Анализ алиасов

Савченко Валерий

24 марта 2014 г.

Содержание

Аннотация

Текст аннотации

1 Введение

Начиная говорить о такой обширной теме, как анализ алиасов, стоит прежде всего прояснить что же такое алиас и почему их так важно анализировать.

1.1 Определение

Определение 1 (Алиас)**.** *Алиасом* [1] будем называть ситуацию, в которой доступ к одному и тому же участку памяти может осуществляться под разными именами.

Пример 1. В данном примере [2] $*p$ и $*q$ являются алиасом.

```
A *p, *q;
p = new A();
q = p;
```

Листинг 1: Пример алиаса

1.2 Природа алиасов

Однако, стоит также прояснить почему в коде программ могут возникать алиасы.
Алиасы могут быть получены при использовании следующих сущностей (для лучшего понимания каждая категория будет иллюстрироваться наглядным примером).

1.2.1 Указатели

Самым очевидным способом получения алиасов, конечно же, являются указатели.
Два (и более) различных указателя могут указывать на одну область в памяти, образуя при этом алиас. Или же указатель может указывать на область памяти, в которой расположена некоторая переменная и быть,

[1] В отечественной литературе алиасы также называют *псевдонимами* или *синонимами*

[2] Здесь и далее в листингах будет использоваться язык C/C++, так как для него наиболее остро стоит проблема анализа алиасов

соответственно, ее алиасом.
Пример 2. В качестве примера алиаса на указателях можно взять уже упомянутый пример1, а также добавить еще один.

```
int x, *y;
y = &x;
```

Листинг 2: Пример алиаса на указателях

В данном случае x и $*y$ - алиас.

1.2.2 Передача параметров по ссылке

В случае получения параметров по ссылке можно также получить и парочку алиасов аргументов функции, как между собой, так и с глобальными переменными.

Пример 3. В данном примере алиасами являются x и глобальная переменная a, а также аргументы y, z и локальная переменная b.

```
int a;
void foo(int &x, int &y, int &z) {
  . . .
}
int main() {
  . . .
  int b;
  foo(a, b, b);
  . . .
}
```

Листинг 3: Пример алиаса при передаче параметров по ссылке

1.2.3 Индексация массива

Если говорить о массивах, то разные индексные выражения (в особенности в циклах) могут привести к равенству индексов и, следовательно, к обращению к той же ячейке памяти (т.е. к появлению алиаса).

Пример 4. В данном примере индексные выражения i и $9 - i$ равны при $i = 5$, т.е. при $i = 5$ $a[i]$ и $a[9 - i]$ являются алиасом.

```
int a[10];
. . .
for (int i = 0; i <= 9; ++i) {
  a[i] = a[9 - i] + 3;
}
```

Листинг 4: Пример алиаса, порожденного различными индексными выражениями

1.2.4 Полиморфные структуры данных

В данном случае (в контексте языка C) стоит говорить о *union*. Данная структура по своей природе подразумевает алиас.

Пример 5. В данном примере алиасом являются $x.a$ и $x.b$.

```
union {
  int a;
  double b;
} x;
```

Листинг 5: Пример алиаса при использовании union

Стоит отметить, что алиасы могут быть получены и любой комбинацией из рассмотренных выше способов.

Однако, несмотря на такое количество возможностей появления алиаса, очень часто можно точно сказать об отсутствии алиаса.

1.2.5 Случаи гарантированного отсутствия алиаса

Пример 6.

```
void foo() {
  . . .
  int i, j;
  . . .
}
```

Листинг 6: Статически аллоцированные объекты

Пример 7.

```
void bar() {
  . . .
  p = new TreeNode();
  p->left = new TreeNode();
  p->right = new TreeNode();
  . . .
}
```

Листинг 7: Динамически аллоцированные объекты

В данных примерах можно утверждать, что i и j, а также p, $p \to left$ и $p \to right$ гарантированно не являются алиасами.

1.3 Важность анализа алиасов

Перед формальным введением в анализ алиас желательно понять зачем его необходимо проводить и почему данная задача столь востребована.

Прежде всего анализ алиасов имеет особую важность для *компиляторных оптимизаций*, т.к. может напрямую влиять на результаты их работы. Для более ясного представления такого рода эффектов будет рассмотрено несколько распространенных оптимизаций и примеров влияния алиасов на них.

1.3.1 Устранение общих подвыражений

Пример 8. В рассматриваемом примере оптимизация *устранения общих подвыражений* [1] могла бы удалить общее подвыражение $a + b$ и получить код, который вычисляет его единожды[листинг 9].

```
*p = a + b;
c = a + b;
```

Листинг 8: До оптимизации

```
t = a + b;
*p = t;
c = t;
```

Листинг 9: После оптимизации

Однако, данная оптимизация может быть проведена *только* в том случае, когда известно, что $*p$ не является алиасом ни для a, ни для b.

1.3.2 Перемещение кода инвариантного относительного цикла

Пример 9. В рассматриваемом примере оптимизация *перемещения кода* [2] могла бы переместить подвыражение $*p + a$ за границу цикла и получить код[листинг 11], не вычисляющий одно и то же выражение на каждой итерации.

```
while (--i >= 0) {
  c[i] = *p + a;
}
```

Листинг 10: До оптимизации

```
t = *p + a;
while (--i >= 0) {
  c[i] = t;
}
```

Листинг 11: После оптимизации

Однако, это может быть сделано *только* в том случае, когда известно, что $\forall i \in [0, n)$ верно, что $*p$ и $c[i]$ не являются алиасом.

[1] Устранение общих подвыражений - компиляторная оптимизация, которая ищет в программе одинаковые вычисления и удаляет второе и последующие вхождения рассматриваемого выражения, заменяя его уже вычисленным значением

[2] Перемещение кода инвариантного относительно цикла - компиляторная оптимизация, которая перемещает за пределы цикла код, перемещение которого никак не повлияет на семантику цикла и программы в целом

1.3.3 Распространение констант

Пример 10. В рассматриваемом примере оптимизация *распространения констант* [1] могла бы распространить значение константной переменной x и получить код[листинг 13].

```
x = 3;
*p = 4;
y = x;
```

Листинг 12: До оптимизации

```
x = 3;
*p = 4;
y = 3;
```

Листинг 13: После оптимизации

Однако, это может быть сделано *только* в том случае, когда известно, что $*p$ и x не являются алиасом.

1.3.4 Распределение регистров

В случае *распределения регистров* [2] анализ алиасов является просто необходимым условием для того, чтобы переменной в принципе был бы сопоставлен некоторый регистр. То есть на время нахождения переменной в регистре необходимо быть уверенным, что не происходит записи/чтения данной переменной через ее алиас.

Пример 11. В рассматриваемом примере при распределении регистров переменная x на время цикла могла быть размещена на регистре, что значительно сократило бы число обращений к памяти.

```
x = 0;
. . .
while (--i >= 0) {
  *p += a[i];
  x += a[i];
}
```

Листинг 14: Распределение регистров

Однако, если $*p$ является алиасом для x, на выходе из цикла (после оптимизации) $x = \sum_{i=0}^{n-1} a[i]$, хотя без размещения на регистре было бы $x = 2 * \sum_{i=0}^{n-1} a[i]$, что, очевидным образом, является некорректным поведением программы.

[1] Распространение констант - компиляторная оптимизация, уменьшающая число избыточных вычислений заменой константных выражений и переменных их значениями

[2] Распределение регистров - компиляторная оптимизация, состоящая в отображении множества переменных программы в множество регистров целевого процессора

Рассмотренные оптимизации не покрывают всего спектра проблем, однако, хорошо демонстрируют тот класс проблем, который может быть вызван незнанием набора алиасов программы, а также возможные масштабы упущенной выгоды.

2 Постановка задачи

Теперь стоит сказать собственно о самой задаче анализа алиасов.

2.1 Определение

Пусть $\mathcal{V}$ - множество переменных в программе, а $\mathcal{D}$ - множество решений. Чаще всего выбирают $\mathcal{D}$ следующего вида: $|\mathcal{D}| = 3, \mathcal{D} = \{must, may, never\}$, где[1] :

- $must$ отвечает тому, что две переменные **точно** являются алиасом, т.е. изменение одной *всегда* влечет за собой изменение второй переменной
- may отвечает тому, что две переменные **могут** быть алиасом, т.е. изменение одной *может* повлечь за собой изменение второй переменной, а может и нет
- $never$ отвечает тому, что две переменные **точно** не являются алиасами, т.е. изменение одной *никогда* не влечет за собой изменение второй переменной

Определение 2 (Анализ алиасов)**.** *Анализом алиасов* назовем отображение $\varphi : \mathcal{V} \times \mathcal{V} \to \mathcal{D}$ такое, что $\forall a, b \in \mathcal{V} \to \varphi(a, b) = \varphi(b, a)$.

