# Control Flow Graph (CFG)

Граф потока управления

Синявин А.В.

# Понятие CFG

Def: **Базовый блок** - максимальные последовательности кода, обладающие следующими св-ми:

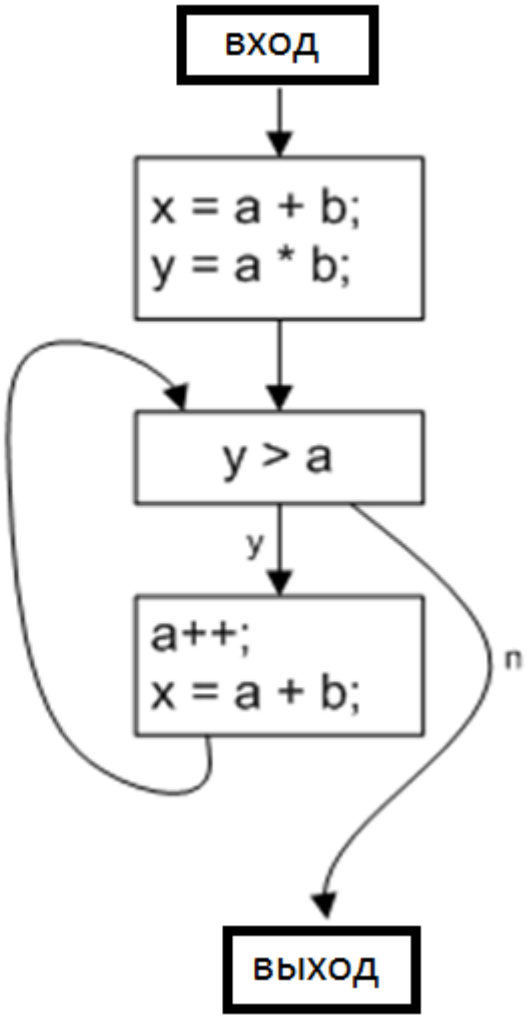
- поток управления может входить только через первую команду

- управление покидает блок без останова или ветвления, за исключением, быть может последней команды

Def: **Граф потока управления (CFG)** – ориентированный граф. Базовые блоки – вершины графа. Дуги показывают возможные передачи управления. Задана одна входная вершина. Задано мн-во выходных вершин.

# Понятие CFG

```
x = a + b;
y = a * b;
while( y > a )
{
    a++;
    x = a + b;
}
```

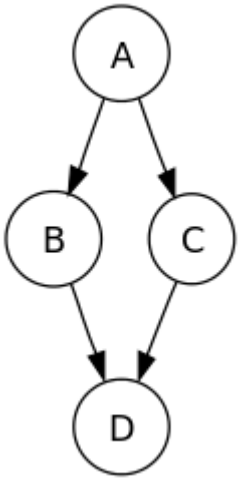


# Обозначения для CFG

| id | значение |
|---|---|
| class vertex | вершина графа |
| class edge | дуга графа |
| edge.v1 | начальная вершина графа |
| edge.v2 | конечная вершина графа |
| Succ(v), v – объект vertex | набор вершин преемников |
| Pred(v), v – объект vertex | набор вершин предшественников |
| InEdge(v), v – объект vertex | набор входящих дуг |
| OutEdge(v), v – объект vertex | набор исходящих дуг |
| AllVert(g), g – CFG – граф | набор всех вершин |
| AllEdge(g), g – CFG – граф | набор всех дуг |
| g.entry | входящая вершина графа |

# Порядки вершин при DFS пример

DFS: A B D B A C A




Preorder: A B D C

Postorder: D B C A

Reverse postorder: A C B D

# Классификация дуг в CFG

Алгоритм: построение глубинного остовного дерева и упорядочение графа в глубину
Вход: g CFG граф
Выход:
- дуги остовного дерева
- распределение дуг по классам
- NPre (preorder number)
- NPost (postorder number)
- NRPost (reverse postorder number)

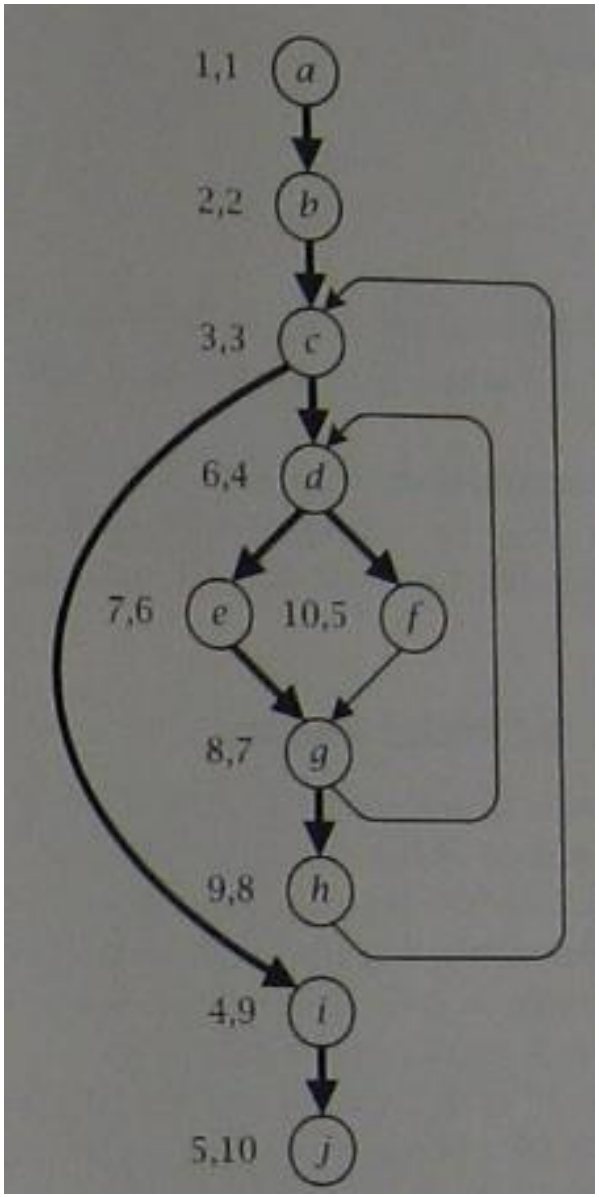
```
map<vertex, int> NPre;
map<vertex, int> NPost;
map<vertex, int> NRPost;
int N = 0, K=0;
int M = AllVert(g).get_num();

“элементы NPre, NPost, NRPost обнулены для всех вершин”;
search(g.entry);
/* продолжение на следующем слайде */
```

```
void search(vertex v)
{
    N++;
    NPre[v] = N;
    foreach(edge e in OutEdge(v))
    {
        if ( NPre[e.v2] == 0 )
        {
            search(e.v2);
          /* e принадлежит глубинному остовному дереву (1) */
        }
        else if ( NPre[v] < NPre[e.v2] )
          /* advancing (наступающие) e (2) */
        else if ( NRPost[e.v2] == 0 )
          /* retreating (отступающие) e  (3) */
        else /* cross e (4) */
    }
    NRPost[v] = M;
    M--;
    K ++;
    NPost[v] = K;
}
```

В “Ахо – Ульмане” классы дуг 1 и 2 объединены

дуги, которые идут от узла m к предшественнику узла m в дереве (возможно, к самому узлу m )

# Классификация дуг (пример)




Жирные дуги – глубинное остовное дерево

Для каждой вершины первое число NPre, второе - NRPost

g->d, h->c - retreating дуги

f->g – cross дуга

# Доминаторы

Пусть дан некоторый CFG-граф. E – входная вершина. X, Y – некоторые вершины.

Def: Отношение доминирования (нестрого).

X dom Y <=> все пути от E до Y обязательно проходят через X.

