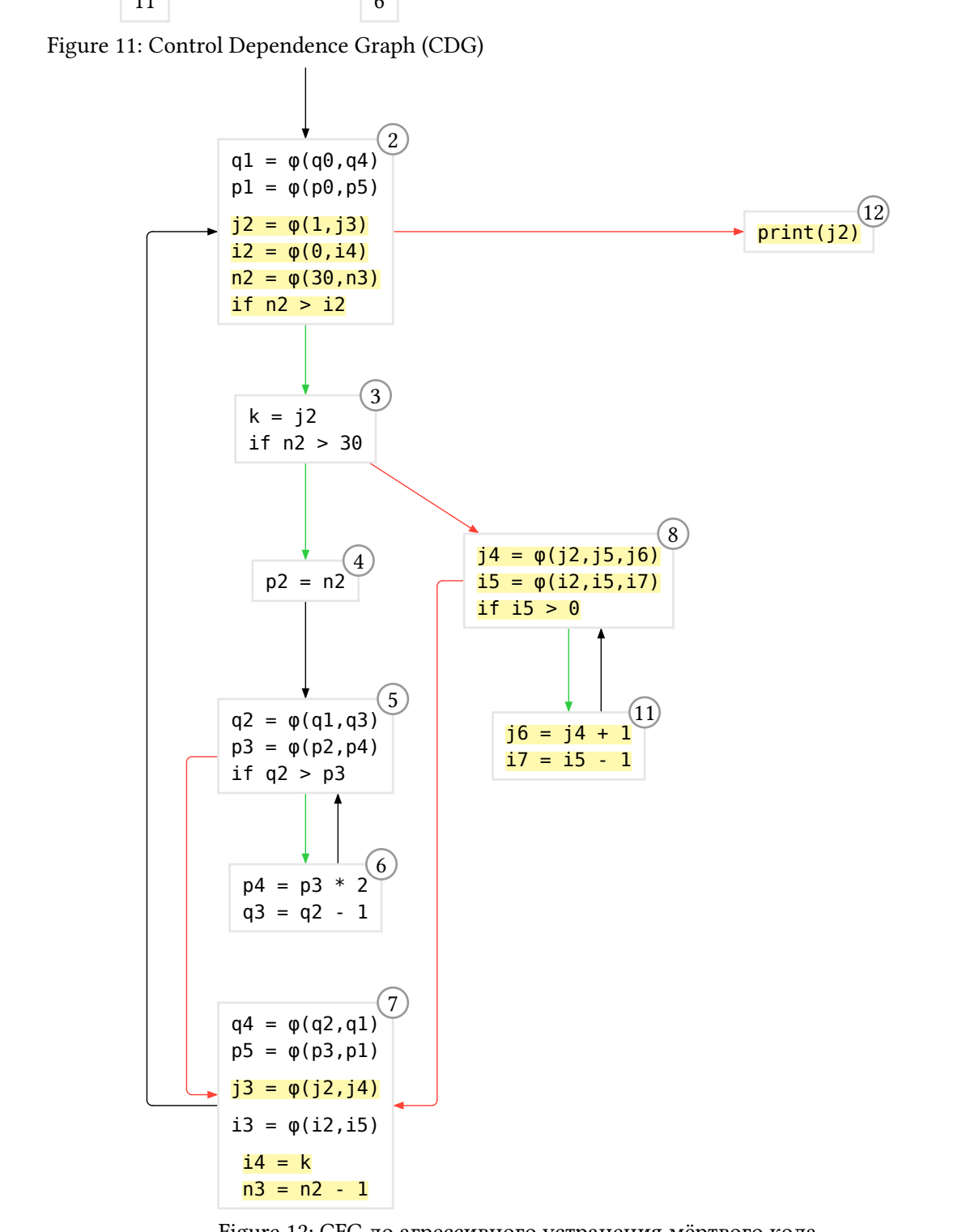

Figure 12: CFG до агрессивного устранения мёртвого кода

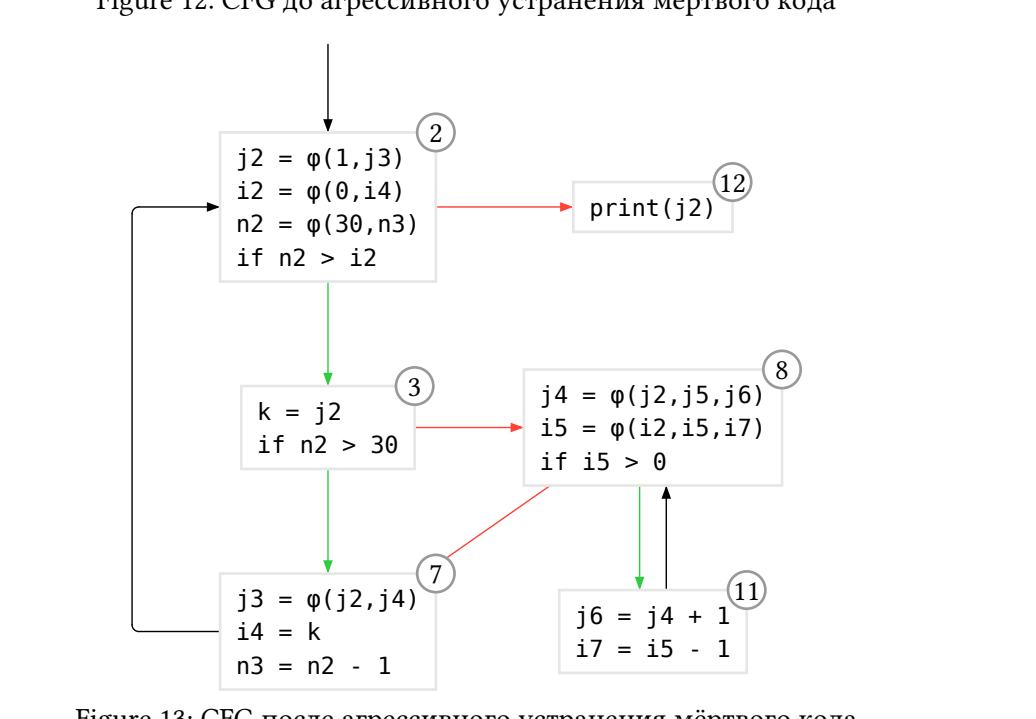

Figure 13: CFG после агрессивного устранения мёртвого кода

5. (a)