То есть, анализ алиасов есть сопоставление каждой паре переменных некоторого решения о том, могут ли данные переменные быть алиасом или нет.

2.2 Сравнение

Введем также сравнение [2] различных анализов алиасов.

Определение 3. Пусть φ и ψ - анализы алиасов. Тогда будем говорить, что φ **слабее** ψ и обозначать $\varphi \leq \psi$, если

$$\forall v_1, v_2 \in \mathcal{V} \to \varphi(v_1, v_2) = \begin{cases} must, & \text{только если } \psi(v_1, v_2) = must \\ never, & \text{только если } \psi(v_1, v_2) = never \\ may, & \text{при любых значениях } \psi(v_1, v_2) \end{cases}$$

[1] По сути своей такое множество решений соотвтетсвует *трехзначной логике*

[2] Не всякие два анализа алиасов можно сравнить при помощи данной операции отношения

Определение 4. Будем говорить, что φ **строго слабее** ψ и обозначать $\varphi < \psi$, если $\varphi \leq \psi$ и $\exists v, w \in \mathcal{V} : \psi(v, w) \in \{must, never\}, \varphi(v, w) = may$.

То есть, если $\varphi \leq \psi$ и для некоторых $v, w \in \mathcal{V}$ выполняется $\varphi(v, w) = must$, то $\psi(v, w) = must$. Обратное, вообще говоря, не верно. Аналогичное верно и для $never$.

Утверждение 1. *Введенная операция сравнения обладает свойством транзитивности, то есть, если φ, ψ и χ - анализы алиасов и $\varphi \leq \psi$, $\psi \leq \chi$, то верно, что $\varphi \leq \chi$.*

Доказательство. Рассмотрим все $v, w \in \mathcal{V} : \varphi(v, w) = must$. Так как $\varphi \leq \psi$, то $\psi(v, w) = must$. Так как $\psi \leq \chi$, то $\chi(v, w) = must$. Получили, что если $\varphi(v, w) = must$, то $\chi(v, w) = must$. Аналогично можно получить, что если $\varphi(v, w) = never$, то $\chi(v, w) = never$. То есть $\varphi \leq \chi$. □

Аналогично доказывается транзитивность строгого сравнения.

2.3 Консервативность

Как вы могли догадаться, если рассматривать все такие отображения, толку от этого будет, мягко говоря, никакого. Анализ алиасов, естественно, *должен* быть связан с семантикой конкретной программы.

Для каждой программы существует анализ алиасов, который наиболее точно описывает конкретно ее. Такой анализ алиасов будем называть *точным* и обозначать ϕ^*.

Определение 5. Анализ алиасов φ будем называть *консервативным*, если $\varphi \leq \phi^*$.

Утверждение 2. *Пусть φ и ψ - анализы алиасов. Тогда, если φ консервативен и $\psi \leq \varphi$, то ψ консервативен.*

Доказательство. Доказательство непосредственно следует из транзитивности сравнения: $\psi \leq \varphi, \varphi \leq \phi^* \Rightarrow \psi \leq \phi^*$ □

При этом существует очевидный анализ алиасов, являющийся при этом консервативным.

Определение 6. Будем называть *тривиальным* и обозначать ϕ^- анализ алиасов такой, что

$$\forall v, w \in \mathcal{V} \to \phi^-(v, w) = may$$

Если говорить об анализе алиасов в контексте компиляторных оптимизаций, то имеет смысл рассматривать лишь консервативные отображения, т.е. задача состоит в поиске $\varphi : \phi^- \leq \varphi \leq \phi^*$.

2.4 Композиция

Определение 7 (Уточнение)**.** На множестве решений $\mathcal{D}$ операцией *уточнения* назовем бинарную операцию $\times$ со следующей таблицей истинности [1] :

$\times$	never	must	may
never	never	-	never
must	-	must	must
may	never	must	may

Операция уточнения является коммутативной и ассоциативной, что легко устанавливается из таблицы истинности.

Определение 8 (Композиция)**.** *Композицией* анализов алиасов φ и ψ будем называть следующий операцию:

$$\chi = \varphi \circ \psi \Leftrightarrow \forall v, w \in \mathcal{V} \to \chi(v, w) = \varphi(v, w) \times \psi(v, w)$$

Из коммутативности и ассоциативности уточнения непосредственно следует и коммутативность и ассоциативность композиции.

Утверждение 3. *Пусть φ и ψ - анализы алиасов, $\chi = \varphi \circ \psi$ и $N = \{(v, w) \in \mathcal{V} \times \mathcal{V} : \varphi(v, w), \psi(v, w) \in \{must, never\}$ и $\varphi(v, w) \neq \psi(v, w)\}$, тогда на множестве $\overline{N}$ χ является анализом алиасов.*

Доказательство. $\forall (a, b) \in \overline{N}$ рассмотрим $\chi(a, b) = \varphi(a, b) \times \psi(a, b) =$ {так как φ и ψ - анализы алиасов} $= \varphi(b, a) \times \psi(b, a) =$ {так как $(a, b) \in \overline{N}$, то операция уточнения в данном случае определена } $= \chi(b, a)$ $\square$

2.5 Теорема о композиции консервативных анализов алиасов

Теорема 1 (О композиции консервативных анализов алиасов)**.** *Пусть φ и ψ - консервативные анализы алиасов, тогда $\chi = \varphi \circ \psi$ является определенным на всем $\mathcal{V} \times \mathcal{V}$ консервативным анализом алиасов, причем $\varphi \leq \chi$ и $\psi \leq \chi$.*

Доказательство. Разобьем доказательство на три части: определенность χ на всем $\mathcal{V} \times \mathcal{V}$, консервативность χ и выполнение $\varphi \leq \chi$ и $\psi \leq \chi$.

1. Докажем от противного. Пусть χ определен не везде, тогда

   $$\exists v, w \in \mathcal{V} : \varphi(v, w), \psi(v, w) \in \{must, never\} \text{ и } \varphi(v, w) \neq \psi(v, w)$$

   Без ограничения общности можно считать, что $\varphi(v, w) = never$ и $\psi(v, w) = must$. Так как φ консервативен, то $\varphi(v, w) = never \Rightarrow \phi^*(v, w) = never$. Однако, ψ тоже консервативен и аналогичным образом получаем, что $\phi^*(v, w) = must$. Пришли к противоречию.

[1] Прочерк означает, что на этой паре операция не определена

2. Докажем от противного. Пусть χ неконсервативен, тогда

$$\exists v, w \in \mathcal{V} : \chi(v, w) \in \{must, never\} \text{ и } \phi^*(v, w) \neq \chi(v, w)$$

Без ограничения общности можно считать, что $\chi(v, w) = never$ и $\phi^*(v, w) = must$. Так как $\chi = \varphi \circ \psi$, то $\varphi(v, w) \times \psi(v, w) = never$. Откуда следует, что как минимум одно из $\varphi(v, w)$ и $\psi(v, w)$ равняется $never$. Для определенности будем считать, что $\varphi(v, w) = never$. Так как φ консервативен, то $\varphi(v, w) = never \Rightarrow \phi^*(v, w) = never$. Однако, выше уже было определено, что $\phi^*(v, w) = must$. Пришли к противоречию.

3. Так как композиция коммутативна, то достаточно доказать, что $\varphi \leq \chi$. Покажем, что

$$\forall v, w \in \mathcal{V} : \varphi(v, w) \in \{must, never\} \Rightarrow \chi(v, w) = \varphi(v, w)$$

Возьмем $v, w : \varphi(v, w) = never$, тогда $\chi(v, w) = never \times \psi(v, w)$. Так как χ определен на всем $\mathcal{V} \times \mathcal{V}$, то $\psi(v, w) \in \{never, may\}$. Тогда $\chi(v, w) = never$. Аналогично показывается, что для $v, w : \varphi(v, w) = must$ верно, что $\chi(v, w) = must$. Откуда по определению $\varphi \leq \chi$.