Замечание: ∀ X справедливо X dom X

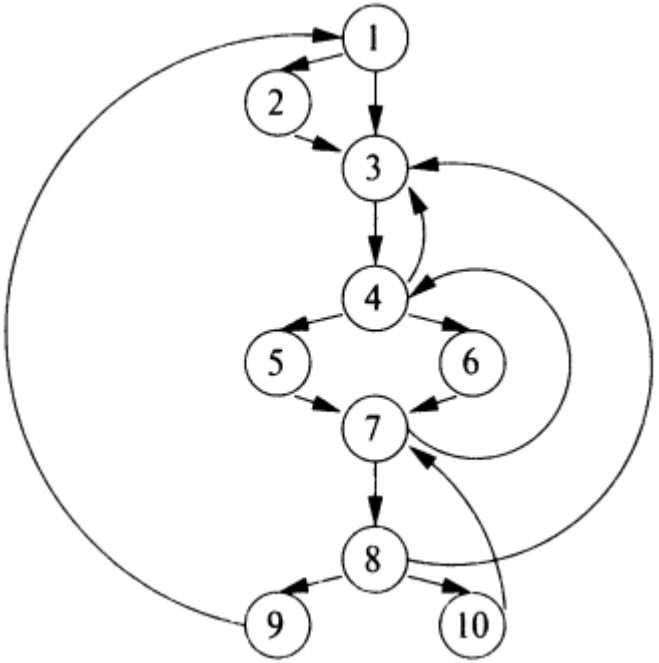
Def: Отношение непосредственного доминирования.

X idom Y <=> X != Y и X dom Y и ∄ C | C != X и C != Y и X dom C и C dom Y

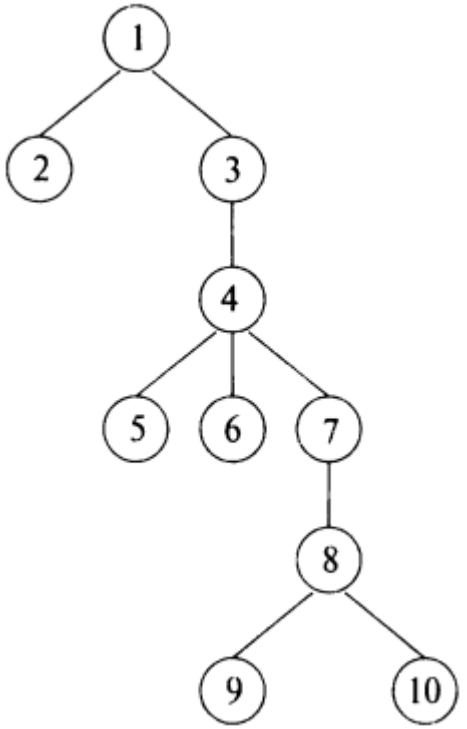
# Доминаторы

| Символ | Значение |
|---|---|
| X dom Y | X нестрого доминирует над Y |
| X sdom Y | X строго доминирует над Y. X dom Y и X != Y |
| X idom Y | X непосредственно доминирует над Y. |
| X ~~dom~~ Y | Отрицание X dom Y |
| X ~~sdom~~ Y | Отрицание X sdom Y |
| X ~~idom~~ Y | Отрицание X idom Y |
| idom(Y) | idom(Y)  idom  Y ≡ true |

# Доминаторы




Исходный CFG




Дерево доминаторов

# Постдоминаторы

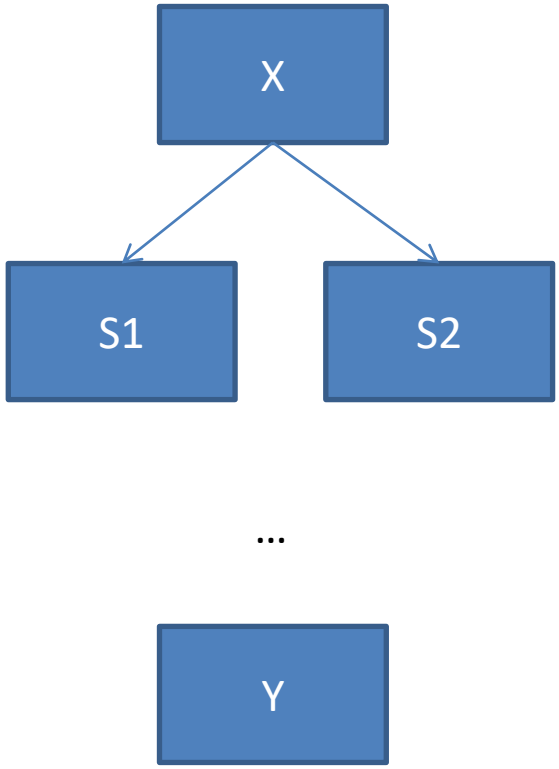
Пусть дан некоторый CFG-граф. O – выходная вершина. X, Y – некоторые вершины.

Def: Отношение постдоминирования (нестрого).

X pdom Y <=> все пути от Y до O обязательно проходят через X.

Замечание: понятия spdom, ipdom, постдоминаторного дерева формулируются аналогично на базе pdom

# Зависимость ББ по управлению




Def: Y зависит от X по управлению

<=>

- ∃ S ∈ succ(X) | Y pdom S и
- ∃ S ∈ succ(X) | Y ~~pdom~~ S

# Обратные рёбра и приводимость

Def: Обратное ребро (back edge) называется ребро a -> b, у которого b dom a

Утв. Каждое обратное ребро является отступающим.

◅ Пусть **a -> b** – обратное ребро => **b dom a**

последовательность рекурсивных вызовов search, приводящая к узлу **a**, соответствует некоторому пути в графе. Этот путь должен содержать **b**.

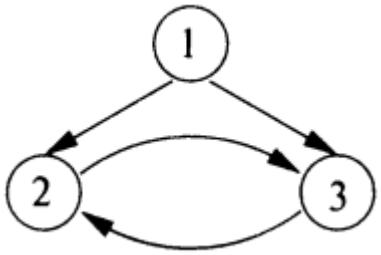
=>

вызов **search(b)** должен быть открыт, когда вызывается **search(a)**, **NRPost[b] = 0** (b находится в стеке), т.е. **a->b** – отступающее ребро

▻

# Обратные рёбра и приводимость

Не всякое отступающее ребро является обратным.




2 ~~dom~~ 3

3 ~~dom~~ 2

Если из search(1) первым будет вызван search(2), то 3 -> 2 отступающее ребро (но не обратное)

Если из search(1) первым будет вызван search(3), то 2 -> 3 отступающее ребро (но не обратное)

# Обратные рёбра и приводимость

Def: CFG – приводимый граф, если ∄ отступающего ребра, которое не является обратным.

Все CFG, которые получаются при помощи

- **if then / if then else / switch**
- **for / while / do while**
- **continue / break**