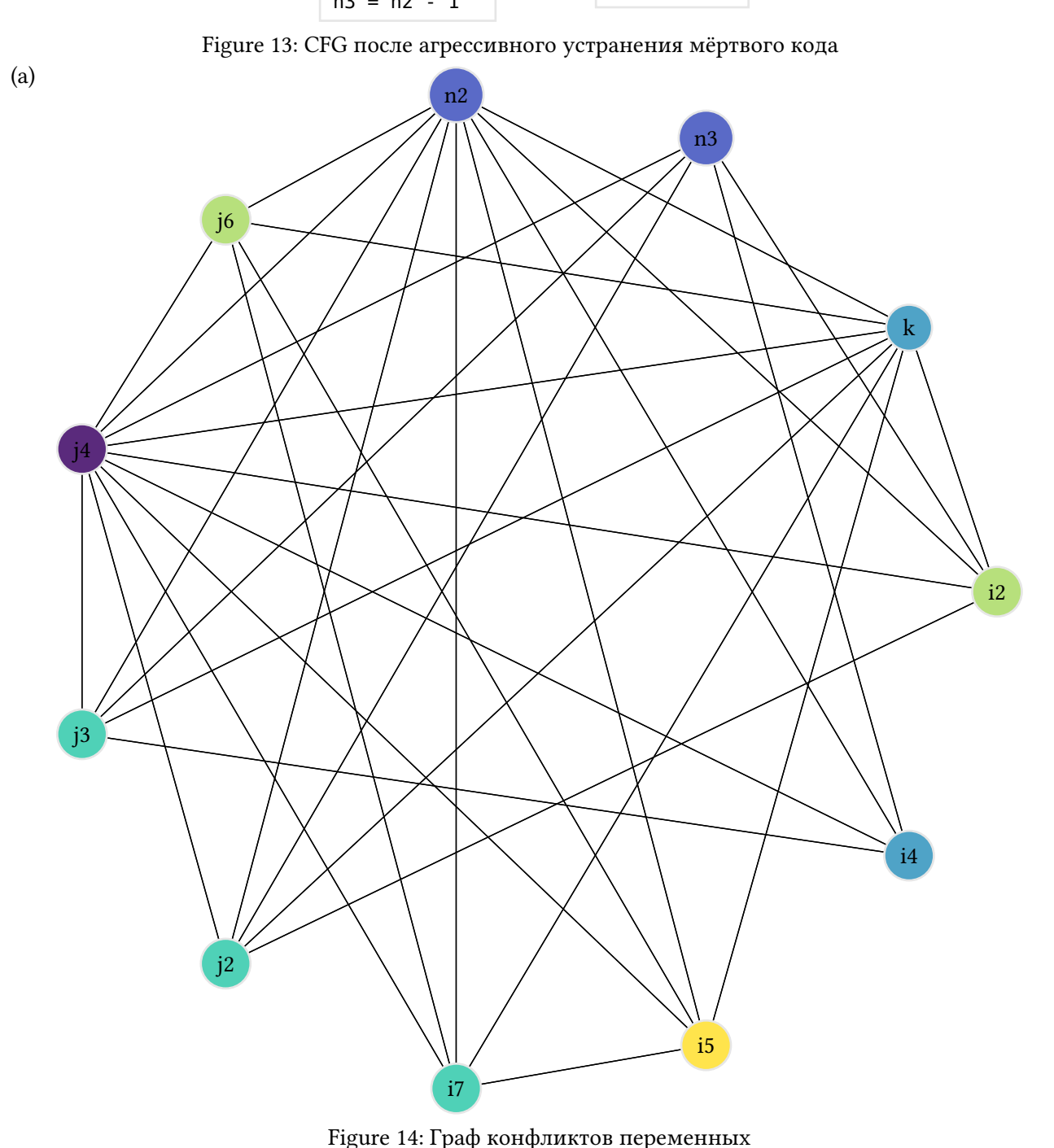

Figure 14: Граф конфликтов переменных

(b) Вернём графу свойство уникального предшественника и уникального последователя, добавив блоки 13 и 14.