□

Дополнение 1. *Если в условиях теоремы о композиции консервативных анализов алиасов $\varphi \leq \psi$, то $\chi = \psi$.*

Доказательство. Рассмотрим произвольные $v, w \in \mathcal{V}$.

Пусть $\psi(v, w) = never$. Так как $\psi \leq \chi$, то $\chi(v, w) = never$. Аналогично показывается, что если $\psi(v, w) = must$, то $\chi(v, w) = must$.

Пусть $\psi(v, w) = may$. Так как из теоремы $\varphi \leq \psi$, то $\varphi(v, w) = may$. Тогда $\chi(v, w) = \varphi(v, w) \times \psi(v, w) = may \times may = may = \psi(v, w)$. Откуда получаем, что значения χ и ψ для произвольных v, w совпадают, то есть $\chi = \psi$. □

Дополнение 2. *Если в условиях теоремы о композиции консервативных анализов алиасов φ и ψ несравнимы, то $\varphi < \chi$ и $\psi < \chi$.*

Доказательство. Если φ и ψ несравнимы, то

$$\exists v_1, w_1 \in \mathcal{V} : \varphi(v_1, w_1) = may, \text{ а } \psi(v_1, w_1) \in \{must, never\}$$

$$\exists v_2, w_2 \in \mathcal{V} : \psi(v_2, w_2) = may, \text{ а } \varphi(v_2, w_2) \in \{must, never\}$$

Для определенности будем считать, что $\psi(v_1, w_1) = \varphi(v_2, w_2) = must$. Рассмотрим значения $\chi(v_1, w_1)$ и $\chi(v_2, w_2)$:

$$\chi(v_1, w_1) = \varphi(v_1, w_1) \times \psi(v_1, w_1) = may \times must = must$$

$$\chi(v_2, w_2) = \varphi(v_2, w_2) \times \psi(v_2, w_2) = must \times may = must$$

Из теоремы $\varphi \leq \psi$. Учитывая также, что $\exists v_1, w_1 \in \mathcal{V} : \varphi(v_1, w_1) = may$, а $\chi(v_1, w_1) = must$, по определению получаем, что $\varphi < \chi$.
Аналогично показываем, что $\psi < \chi$. □

2.6 Простейший нетривиальный анализ алиасов

Данный подход является первым подходом к анализу алиасов, который может предоставлять хоть какую-то *полезную* информацию.

Перед тем, как представить собственно анализ алиасов, введем несколько обозначений.

Пусть $\mathcal{G} \subseteq \mathcal{V}$ - множество глобальных переменных, а
$\mathcal{R} = \{v : v \in \mathcal{V}, \exists p \in \mathcal{V} : v = *p\}$ - множество разыменований указателей (множество значений v из $\mathcal{V}$ таких, что существует переменная p из $\mathcal{V}$, разыменованием которой и является v).

Обращаясь теперь к коду целевой программы, рассмотрим все $v \in \mathcal{V}$ такие, что подвергались операции взятия адреса, т.е. в коде программы существует инструкция, в которую входит $\&v$. Множество таких переменных обозначим $\mathcal{A}$.

Анализом алиасов *по взятию адреса* (address-taken) назовем следующий анализ алиасов:

$$\varphi^{\mathcal{A}}(v, w) = \begin{cases} may, & \text{если } v, w \in \mathcal{R} \\ may, & \text{если } v \in \mathcal{R}, w \in \mathcal{A} \cup \mathcal{G} \\ never, & \text{в остальных случаях} \end{cases}$$

Пример 12. Построим $\varphi^{\mathcal{A}}$ для данного примера.

Для этого построим множества $\mathcal{V}, \mathcal{G}, \mathcal{R}$ и $\mathcal{A}$.

```
int c;

void foo() {
  int a, b, *p, *q;
  p = &a;
  a = 3, b = 4;
  . . .
  *q = 5;
}
```

$\mathcal{V} = \{a, b, c, p, q, *p, *q, \dots\}$.

$\mathcal{G} = \{c\}$, т.к. переменная c - глобальная.

$\mathcal{A} = \{a\}$, т.к. в коде программы присутствует инструкция `p = &a`.

$\mathcal{R} = \{*p, *q\}$, т.к. $p, q \in \mathcal{V}$.

Тогда $\varphi^{\mathcal{A}}(*p, *q) = \varphi^{\mathcal{A}}(*p, a) = \varphi^{\mathcal{A}}(*q, a) = \varphi^{\mathcal{A}}(*p, c) = \varphi^{\mathcal{A}}(*q, c) = may$. Для всех остальных значений из $\mathcal{V} \times \mathcal{V}$ функция $\varphi^{\mathcal{A}}$ равна $never$.

В данном случае, анализ алиасов позволит понять, что у переменной b нет алиасов, чего не мог дать тривиальный подход.

3 Виды анализов

Отталкиваться от тривиального решения для поиска подходящего (как это делалось в случае анализа алиасов по взятию адреса) не является оптимальным подходом. Поэтому будем пробовать эффективно считать анализы алиасов наиболее приближенные к ϕ^*, нежели к ϕ^-. Для этого попробуем классифицировать все многообразие анализов алиасов и решить уже более узко в каком именно классе мы будем работать.

3.1 Основные виды

Здесь будут перечислены и описаны [1] основные классы анализов алиасов. Данные термины общеприняты и используются при общем описании любого анализа алиасов.

3.1.1 Межпроцедурность

Различают *межпроцедурные* и *внутрипроцедурные* анализы алиасов, которые различаются тем, что первые учитывают взаимодействия между различными функциями в программе и используют информацию сразу о всех функциях в программе, а вторые используют информацию лишь одной функции.

Если говорить о внутрипроцедурном анализе, то формально его можно представить следующим образом.

Представим множество переменных $\mathcal{V}$ в виде $\mathcal{G} \cup \mathcal{V}_1 \cup \mathcal{V}_2 \cup \cdots \cup \mathcal{V}_m$, где m - число функций в рассматриваемой программе, а $\mathcal{V}_i$ - множество переменных i-ой функции, причем $\forall i, j \in [1, m] : i \neq j \to \mathcal{V}_i \cap \mathcal{V}_j = \emptyset$. Тогда внутрипроцедурным анализом алиасов для i-ой функции является анализ алиасов $\varphi_i : \forall v, w \in \bigcup\limits_{\substack{j=1 \\ j \neq i}}^{m} \mathcal{V}_j \to \varphi_i(v, w) = may$, а анализ алиасов φ для всей программы на основании внутрипроцедурных анализов φ_i может быть получен с помощью композиции: $\varphi = \varphi_1 \circ \varphi_2 \circ \cdots \circ \varphi_m$. Если потребовать консервативность всех φ_i, то, согласно теореме о композиции, φ будет консервативен и определен на всех парах переменных.

Но, даже если рассмотреть $\phi_i^* : \nexists \varphi_i' : \phi_i^* < \varphi_i' \leq \phi^*$ (наиболее точные внутрипроцедурные анализы алиасов), то несмотря на то, что в общем

[1] При чтении не зацикливайтесь на чтении определения вида, если не можете понять о чем идет речь, посмотрите на пример.

случае $\phi_1^* \circ \phi_2^* \circ \cdots \circ \phi_m^* \leq \phi^*$, на практике, в подавляющем числе случаев, стоит говорить о строгом неравенстве.

Пример 13. Пример межпроцедурного и внутрипроцедурного анализов

```
void foo(int &a, int &b) {
  int *p, *q;
  p = &a;
  q = &b;
  . . .
}

void bar() {
  int x;
  foo(x, x);
}
```

В данном случае $\mathcal{V}_1 = \{a, b, p, q, *p, *q\}, \mathcal{V}_2 = \{x\}$ и $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$. Рассмотрим [1] для начала ϕ^*:

1. Из-за вызова `foo(x,x)` получаем $\phi^*(x, a) = \phi^*(x, b) = must$
2. Второе свойство анализов алиасов дает нам, что $\phi^*(a, b) = must$.
3. Из инстуркций `p = &a` и `q = &b` устанавливаем, что $\phi^*(*p, a) = \phi^*(*q, b) = must$
4. Опять же из второго свойства имеем $\phi^*(x, *p) = \phi^*(x, *q) = must$
5. Для остальных пар вида (p, v) и (q, v), где $v \in \mathcal{V}$ верно, что $\phi^*(p, v) = \phi^*(q, v) = never$

Очевидно, что ϕ^* - межпроцедурный, т.к. даже в пояснении использовались инструкции и переменные из двух различных функций. Теперь попробуем рассмотреть ϕ_1^* и ϕ_2^*. Начнем с ϕ_1^*:

1. Из инстуркций `p = &a` и `q = &b` устанавливаем, что $\phi_1^*(*p, a) = \phi_1^*(*q, b) = must$
2. Так как не может использоваться информация из других функций, то мы не имеем никакой информации о том, с какими параметрами могла вызываться рассматриваемая функция. Поэтому для сохранения свойства консервативности приходится считать, что параметры могут быть алиасами для всего вне рассматриваемой функции, т.е. в данном случае $\phi_1^*(a, x) = \phi_1^*(b, x) = may$.
3. Из второго свойства получаем, что $\phi_1^*(*q, x) = \phi_1^*(*p, x) = ... = may$

ϕ_2^* в данном случае совпадает с ϕ^-. И как результат имеем, что даже в таком простом случае $\phi_1^* \circ \phi_2^* = \phi_1^* < \phi^*$.