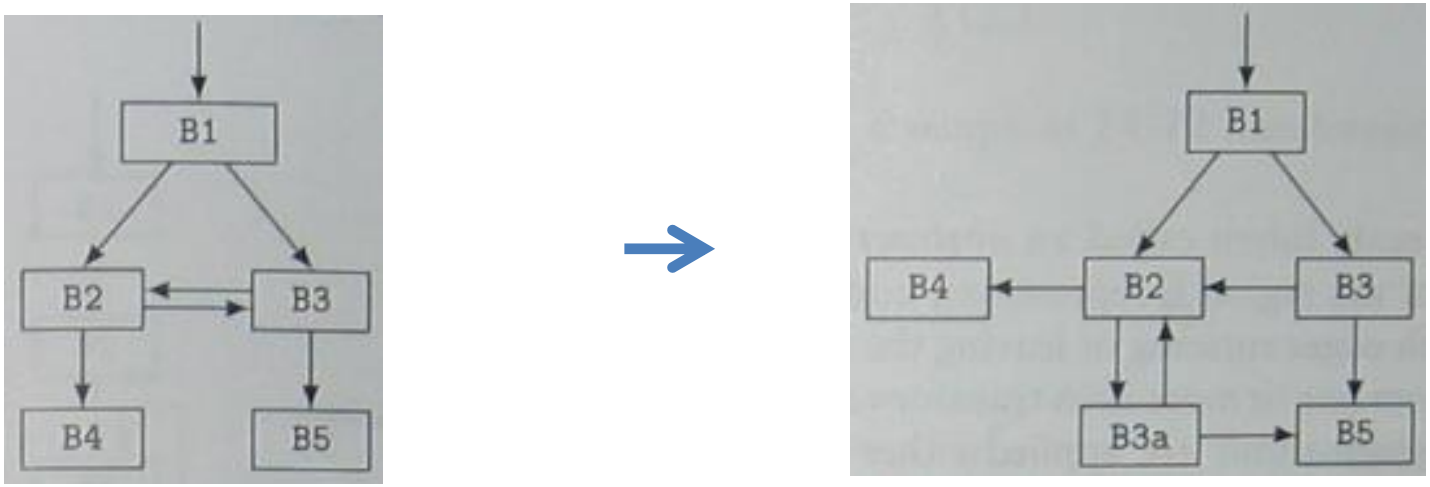
являются приводимыми

Неприводимым CFG может получится при помощи **goto**

Т.о., случай неприводимого CFG довольно редкий

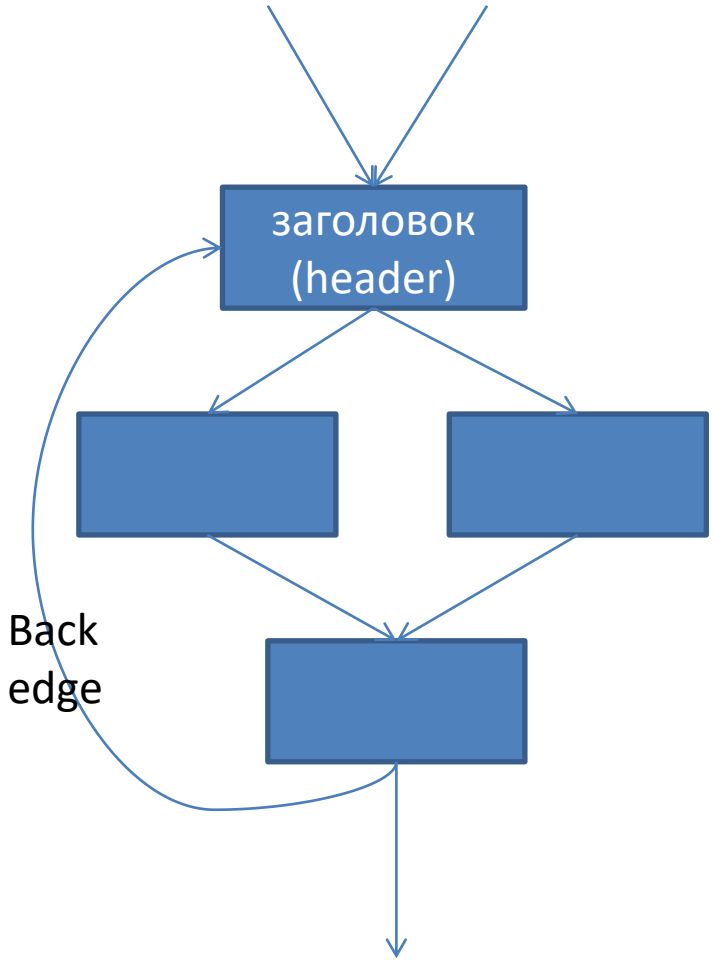
# Обратные рёбра и приводимость

Неприводимые CFG можно трансформировать к приводимым



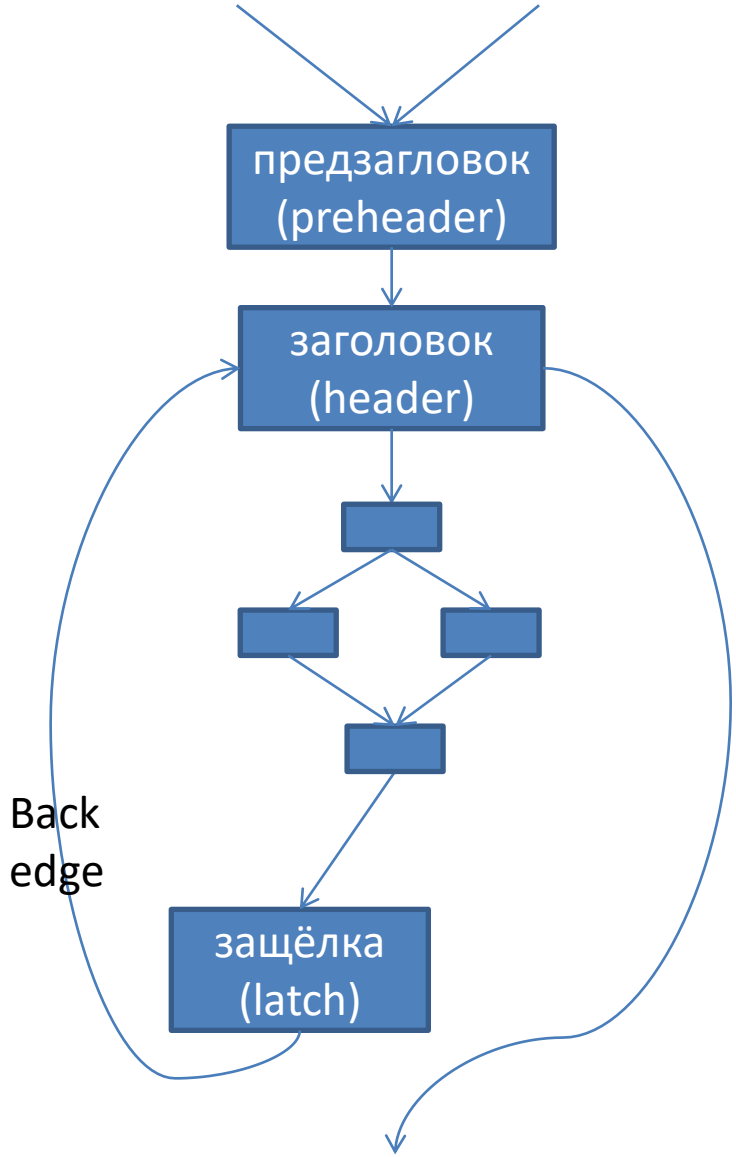

∃ алгоритмы, которые преобразуют неприводимые графы к приводимым

# Естественные циклы




1) Заголовок – единственный входной узел. Заголовок доминирует над всеми узлами цикла.
2) Обратное ребро ведёт в заголовок

# Канонический вид циклов




Часто циклы приводятся к некоторому единому виду, чтобы все фазы могли единообразно с ними работать.

Два дополнительных пустых ББ

- preheader
- latch

# Выявление естественных циклов

Алгоритм: построение естественного цикла для обратной дуги

Вход: g CFG граф и back edge n -> d

Выход: мн-во ББ L, которые входят в цикл

Метод:

“L = {n, d}”;

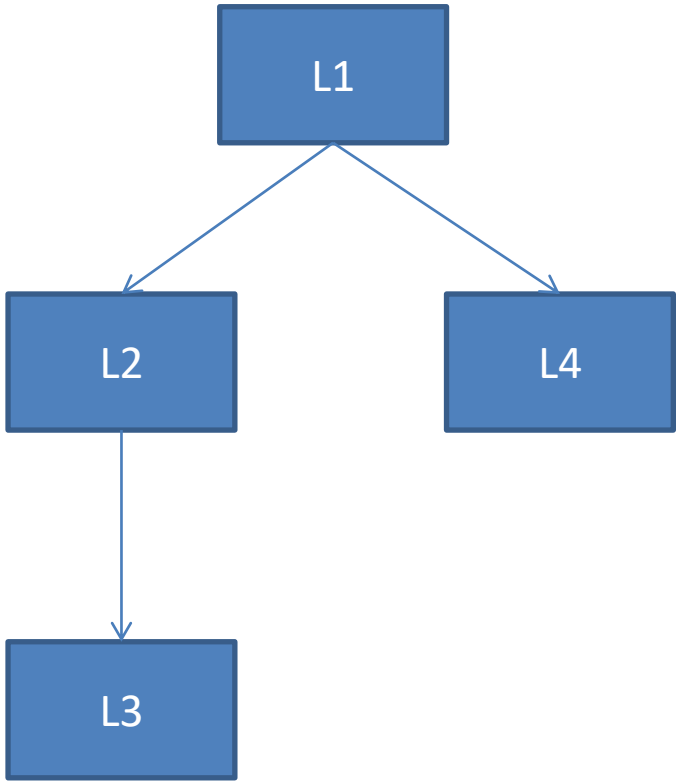
“пометим d как посещённый”;