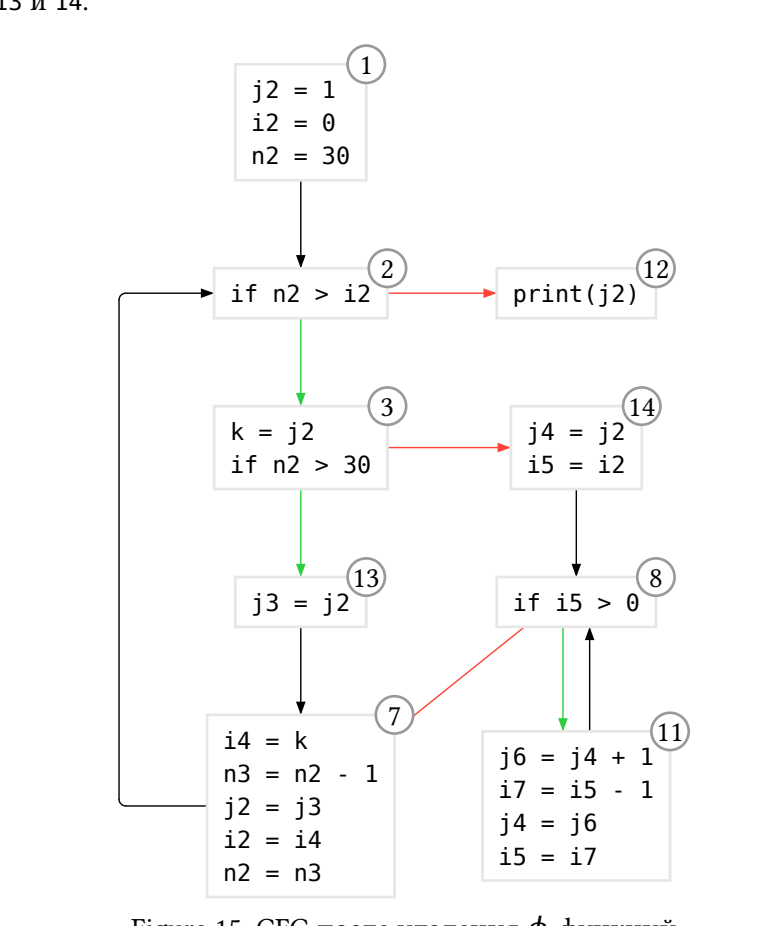

Figure 15: CFG после удаления ϕ-функций

Переименуем переменные: переменным одинакового цвета из графа конфликтов дадим одинаковые имена.