[1] Здесь не представлен алгоритм построения ϕ^*, а лишь пояснения относительно его вида

3.1.2 Контекстная чувствительность

Различают *контекстно-чувствительные* и *контекстно-нечувствительные* анализы, понимая под этим, что при анализе функции и ее переменных учитывается или нет контекст, в котором была вызвана функция.

Пример 14. Пример контекстно-чувствительного и контекстно-нечувствительного анализов

3.1.3 Потоковая чувствительность

3.1.4 Поле-чувствительность

3.1.5 Чувствительность к типам

3.2 Дополнительные различия

3.2.1 Модели памяти

3.2.2 Необходимость полного кода программы

3.2.3 Представление алиасов

3.2.4 Представление агрегатов

Если же говорить о реальных приложениях, то построение точного решения даже в классе контекстно- и потоково-нечувствительных анализов является NP-трудной задачей, что делает ее неприменимой на практике. Однако, существует несколько методов построения консервативных анализов алиасов в этом классе, которые значительно лучше анализа алиасов по взятию адреса.

4 Анализ указателей

Определяющим фактором сложности анализа алиасов, несмотря на наличие и других способов, остаются указатели. Разрешение алиасов, полученных другими способами, не является трудоемкой задачей, соответственно, главной задачей становится *анализ указателей*.

4.1 Определение

Пусть $\mathcal{P}$ - множество указателей программы. Отметим, что $\mathcal{P} \subseteq \mathcal{V}$.

Определение 9 (Анализ указателей)**.** *Анализом указателей* назовем отображение $\xi : \mathcal{P} \to 2^{\mathcal{V}}$.

То есть анализ указателей есть сопоставление каждому указателю подмножества переменных (множество переменных, на которые может указывать указатель).

4.2 Построение анализа алиасов по анализу указателей

Так как мы говорим об анализе указателей в контексте анализа алиасов, то необходимо уметь строить из первого второе.

Построим анализ алиасов φ_ξ из анализа указателей ξ:

1. За основу возьмем тривиальный анализ алиасов: $\varphi_\xi = \phi^-$
2. $\forall p \in \mathcal{P}, \forall a \notin \xi(p) \to \varphi_\xi(*p, a) = \varphi_\xi(a, *p) = never$
3. В третьем пункте имеются некоторые расхождения:
   - Анализ указателей называют $must$-анализом, если на его основании можно предъявить решение $must$, то есть $\forall p \in \mathcal{P}, \forall a \in \mathcal{V}_p \to \varphi_\xi(*p, a) = \varphi_\xi(a, *p) = must$
   - Если же нет, то такой анализ называют may-анализом.

Далее, несмотря на то, что $must$-анализ предоставляет лучшие результаты, мы неявно будем иметь ввиду использование may-анализа, так как его построение проще и так как чаще всего важнее именно решение $never$, которое может быть получено на его основании.

Тогда для may-анализа указателей ξ можно сформулировать, что

$$\forall p \in \mathcal{P}, \forall a \in \mathcal{V} \to \varphi_\xi(*p, a) = never \Leftrightarrow a \notin \xi(p)$$

$$\forall a, b \in \mathcal{V} \to \varphi_\xi(a, b) \neq must$$

Далее постараемся перенести терминологию анализов алиасов на анализы указателей.

4.3 Сравнение

Введем сравнение анализов указателей.

Определение 10. Пусть ξ и ζ - анализы указателей. Тогда будем говорить, что ξ **слабее** ζ и обозначать $\xi \leq \zeta$, если $\forall p \in \mathcal{P} \to \zeta(p) \subseteq \xi(p)$.

Определение 11. Будем говорить, что ξ **строго слабее** ζ и обозначать $\xi < \zeta$, если $\xi \leq \zeta$ и $\exists q \in \mathcal{P} : \zeta(p) \subset \xi(p)$.

Лемма 1 (О сравнении анализов указателей)**.** *Пусть ξ и ζ - анализы указателей. Тогда*

$$\xi \leq \zeta \Leftrightarrow \varphi_\xi \leq \varphi_\zeta$$

Доказательство. 1. Пусть $\xi \leq \zeta$. Тогда $\forall p \in \mathcal{P} \to \xi(p) \supseteq \zeta(p)$.

Рассмотрим φ_ξ. Для него выполнятся, что

$$\forall q \in \mathcal{P}, \forall a \in \mathcal{V} \to \varphi_\xi(*q, a) = never \Leftrightarrow a \notin \xi(q)$$

Так как $\zeta(q) \subseteq \xi(q)$, то $a \notin \zeta(q) \Rightarrow \varphi_\zeta(*q, a) = never$. Учитывая, что $\forall a, b \in \mathcal{V} \to \varphi_\xi(a, b) \neq must$ (аналогичное верно и для φ_ζ), получаем $\varphi_\xi \leq \varphi_\zeta$.

2. Пусть $\varphi_\xi \leq \varphi_\zeta$.

   Так как φ_ξ и φ_ζ не принимают $must$ ни на каких парах переменных, то достаточно рассмотреть случай с равенством $never$. Из того, что $\varphi_\xi \leq \varphi_\zeta$ следует, что

   $$\forall q \in \mathcal{P}, \forall a \in \mathcal{V} \to \varphi_\xi(*q, a) = never \Rightarrow \varphi_\zeta(*q, a) = never$$

   Это же утверждение можно записать в следующем виде:

   $$a \notin \xi(q) \Rightarrow a \notin \zeta(q)$$

   Преобразовывая дальше получаем, что $a \in \overline{\xi(q)} \Rightarrow a \in \overline{\zeta(q)}$. Так как это верно для произвольного a, то $\overline{\zeta(q)} \supseteq \overline{\xi(q)}$. Откуда получаем, что $\xi(q) \supseteq \zeta(q)$, что в свою очередь верно для произвольного q. Тогда $\xi \leq \zeta$.

□

4.4 Консервативность

Определение 12. Анализ указателей ξ будем называть *консервативным*, если соответствующий ему анализ алиасов φ_ξ консервативен, т.е. $\varphi_\xi \leq \phi^*$.

Определение 13. Анализ указателей ρ^* будем называть *точным*, если ρ^* консервативен и $\nexists \xi : \xi$ консервативен и $\varphi_{\rho^*} < \varphi_\xi$.

Теорема 2. *(О существовании точного анализа указателей) Точный анализ указателей существует.*

Доказательство. Докажем существование построением такого анализа.

На основании ϕ^* построим анализ указателей ξ следующим образом:

$$\forall p \in \mathcal{P} \text{ выполняется, что } \forall v \in \mathcal{V} : \phi^*(*p, v) \neq never \Leftrightarrow v \in \xi(p)$$

Очевидно, что ξ консервативен по построению. Покажем, что $\xi = \rho^*$. Пусть это не так, тогда $\exists \zeta : \zeta$ консервативен и $\varphi_\xi < \varphi_\zeta$.