“выполним DFS против направления рёбер, начиная с n”;

“внесём все посещённые узлы в L”;

# Дерево циклов

```
for (…) // L1
{
   for(…) // L2
   {
      for(…) // L3
      {
         …
      }
   }
   for(…) // L4
   {
      …
   }
}
```



# Control Flow Analysis (CFA)

- Выявление обратных рёбер (back edge) и циклов
- Вычисление доминаторов и постдоминаторов
- Вычисление зависимостей ББ по управлению
- Построение дерева циклов
- T1 – T2 анализ
- Структурный CFA

# T1 – T2 анализ

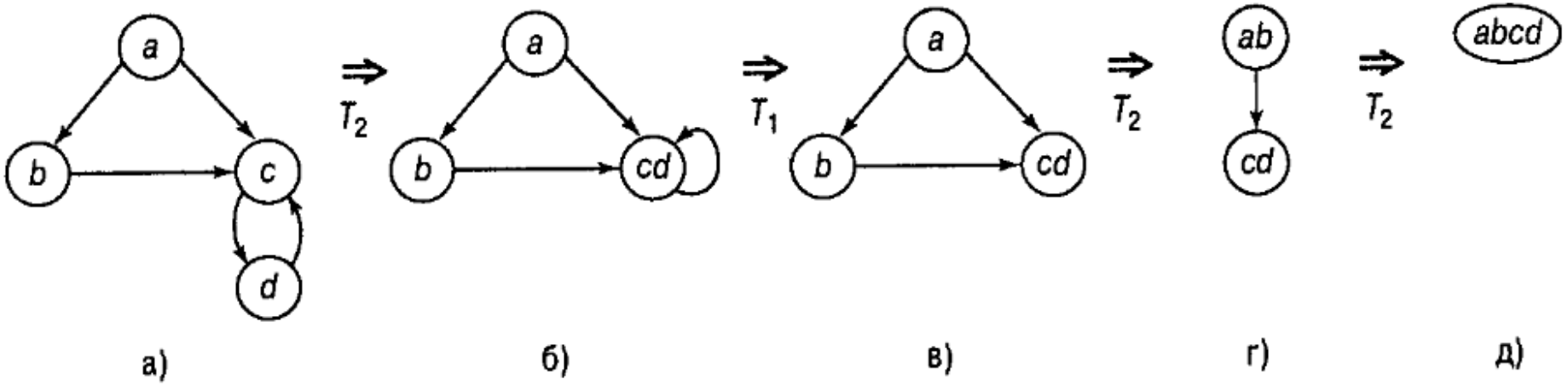
Преобразование T1:

Удалить дуги вида n -> n

Преобразование T2:

Если ∃ (не входной) узел n, имеющий единственного предшественника m, то m может поглотить n, удаляя его и делая всех преемников n (включая, возможно, m) преемниками m

# T1 – T2 анализ



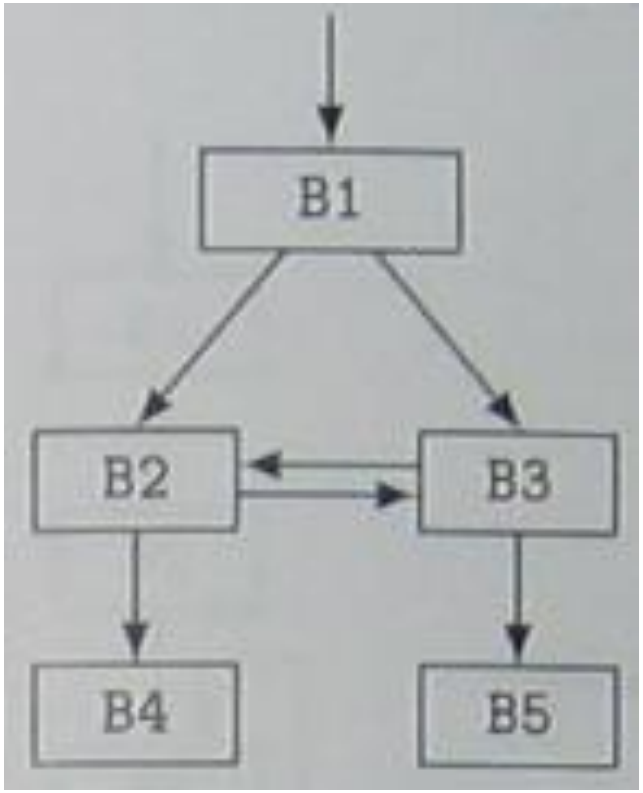
# T1 – T2 анализ

Св-ва преобразований T1/T2:

- Граф потока, полученный исчерпывающим применением T1/T2 к исходному CFG, даёт некий предельный граф G
- предельный граф G не зависит от порядка применения преобразований

# T1 – T2 анализ

Вопрос: как будет вести себя алгоритм T1/T2 на неприводимых графах?