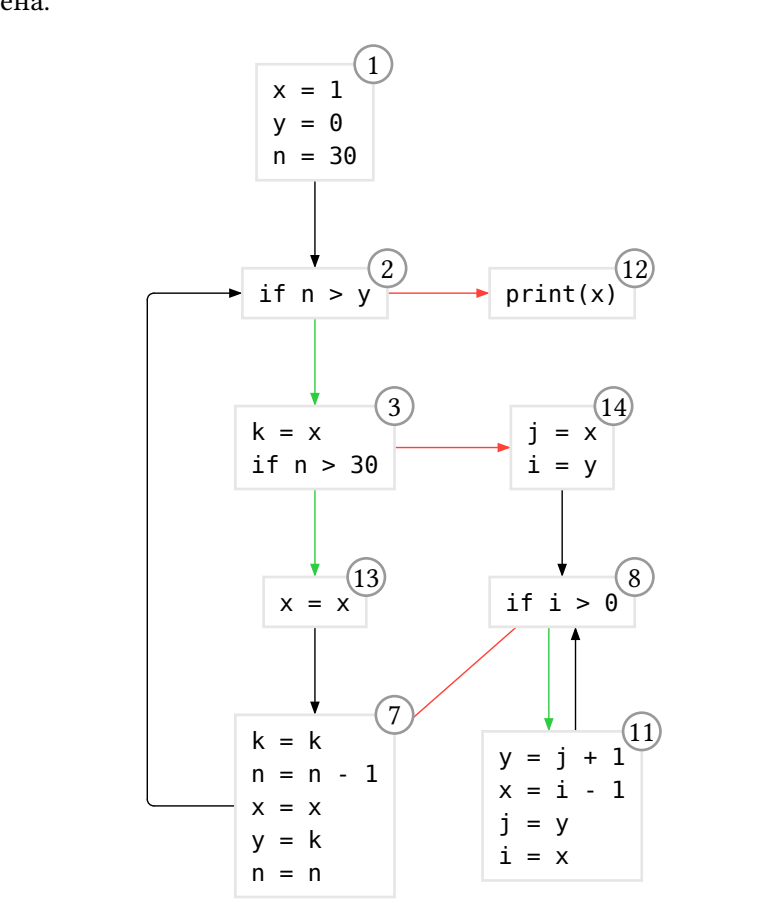

Figure 16: CFG после переименования переменных

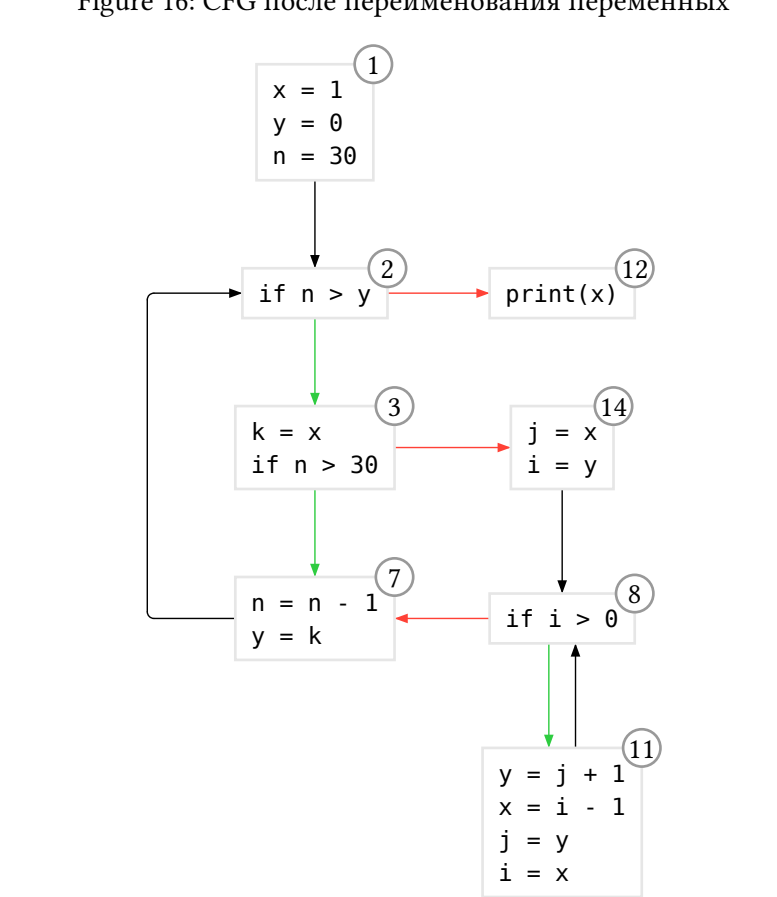

Figure 17: CFG после удаление лишних копирований

□