Так как $\forall v, w \in \mathcal{V} \to \varphi_\xi(v, w) \neq must$ и $\varphi_\zeta(v, w) \neq must$, а также $\varphi_\xi < \varphi_\zeta$, то

$$\exists a, b \in \mathcal{V} : \varphi_\zeta(a, b) = never, \varphi_\xi(a, b) = may$$

Так как φ_ζ построен по анализу указателей, то

$$\varphi_\zeta(a, b) = never \Leftrightarrow \exists q \in \mathcal{P} : a = *q \text{ или } b = *q$$

Для определенности будем считать, что $a = *q$. Тогда $b \notin \zeta(q)$, но $b \in \xi(q)$. Так как $\varphi_\zeta(*q, b) = never$ и φ_ζ консервативен, то $\phi^*(*q, b) = never$. Однако, из того, что $b \in \xi(q)$ следует, что $\phi^*(*q, b) \neq never$. Получили противоречие, значит, $\xi = \rho^*$. □

4.5 Композиция

Определение 14. Пусть ξ и ζ - анализы указателей. Тогда композицией анализов указателей будем называть следующую операцию:

$$\gamma = \xi \circ \zeta \Leftrightarrow \forall p \in \mathcal{P} \to \gamma(p) = \xi(p) \cap \zeta(p)$$

Лемма 2 (О композиции анализов указателей)**.** *Пусть ξ и ζ - анализы указателей. Тогда*

$$\gamma = \xi \circ \zeta \Leftrightarrow \varphi_\gamma = \varphi_\xi \circ \varphi_\zeta$$

Доказательство. 1. Пусть $\gamma = \xi \circ \zeta$. Рассмотрим φ_γ.

$$\forall q \in \mathcal{P}, \forall a \in \mathcal{V} \to \varphi_\gamma(*q, a) = never \Leftrightarrow a \notin \gamma(q)$$

Так как $\gamma = \xi \circ \zeta$, то $a \notin \xi(q) \cap \zeta(q)$. Преобразуем полученное выражение:

$$a \in \overline{\xi(q) \cap \zeta(q)} \Leftrightarrow a \in \overline{\xi(q)} \cup \overline{\zeta(q)}$$

Получаем, что $\left[\begin{array}{l} a \in \overline{\xi(q)}, \\ a \in \overline{\zeta(q)}, \\ a \in \overline{\xi(q)} \cap \overline{\zeta(q)} \end{array}\right. \Leftrightarrow \left[\begin{array}{l} a \notin \xi(q), \\ a \notin \zeta(q), \\ a \notin \xi(q) \cup \zeta(q) \end{array}\right.$

Как результат имеем, что $\varphi_\gamma(*q, a) = never \Leftrightarrow \left[\begin{array}{l} \left\{\begin{array}{l} \varphi_\xi(*q, a) = never, \\ \varphi_\zeta(*q, a) = may; \end{array}\right. \\ \left\{\begin{array}{l} \varphi_\xi(*q, a) = may, \\ \varphi_\zeta(*q, a) = never; \end{array}\right. \\ \left\{\begin{array}{l} \varphi_\xi(*q, a) = never, \\ \varphi_\zeta(*q, a) = never. \end{array}\right. \end{array}\right.$

Это соответствует операции уточнения на $\mathcal{D}$, откуда можно заключить, что $\varphi_\gamma = \varphi_\xi \circ \varphi_\zeta$.

2. Пусть $\varphi_\gamma = \varphi_\xi \circ \varphi_\zeta$. Тогда φ_γ равен $never$ для таких $q \in \mathcal{P}$ и $a \in \mathcal{V}$, что $\varphi_\xi(*q, a) = never$ или $\varphi_\zeta(*q, a) = never$. Это можно записать следующим образом:

$$\forall q \in \mathcal{P}, \forall a \in \mathcal{V} \to \varphi_\gamma(*q, a) = never \Leftrightarrow \left[\begin{array}{l} \varphi_\xi(*q, a) = never, \\ \varphi_\zeta(*q, a) = never \end{array}\right.$$

Преобразуем полученное выражение: $a \notin \xi(q)$ или $a \notin \zeta(q) \Leftrightarrow a \in \overline{\xi(q)}$ или $a \in \overline{\zeta(q)}$. Откуда можно заключить, что $a \in \overline{\xi(q)} \cup \overline{\zeta(q)}$.

То есть $\varphi_\gamma(*q, a) = never \Leftrightarrow a \notin \xi(q) \cap \zeta(q)$, что верно для произвольных q и a. То есть $\gamma = \xi \circ \zeta$.

□

Теорема 3 (О композиции консервативных анализов указателей)**.** *Пусть ξ и ζ - консервативные анализы указателей. Тогда $\gamma = \xi \circ \zeta$ также является консервативным анализом алиасов, причем $\xi \leq \gamma$ и $\zeta \leq \gamma$.*

Доказательство. Из леммы 2 имеем, что $\varphi_\gamma = \varphi_\xi \circ \varphi_\zeta$. Из теоремы о композиции консервативных анализов алиасов имеем, что φ_γ консервативен, а также, что $\varphi_\xi \leq \varphi_\gamma$ и $\varphi_\zeta \leq \varphi_\gamma$. Тогда γ консервативен по определению. Из леммы 1 получаем, что $\xi \leq \gamma$ и $\zeta \leq \gamma$. □

4.6 Теорема о консервативности анализа указателей

Теорема 4. *Пусть ξ - анализ указателей. ξ консервативен тогда и только тогда, когда $\xi \leq \rho^*$.*

Доказательство.

1. Пусть $\xi \leq \rho^*$. Из леммы 1 имеем, что $\varphi_\xi \leq \varphi_{\rho^*}$. Так как φ_{ρ^*} консервативен, то и φ_ξ тоже консервативен. Откуда по определению ξ консервативен.

2. Пусть ξ консервативен. Из определения ρ^* имеем, что $\nexists\zeta : \zeta$ консервативен и $\varphi_{\rho^*} < \varphi_\zeta$. Тогда либо φ_ξ и φ_{ρ^*} несравнимы, либо $\varphi_\xi \leq \varphi_{\rho^*}$.

   Пусть φ_ξ и φ_{ρ^*} несравнимы. Рассмотрим $\chi = \varphi_\xi \circ \varphi_{\rho^*}$. Из дополнения 2 к теореме о композиции консервативных анализов алиасов χ консервативен, причем $\varphi_{\rho^*} < \chi$. Из леммы 2 следует, что $\exists\gamma : \gamma = \xi \circ \rho^*$ и $\chi = \varphi_\gamma$. То есть

   $$\exists\gamma : \gamma \text{ консервативен и } \varphi_{\rho^*} < \varphi_\gamma$$

   Это противоречит определению ρ^*.

   Тогда остается, что $\varphi_\xi \leq \varphi_{\rho^*}$. Из леммы 1 получаем, что $\xi \leq \rho^*$.

□

Данные результаты позволяют работать исключительно с анализом указателей и говорить об их свойствах вне контекста анализа алиасов.

Далее, для краткости, в ситуациях, когда подразумевается один выбранный анализ указателей ξ, $\forall p \in \mathcal{P}$ будем писать $\mathcal{V}_p$ и иметь в виду, что $\xi(p) = \mathcal{V}_p$.

5 Классические решения

Все классические решения относятся к категории потоково- и контекстно-нечувствительных анализов.

5.1 Алгоритм Андерсена (94‘)

Разработан Ларсом Уле Андерсеном и представлен в соответствующей работе в 1994-ом году. Стоит отметить, что данный метод является наиболее точным среди тех, что могут быть получены за полиномиальное время.

При рассмотрении алгоритма будем использовать ориентированный граф следующего вида: $G =< V, E >$, где $V = \mathcal{V}$, а $E = \{(p, a) : p \in \mathcal{P}, a \in \mathcal{V}_p\}$.

То есть в данном графе вершины представляют все переменные программы. От вершины p имеется ребро в вершину a, только если p может указывать на a.

Основным остается вопрос о том, как на основании кода программы составить анализ указателей.
Андерсен выделяет следующие операции:

- `p = &a;`
- `p = q;`
- `p = *r;`
- `*p = &a;`
- `*p = q;`
- `*p = *r;`

Все остальные операции могут быть сведены к данным случаям.

Пример 15. Ниже приведен пример такого сведения.

```
...
a = **b;
...
```

Листинг 15: До приведения

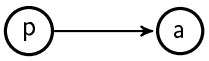
```
...
t = *b;
a = *t;
...
```

Листинг 16: После

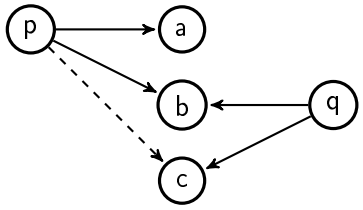
5.1.1 Построение анализа указателей

Теперь рассмотрим как анализируется каждый тип присваивания. Для большей наглядности каждый тип сразу же рассматривается на примере.