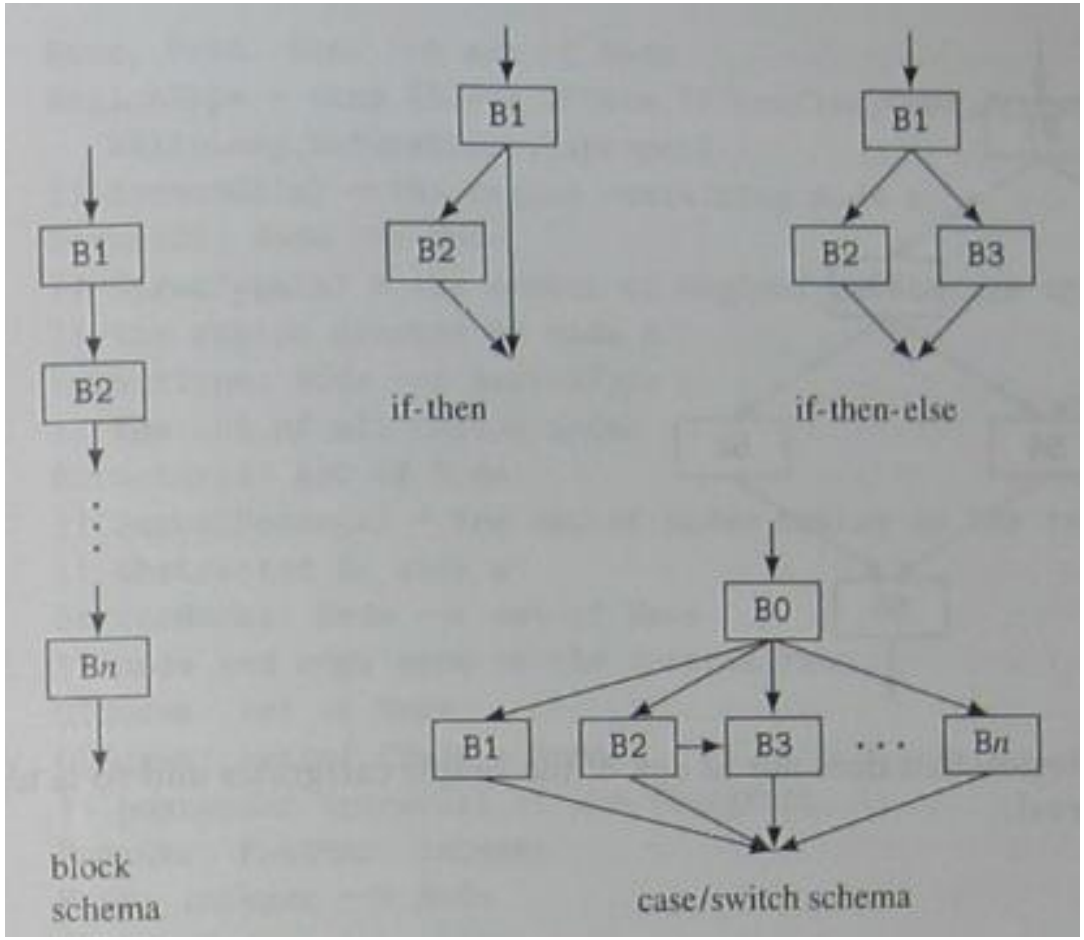
# T1 – T2 анализ

Def: Граф является приводимым

<=>

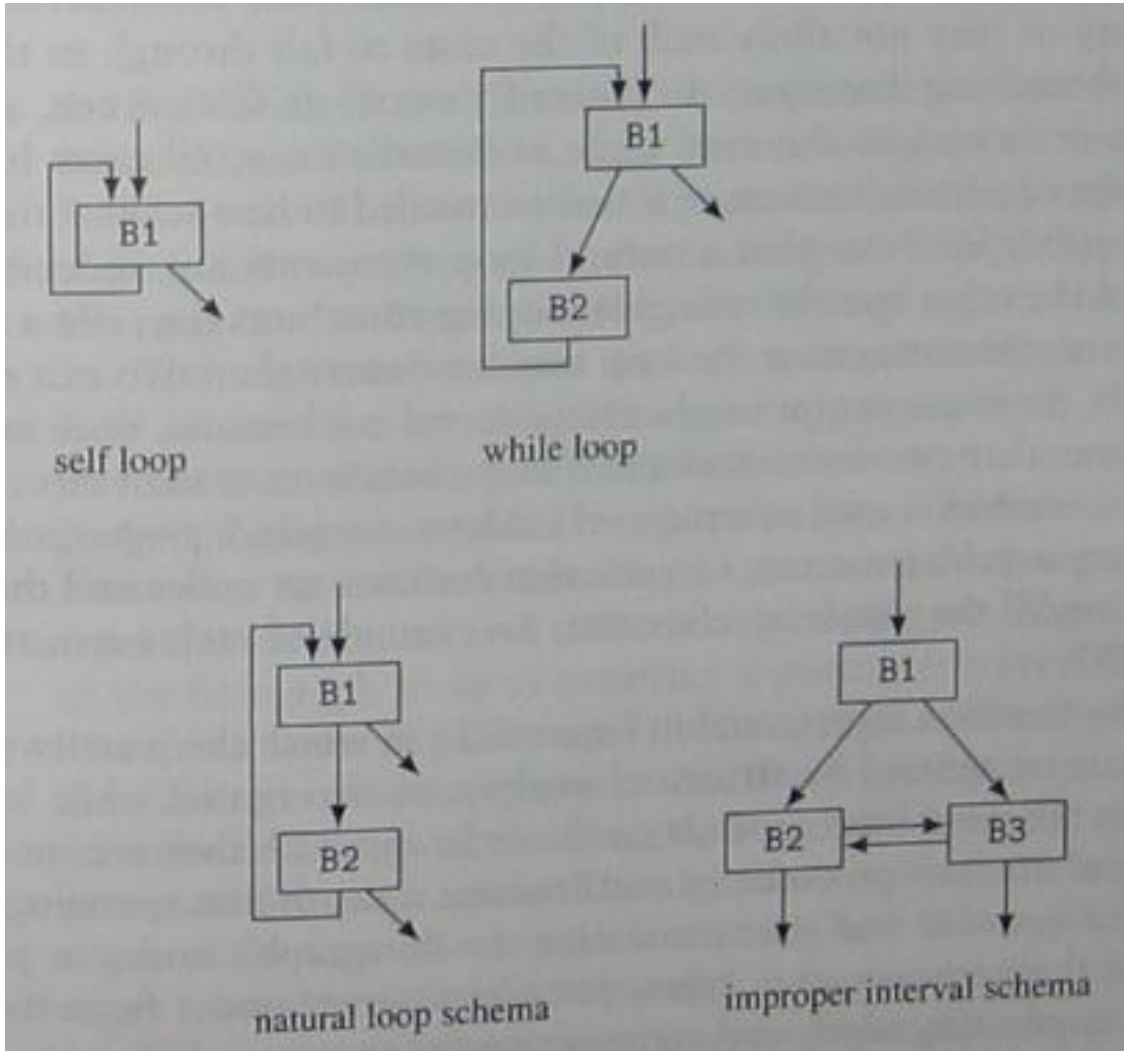
его предельный граф представляет собой одну вершину