1. `p = &a;`
   Добавляем ребро от p к a, показывая, что $a \in \mathcal{V}_p$.

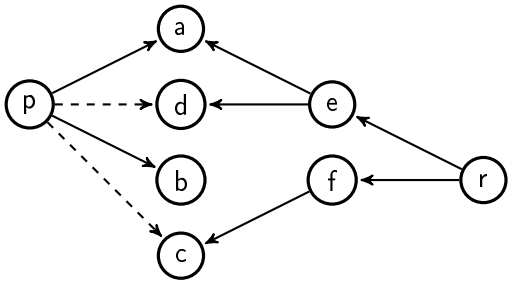


2. `p = q;`
   Добавляем ребра от p ко всем вершинам, куда есть ребра из q. Это значит также, что если позже будут добавлены ребра из q, аналогичные ребра должны быть добавлены из p. То есть должно сохраняться свойство, что $\mathcal{V}_q \subseteq \mathcal{V}_p$.



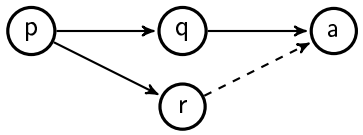
Здесь и далее пунктиром обозначены ребра, которые были добавлены позже. В данном случае на момент обработки инструкции `p = q;` было неизвестно, что $c \in \mathcal{V}_q$ (изменилось $\mathcal{V}_q$).

3. `p = *r;`
   Пусть $T = \bigcup\limits_{t \in \mathcal{V}_r} \mathcal{V}_t$. Тогда добавляем ребро из p к каждому элементу из T. Аналогично предыдущему случаю при изменении $\mathcal{V}_r$ или T необходимо добавить соответствующие ребра. То есть должно сохраняться свойство, что $\forall t \in \mathcal{V}_r \Rightarrow \mathcal{V}_t \subseteq \mathcal{V}_p$.



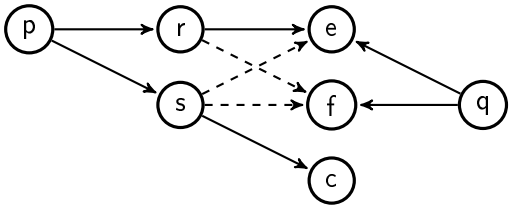
В данном примере пунктиром выделены ребра (p, c) и (p, d), так как на момент обработки инструкции `p = *r;` было неизвестно, что $d \in \mathcal{V}_e$ (изменилось T) и $f \in \mathcal{V}_r$ (изменилось $\mathcal{V}_r$).

4. `*p = &a;`
   Из каждой вершины, куда есть ребра из p добавляем ребро к a. Опять таки, при добавлении новых вершин в $\mathcal{V}_p$ из них необходимо добавить ребра к a. То есть должно сохраняться свойство, что $\forall t \in \mathcal{V}_p \rightarrow a \in \mathcal{V}_t$.



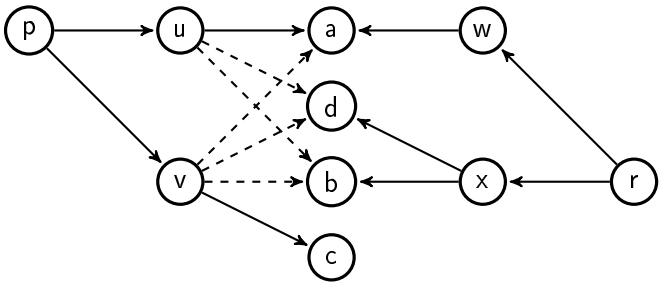
В данном примере пунктиром выделено ребро (r, a), так как на момент обработки было неизвестно, что $r \in \mathcal{V}_p$ (изменилось $\mathcal{V}_p$).

5. `*p = q;`
   Из каждой вершины из $\mathcal{V}_p$ добавляем ребро к каждой вершине из $\mathcal{V}_q$. При добавлении новых вершин в $\mathcal{V}_p$ или в $\mathcal{V}_q$ соответствующие новые ребра должны быть добавлены. То есть должно сохраняться свойство, что $\forall t \in \mathcal{V}_p \to \mathcal{V}_q \subseteq \mathcal{V}_t$.



   В данном примере пунктиром выделены ребра (r, f), (s, e) и (s, f), так как на момент обработки инструкции `*p = q;` было неизвестно, что $s \in \mathcal{V}_p$ (изменилось $\mathcal{V}_p$) и $f \in \mathcal{V}_q$ (изменилось $\mathcal{V}_q$).

6. `*p = *r;`
   Пусть $T = \bigcup_{t \in \mathcal{V}_r} \mathcal{V}_t$. Тогда необходимо добавить ребро из каждой вершины из $\mathcal{V}_p$ в каждую вершину из T. При добавлении новых вершин в T, $\mathcal{V}_r$ или $\mathcal{V}_p$ соответствующие ребра должны быть добавлены. То есть должно сохраняться свойство, что $\forall s \in \mathcal{V}_p, \forall t \in \mathcal{V}_r \to \mathcal{V}_t \subseteq \mathcal{V}_s$.



   В данном примере пунктиром выделены ребра (u, d), (u, b), (v, a), (v, d) и (v, b), так как на момент обработки инструкции `*p = *r;` не было известно, что $v \in \mathcal{V}_p$ (изменился $\mathcal{V}_p$), $x \in \mathcal{V}_r$ (изменился $\mathcal{V}_r$) и $b \in \mathcal{V}_x$ (изменился T).

5.1.2 Разрешение ограничений

В рассмотренных выше различных типах инструкций были выделены свойства, которые должны сохраняться. Данные свойства принято называть *ограничениями*. Основную сложность в построении анализа указателей Андерсена составляет как раз разрешение ограничений такое, чтобы все они были удовлетворены.

Еще раз перечислим все типы инструкций и ограничения им соответствующие:

1. `p = &a`: $a \in \mathcal{V}_p$
2. `p = q`: $\mathcal{V}_p \supseteq \mathcal{V}_q$
3. `p = *r`: $\forall t \in \mathcal{V}_r \to \mathcal{V}_t \subseteq \mathcal{V}_p$
4. `*p = &a`: $\forall t \in \mathcal{V}_p \to a \in \mathcal{V}_t$
5. `*p = q`: $\forall t \in \mathcal{V}_p \to \mathcal{V}_q \subseteq \mathcal{V}_t$
6. `*p = *r`: $\forall s \in \mathcal{V}_p, \forall t \in \mathcal{V}_r \Rightarrow \mathcal{V}_t \subseteq \mathcal{V}_s$

Определим формально задачу построения анализа указателей Андерсена ξ. Пусть $\mathcal{I}$ - множество инструкций, которые влекут за собой изменения состояний указателей, рассматриваемой программы [1] , а Λ - функция, сопоставляющая каждому $i \in \mathcal{I}$ предикат логики первого порядка.

Определение 15. *Ограничением*, связанным с $i \in \mathcal{I}$, будем называть $\Lambda(i)$.

Определение 16. Анализ указателей ξ будем называть *построенным по Андерсену* [2] , если

$$\forall i \in \mathcal{I} \to \Lambda(i)(\xi)$$

То есть ξ построен по Андерсену, если все ограничения истинны на ξ.

Как можно разрешить все ограничения? Можно итеративно обходить все инструкции и пытаться разрешить каждое ограничение по отдельности. Если за один такой проход ничего не было изменено, то все ограничения удовлетворены и анализ указателей построен. Отобразим данное решение в псевдокоде.

Алгоритм 1: (Андерсен)

```
while changed do
    changed ← false;
    foreach i ∈ I do
        if not Λ(i)(ξ) then
            solve Λ(i)(ξ);
            changed ← true;
        end
    end
end
```

[1] Будем считать, что в данном множестве все инструкции уже приведены к базовым типам

[2] Наряду с этим названием используется также термин "анализ указателей *по включению*"(inclusion-based)

Шаг *solve* $\Lambda(i)(\xi)$ осуществляется по правилам, описанным в разделе "Построение анализа указателей".

5.1.3 Результаты

Как уже говорилось ранее, среди анализов алиасов, которые могут быть получены за полиномиальное время, анализ алиасов, построенный по Андерсену, дает наилучший результат. На практике было отмечено, что результаты сильно превосходят по точности анализ по взятию адреса. Также было отмечено, что добавление потоковой и контекстной чувствительности лишь незначительно улучшает результат на распространенных бенчмарках.

Однако, асимптотическая сложность данного подхода $O(n^3)$, где $n = |\mathcal{V}|$, и в том виде, в котором его предложил Андерсен, алгоритм был неприменим к большим программным проектам из-за долгого времени работы алгоритма и огромного потребления памяти ($O(n^2)$) [1] .

5.2 Алгоритм Стинсгарда (96‘)

Разработан Бьярне Стинсгардом и представлен в соответствующей работе в 1996-ом году.

Прежде всего алгоритм Стинсгарда является попыткой существенно улучшить асимптотическую сложность, жертвуя при этом точностью анализа.

5.2.1 Эквивалентность переменных

Введем понятие эквивалентности двух переменных.

Определение 17. $v, w \in \mathcal{V}$ будем называть *эквивалентными по Андерсену* и обозначать $v \overset{A}{\sim} w$, если и только если $V_v = V_w$ и $\forall p \in \mathcal{P}$ верно, что $v \in V_p \Leftrightarrow w \in V_p$.

Утверждение 4. *Введенное отношение* $\overset{A}{\sim}$ *является отношением эквивалентности на* $\mathcal{V}$*, т.е. выполняются следующие свойства:*

1. *Рефлексивность:* $\forall a \in \mathcal{V} \to a \overset{A}{\sim} a$
2. *Симметричность:* $\forall a, b \in \mathcal{V} \to a \overset{A}{\sim} b \Rightarrow b \overset{A}{\sim} a$
3. *Транзитивность:* $\forall a, b, c \in \mathcal{V} \to a \overset{A}{\sim} b, b \overset{A}{\sim} c \Rightarrow a \overset{A}{\sim} c$

[1] Данная оценка легко показывается, если рассматривать задачу на графе. Для полного ориентированного графа на n вершинах имеем $n(n-1)$ ребер, что и дает асимптотическую оценку $O(n^2)$ памяти для его хранения

Доказательство данного утверждения очевидно и напрямую следует из определения отношения.

Пусть $a \in \mathcal{V}$. Тогда множество переменных $b \in \mathcal{V} : a \overset{A}{\sim} b$ называется классом эквивалентности a и обозначается $a/\overset{A}{\sim}$.

Множество классов эквивалентности по отношению $\overset{A}{\sim}$ называют фактор множеством и обозначают $\mathcal{V}/\overset{A}{\sim}$.

Теорема 5. *Пусть ζ - построенный по Андерсену анализ указателей. Если на множестве $\mathcal{V}/\overset{A}{\sim}$ построить анализ указателей по Андерсену ξ, понимая под этим, что $i \in \mathcal{I}$ всякое вхождение произвольной $a \in \mathcal{V}$ понимается как $[a]_A$. Тогда ξ совпадает с ζ.*

Доказательство. Для доказательства теоремы покажем, что в обоих случаях при построении решаются идентичные ограничения. На время доказательства под a' мы понимаем произвольную переменную из $[a]_A$.

1. $a \in \zeta(p)$ и $[a]_A \in \xi([p]_A)$. Из определения эквивалентности $a' \in \zeta(p)$. В силу произвольности a' верно, что $[a]_A \subseteq \zeta(p)$. Так как $\zeta(p) = \zeta(p')$ и это верно для произвольного p', то ζ соответствует ограничению вида $[a]_A \in \xi([p]_A)$.

2. $\zeta(p) \supseteq \zeta(q)$ и $\xi([p]_A) \subseteq \xi([q]_A)$. Из определения p' и q' верны следующие выражения: $\zeta(p') \supseteq \zeta(q), \zeta(p) \supseteq \zeta(q'), \zeta(p') \supseteq \zeta(q')$. Тогда ζ соответствует ограничению вида $\xi([p]_A) \subseteq \xi([q]_A)$.

3. $\forall t \in \zeta(r) \to \zeta(t) \subseteq \zeta(p)$ и $\forall [t]_A \in \xi([r]_A) \to \xi([t]_A) \subseteq \xi([p]_A)$. Так как $t \in \zeta(r)$, то $t' \in \zeta(r)$. Так как $\zeta(r) = \zeta(r'), \zeta(t) = \zeta(t')$ и $\zeta(p) = \zeta(p')$, то $t' \in \zeta(r') \to \zeta(t') \subseteq \zeta(p')$. Так как это выполнено для произвольных t', r' и p', то ζ соответствует ограничению вида $\forall [t]_A \in \xi([r]_A) \to \xi([t]_A) \subseteq \xi([p]_A)$.

Остальные случаи доказываются аналогично. □

Использовать это, однако, для улучшения производительности алгоритма не получится, так как отношение эквивалентности использует уже построенный по Андерсену анализ указателей.

Определение 18. $v, w \in \mathcal{V}$ будем называть *эквивалентными по Стинсгарду* и обозначать $v \overset{S}{\sim} w$, если и только если $\exists p \in \mathcal{P} : v, w \in V_p$.

Утверждение 5. *Введенное отношение $\overset{S}{\sim}$ является отношением эквивалентности на $\mathcal{V}$.*

Определение 19. Анализ указателей $\xi : \mathcal{P} \to 2^{\mathcal{V}}$ будем называть *построенным по Стинсгарду*, если он получен путем построения по Андерсену анализа $\zeta : \mathcal{P}/\overset{S}{\sim} \to 2^{\mathcal{V}/\overset{S}{\sim}}$.

Для данного отношения также верно, что результаты анализа указателей участвуют при формировании классов эквивалентности. Но в данном случае во время построения анализа можно строить классы эквивалентности "на лету". Если говорить другими словами, то как только становится известно, что $a, b \in \mathcal{V}_p$ для некоторого p, то a и b объединяются в класс эквивалентности.

5.2.2 Построение анализа

Алгоритм отличается от аналогичного у Андерсена лишь той деталью, что если во время разрешения ограничения необходимо (в терминах графа, рассмотренного в главе про алгоритм Андерсена) добавить ребро от p к b при том, что имеется ребро к a, то вершины a и b сливаются.

Пример 16. Пунктиром обозначено ребро, которое необходимо было добавить.

Алгоритм 2: (Стинсгард)

```
while changed do
    changed ← false;
    foreach i ∈ I do
        if not Λ(i)(ξ) then
            solve and merge Λ(i)(ξ);
            changed ← true;
        end
    end
end
```

5.2.3 Результаты

Асимптотическая сложность полученного алгоритма составляет $O(n * \alpha(n, n))$ [1] .

Также анализ указателей, построенный по Стинсгарду, показывает результаты значительно превосходящие подход по взятию адреса.

[1] Здесь $\alpha(n, n)$ - обратная функция Аккермана. Для всех практически встречающихся чисел значение этой функции меньше 5, что позволяет утверждать, что полученная сложность почти линейна

5.2.4 Разница с Андерсеном

Результаты подхода Стинсгарда уступают по точности результатам алгоритма Андерсена. Для понимания причин этого факта проще рассмотреть пример.

Пример 17. Рассмотрим анализы указателей, построенные по Андерсену и по Стинсгарду, для следующего кода:

```
p = &a;
q = &a;
p = &b;
```

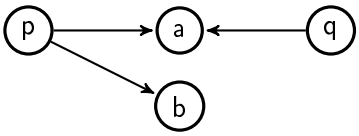


Рис. 1: Андерсен

Рис. 2: Стинсгард

В данном примере в анализе Стинсгарда ошибочно решается, что q может указывать на b.

Говоря о разнице в точности между двумя этими подходами, стоит сказать о том, что Горвиц и Шапиро протестировали 61 программу на C с размерами от 300 до 24 300 строк кода. Были получены следующие результаты:

- Стинсгард менее точен - в среднем размер множеств в 4 раза больше; в худшем случае в 15 раз.
- Андерсен медленнее - в среднем в полтора раза медленнее; в худшем случае в 31 раз.

5.3 Подход Горвиц-Шапиро (97‘)

Разработан Сюзан Горвиц и Марком Шапиро и представлен в соответствующей работе в 1997-ом году.

Подход представляет собой способ получить промежуточные звенья между подходами Андерсена и Стинсгарда.

5.3.1 Основная идея

Определение 20. *Степенью* указателя $p \in \mathcal{P}$ будем называть $|\mathcal{V}_p|$.

Заметим, что степень каждого указателя в случае Стинсгарда ограничена 1, а в случае Андерсена n, где n - число переменных в программе.