# Структурный CFA




Ациклические типы регионов

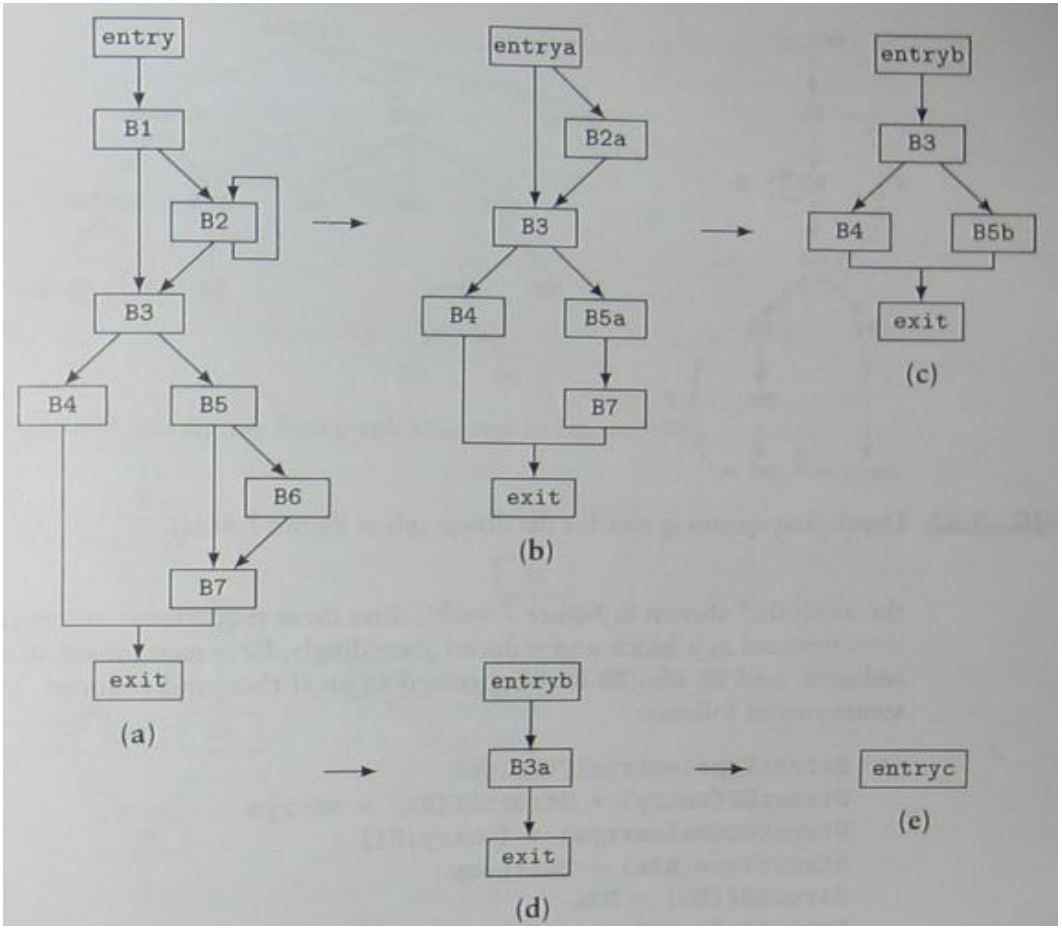
# Структурный CFA



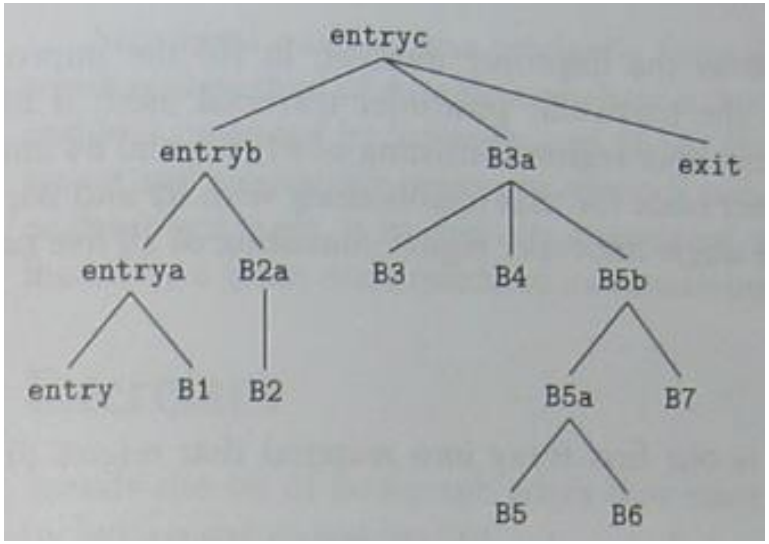

Циклические типы регионов

# Структурный CFA

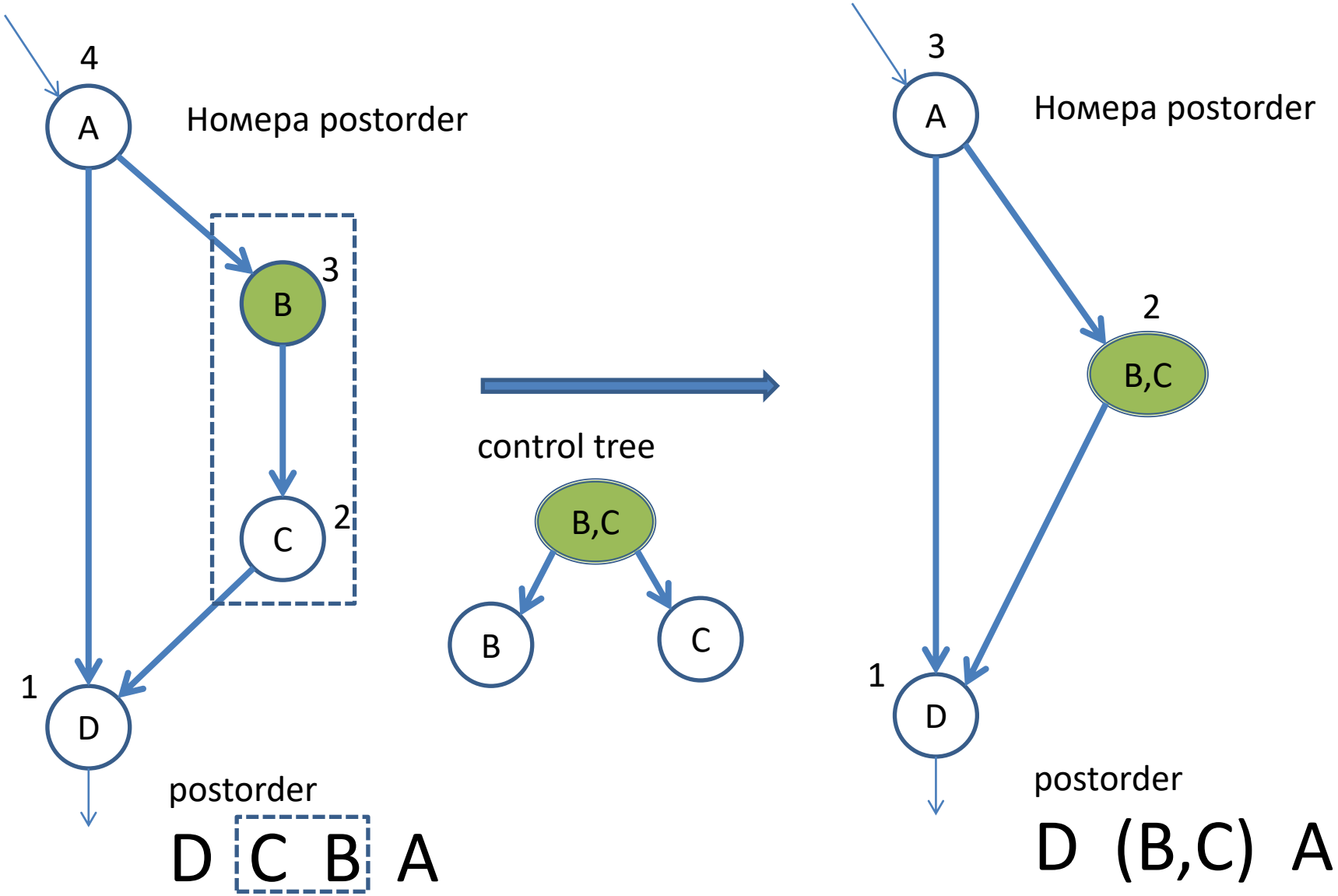
Этапы анализа



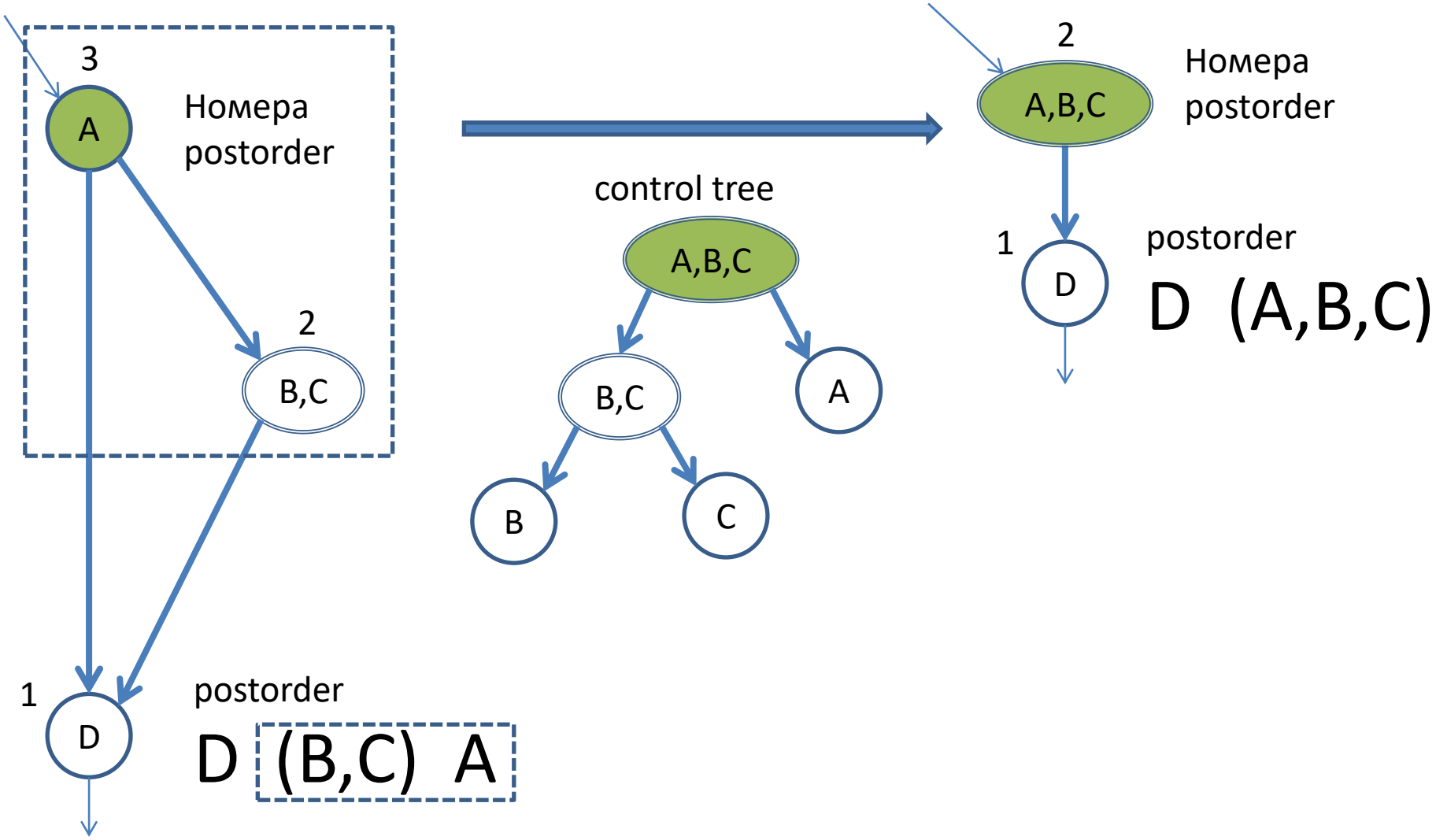

Control tree, который соответствует этим этапам



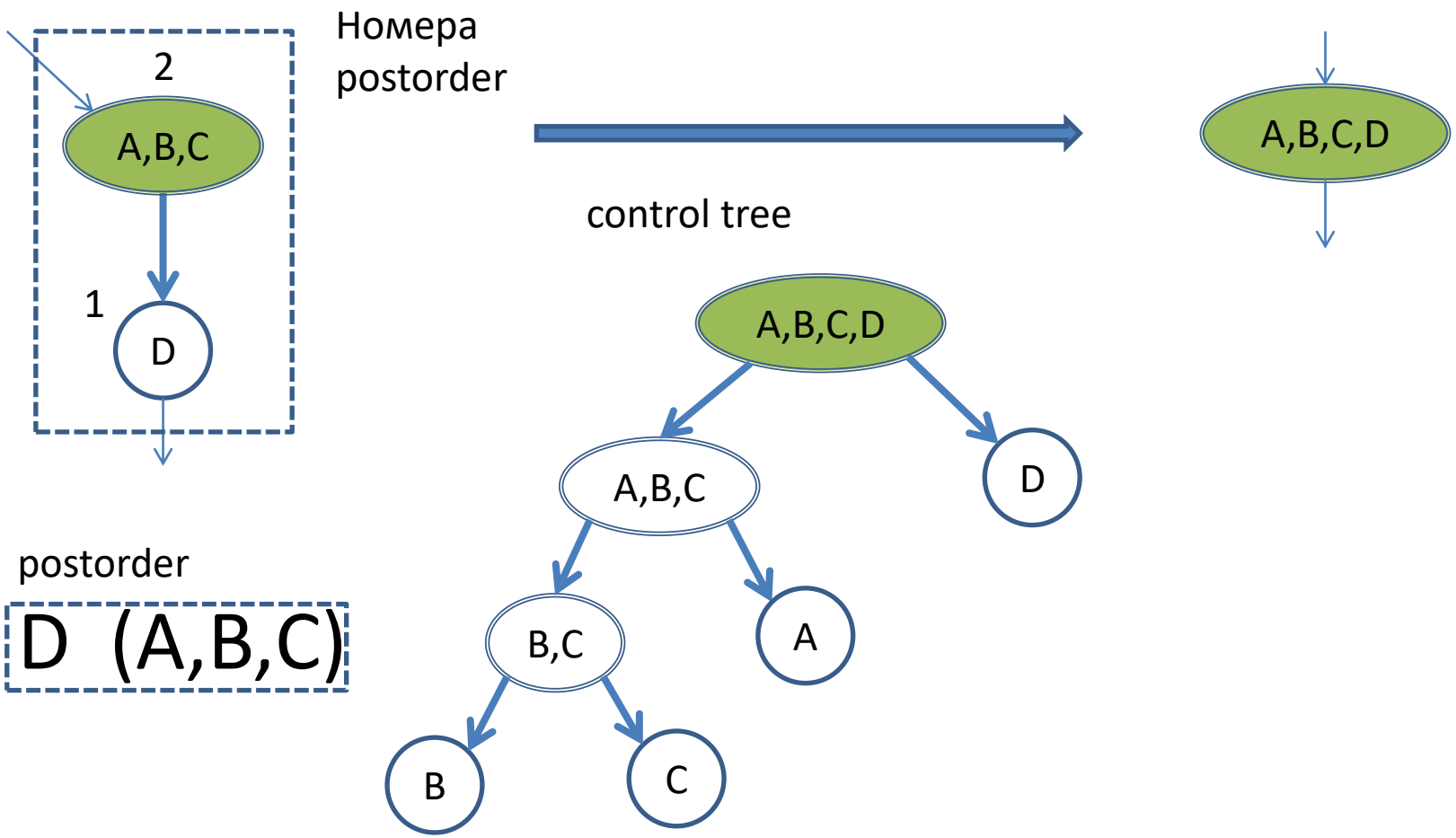
# Структурный CFA



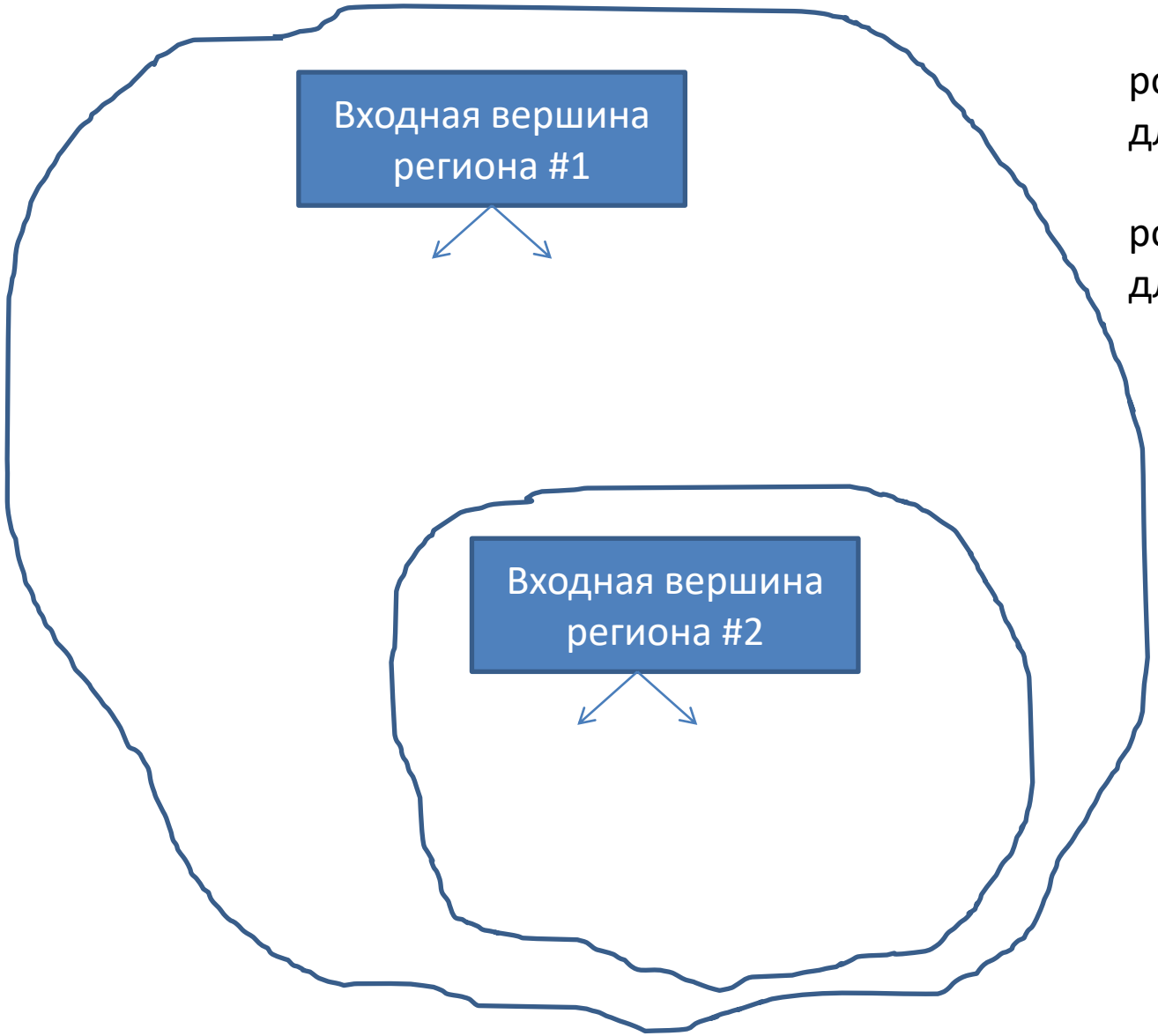
# Структурный CFA



# Структурный CFA



# Структурный CFA




post order номер для #1

>

post order номер для #2

# Структурный CFA

- Входной узел региона доминирует над всеми узлами этого региона
- Входной узел имеет максимальный номер post order среди всех узлов, которые входят в этот регион

Алгоритм: структурный CFA
Вход: g CFG граф
Выход: c CT дерево

```
Vertex Post[ ];
set<Vertex> Visit;
int PostMax;
Vertex n, p;
set<Vertex> NodeSet;

set<CtVertex> ctNodes;
set<CtEdge> ctEdge;

enum ERegionType
{
    UNKNOWN_REGION,
    BLOCK,
    IF_THEN_ELSE,
    SELF_LOOP,
    WHILE_LOOP,
    NATURAL_LOOP
};

void DFS_Postorder(Vertex v);
```

Для простоты мы опустим регионы типа **if/then**, **case/switch**, **неприводимые циклы**.

Будем считать, что их нет в CFG

```
do
{
    PostMax = 0;
    PostCtr = 0;
    Post.clear();
    Visit.clear();
    DFS_Postorder(g.entry);
    INTERNAL_LOOP;  // macros
}
while( AllVert(g).count() != 1 );
```

```
#define INTERNAL_LOOP
    while(AllVert(g).count() > 1 && PostCtr <= PostMax) :
        n = Post[PostCtr];
        rtype = Acyclic_Region_Type(n, NodeSet);
        if ( rtype != ERegionType.UNKNOWN_REGION ) :
            p = Reduce(rtype, NodeSet);
            if ( NodeSet.In(g.entry) ):
                g.entry = p;
        else:
            ReachUnder = GetReachUnder(g, n);
            rtype = Cyclic_Region_Type(n, NodeSet);
            if ( rtype != ERegionType.UNKNOWN_REGION ) :
                p = Reduce(rtype, ReachUnder);
                if (ReachUnder.In(g.entry) )
                    g.entry = p;
            else
                PostCtr ++;
```

```
void DFS_Postorder(Vertex v) // вычисление PostOrder DFS
{
    Visit += v;
    foreach(Vertex y in Succ(v))
    {
        if ( !Visit.In(v) )
            DFS_Postorder(v);
    }
    PostMax ++;
    Post[PostMax-1] = v;
}

set<Vertex> GetReachUnder(CFG g, Vertex n)
{
    set<Vertex> res;
    res += n;
    for (Vertex v in AllVertex(g))
        if (Path_Back(v, n))
            res += m;
    return res;
}
```

Path_Back(v, n) возвращает True, ∃ вершина k |

- ∃ путь (возможно пустой) от v до k, который не проходит через n
- k -> n – обратная дуга

```
ERegionType Acyclic_Region_Type(Vertex node, set<Vertex> nset)
{
      Vertex m, n;
      bool p, s;
      nset.clear();

      // тип региона BLOCK
     RECOGNIZE_BLOCK;

     if ( nset.count() >= 2 )
        return ERegionType.BLOCK;
      // тип региона IF_THEN_ELSE
      else if ( Succ(node).count() == 2 )
      {
            RECOGNIZE_ IF_THEN_ELSE;
      }
}
```

```
#define RECOGNIZE_BLOCK
    n = node; p = true; s = ( Succ(n).count() == 1 );
    while (p && s)
    {
          nset += n; n = Succ(n).get_item();
          p = (Pred(n).count() == 1); s = (Succ(n).count() == 1);
    }
    if (p)
       nset += n;
    n = node; p = (Pred(n) == 1); s = true;
    while (p && s)
    {
         nset += n; n = Pred(n).get_item();
         p = (Pred(n).count() == 1); s = (Succ(n) == 1);
    }
    if (s)
       nset += n;
    node = n;
```

Распознавание региона block.
Сохраняем узлы, которые относятся к региону, в nset

```
#define RECOGNIZE_ IF_THEN_ELSE
    m = Succ(n).get_item();
    n = ( Succ(n) – m ).get_item();
    if (
          Succ(n) == Succ(m) &&
       Succ(m).count() == 1 &&
       Pred(m).count() == 1 &&
       Pred(n).count() == 1
     )
    {
          nset  += node;
          nset += m;
          nset += n;
          return ERegionType.IF_THEN_ELSE;
    }
    else // другие типы регионов
```

Распознавание региона if-then-else
Сохраняем узлы, которые относятся к региону, в nset

```
ERegionType Cyclic_Region_Type(Vertex node, set<Vertex> nset)
{
     if ( nset.count() == 1 )
          if ( “существуе дуга node -> node” )
                    return ERegionType.SELF_LOOP;
          else
                    return ERegionType.UNKNOWN_REGION;
     Vertex m = ( nset - node ).get_item();
     if ( Succ(node).count()==2 && Succ(m).count() == 1 &&
          Pred(node).count()==2 && Pred(m).count() == 1 )
          return ERegionType.WHILE_LOOP;
     else
          return ERegionType. NATURAL_LOOP;
}
```

Распознавание региона-цикла.
Сохраняем узлы, которые относятся к региону, в nset

```
void Reduce(
                CFG g,
                ERegionType rtype,
                set<Vertex> nset)
{
    // создание нового узла для CFG
    Vertex node = CreateNode();
    Replace(g, node, nset, rtype);
    return node;
}
```

```
void Replace(CFG g,  Vertex node, set<Vertex> nset, ERegionType rtype)
{
    CtVertex ctNode = CreateCtNode(rtype);
    Compact(g, node, nset);
    foreach(edge e in AllEdge(g))
      if ( nset.In(e.v1) || nset.In(e.v2) )
      {
          AllEdge(g) -= e;
          Succ(e.v1)  -= e.v2;
          Pred(e.v2)  -= e.v1;
          if ( AllVertex(g).In(e.v1)  || e.v1 != node )
          {
              AllEdge(g) += "e.v1 -> node"; Succ(e.v1) += node;
          }
          else if ( AllVertex(g).In(e.v2)  || e.v2 != node )
          {
              AllEdge(g) += "node -> e.v2"; Pred(e.v2) += node;
          }
      }
   ctNodes += ctNode;
   foreach( Vertex v in nset )
     ctEdges += "ctNodes -> v";
}
```

Замена региона nset в CFG на узел node

Добавление узла в Control Tree

Compact

- добавляем node в AllVertex(g)
- ищем максимальный номер позиции MAX в Post, которые имеют вершины nset
- вставляем в позицию MAX node в Post
- удаляем вершины nset из AllVertex(g)
- удаляем вершины nset из Post
- PostCtr устанавливаем индексу, который соответствует node в Post
- Устанавливаем PostMax

# Список литературы


- Компиляторы. Принципы, технологии и инструментарий. 2-е изд. А. Ахо (DFS, классификация дуг)
- Компиляторы. Принципы, технологии и инструментарий. 1-е изд. А. Ахо (T1-T2 анализ)
- Advanced compiler design & implementation. S. Muchnik (структурный анализ)