Тогда стоит рассмотреть случаи, когда степень указателей ограничена некоторым $k : 1 \leq k \leq n$. В данном случае $\forall p \in \mathcal{P}$ все переменные $v \in \mathcal{V}_p$ определяются в категории, где категория - это некоторое число из $[1, k]$. Тогда пара переменных $v, w \in \mathcal{V}$ считается эквивалентной, если $\exists p \in \mathcal{P}$ такой, что для p категории v и w совпадают.

Если говорить формально, то $\forall p \in \mathcal{P}$ определяется отображение $\eta_p : V_p \to [1, k]$.

Определение 21. $v, w \in \mathcal{V}$ будем называть *эквивалентными по Горвиц-Шапиро* и обозначать $v \overset{HS}{\sim} w$, если и только если $\exists p \in \mathcal{P} : v, w \in \mathcal{V}_p$ и $\eta_p(v) = \eta_p(w)$.

Пример 18. Рассмотрим следующий пример:

```
p1 = &a;
p1 = &b;
p1 = &c;
p2 = &c;
```

Рассмотрим случай $k = 2$.

Пусть $\eta_{p1}(a) = \eta_{p1}(b) = 1$, а $\eta_{p1}(c) = 2$. Тогда

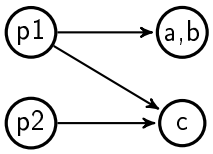


Рис. 3: Горвиц-Шапиро. Удачный выбор категорий

В данном случае точность анализа соответствует алгоритму Андерсена.

Однако, если же $\eta_{p1}(a) = 1$, а $\eta_{p1}(b) = \eta_{p1}(c) = 2$, то

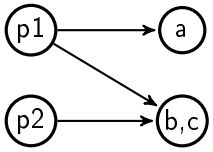


Рис. 4: Горвиц-Шапиро. Неудачный выбор категорий

В данном случае мы потеряли в точности, т.к. результат говорит, что $b \in V_{p2}$. Но стоит также отметить, что результат точнее Стинсгарда, т.к. в его случае ошибочно решается, что $a, b \in V_{p2}$.

Из рассмотренного примера видно, что точность анализа сильно зависит от разбиения на категории, т.е. от выбора отображений η_p.

5.3.2 Конкретизация

С точки зрения реализации, формирование $|\mathcal{P}|$ отображений не является практичным. В связи с этим выбирается одно отображение $\eta : \mathcal{V} \to [1, k]$. Тогда $\forall p \in \mathcal{P}, \forall v \in \mathcal{V}_p \to \eta_p(v) = \eta(v)$.

Асимптотическая сложность полученного алгоритма в данном случае будет равна $O(k^2 n)$.

Результирующий анализ указателей ξ непосредственно зависит от выбора η (далее будем обозначать ξ_η). При этом независимо от выбора η ξ_η остается консервативным, что позволяет для нескольких выборов отображения $\eta_1, \ldots, \eta_m$ произвести композицию анализов указателей $\zeta = \xi_{\eta_1} \circ \cdots \circ \xi_{\eta_m}$. По теореме о композиции консервативных анализов указателей имеем, что ζ консервативен и $\forall i \in [1, m] \to \xi_{\eta_i} \leq \zeta$.

Так как мы запускаем схожие алгоритмы m раз, то и результирующая сложность равна $O(mk^2 n)$.

Однако, до сих пор остался нерешенным вопрос о том, что делать с отображениями $\eta_1, \ldots, \eta_m$ и какому свойству необходимо их строить.

Горвиц и Шапиро предложили следующее *эвристическое* правило: $\forall v, w \in \mathcal{V} : v \neq w \to \exists i \in [1, m] : \eta_i(v) \neq \eta_i(w)$. Или, что то же самое, для любой пары переменных верно, что существует отображение η_i, которое определяет v и w в разные категории.

Рассмотрим следующую процедуру. Перенумеруем каждую переменную по основанию k. Тогда каждой переменной будет сопоставлено m-значное число. Именно m будет количеством запусков, а i-ый разряд будет категорией данной переменной в i-ом запуске.

Пример 19. Пусть имеется 4 переменные a, b, c и d и $k = 2$. Пронумеруем переменные:

$$a^{00}, b^{01}, c^{10}, d^{11}$$

Тогда число запусков $m = \lfloor \log_2(4 - 1) \rfloor + 1 = 1 + 1 = 2$. При первом запуске переменные a и c попадут в одну категорию, а b, d в другую. При втором же в первую категорию попадут a и b, а c и d во вторую.

Утверждение 6. *Подобное распределение отображений $\eta_1, \ldots, \eta_m$ соответствует правилу, сформулированному выше.*

Доказательство. Пусть правило не выполянется, тогда $\exists v, w \in \mathcal{V} : v \neq w, \forall i \in [1, m] : \eta_i(v) = \eta_i(w)$. Однако, это означает, что в числах, соответствующих v и w все разряды равны, чего не может быть при $v \neq w$. Получили противоречие. □

При таком подходе имеем $m = \lfloor \log_k(n - 1) \rfloor + 1$ и, соответственно, время выполнения $O(\log_k(n) k^2 n)$.

5.3.3 Построение анализа

Собственно алгоритм отличается от Стинсгарда тем, что работает $\lfloor \log_k(n-1) \rfloor + 1$ раз и устанавливает немного отличающееся отношение эквивалентности.

Алгоритм 3: (Горвиц-Шапиро)

```
enumerate 𝒱;
m ← ⌊log_k(n − 1)⌋ + 1;
for j ∈ [1, m] do
    while changed do
        changed ← false;
        foreach i ∈ ℐ do
            if not Λ(i)(ξ_j) then
                solve and merge categories Λ(i)(ξ_j);
                changed ← true;
            end
        end
    end
end
ζ ← ξ_1 ∘ ··· ∘ ξ_m;
```

5.3.4 Результаты

На 25 тестах, при использовании 3 категорий, результирующие множества, полученные методом Горвиц-Шапиро, в среднем в 2.67 раз больше, чем множества Андерсена (для Стинсгарда этот множитель равен 4.75).

Подход Горвиц-Шапиро медленнее Стинсгарда, но от 7 до 25 раз быстрее Андерсена.

6 Современные подходы

С ростом, как производительности компьютеров, так и объемов памяти, снова проявилась тенденция к получению более точных анализов. В качестве основы был выбран принцип Андерсена, однако, способы разрешения ограничений значительно отличаются.

Перед рассмотрением непосредственно методов необходимо представить структуру данных, которая используется в каждом из следующих алгоритмов.

6.1 Граф ограничений

Прежде всего стоит отметить, что число основных инструкций было сокращено с 6 до 4 (по сравнению с Андерсеном).

6.1.1 Ограничения

- `p = &a;`
- `p = q;`
- `p = *r;`
- `*p = &a;`
- `*p = q;`
- `*p = *r;`

$\Rightarrow$

- `p = &a;`
- `p = q;`
- `p = *r;`
- `*p = q;`

Это сделано, потому что инструкции `*p = &a;` и `*p = *r;` сводятся к остальным простыми преобразованиями, то есть `*p = &a;` $\equiv$ `t = &a; *p = t;` и `*p = *r;` $\equiv$ `t = *r; *p = t;`.

Ограничения, связанные с основными инструкциями, получили несколько иные обозначения и специальные названия:

Инструкция	Ограничение	Название ограничения	Смысл ограничения
$p = \&a$	$p \supseteq \{a\}$	Базовое ограничение	$b \in \mathcal{V}_p$
$p = q$	$p \supseteq q$	Простое ограничение	$\mathcal{V}_p \supseteq \mathcal{V}_q$
$p = *q$	$p \supseteq *q$	Сложное ограничение I	$\forall t \in \mathcal{V}_q \to \mathcal{V}_p \supseteq \mathcal{V}_t$
$*p = q$	$*p \supseteq q$	Сложное ограничение II	$\forall t \in \mathcal{V}_p \to \mathcal{V}_t \supseteq \mathcal{V}_q$

Теперь можем переходить непосредственно к графу ограничений.

6.1.2 Определение

Определение 22. *Графом ограничений* G будем называть следующую пятерку $< V, E, \xi, \mathcal{C}_1, \mathcal{C}_2 >$, где

$V = \mathcal{P}$ - множество вершин графа ограничений

E - множество ориентированных ребер графа, такое что $\forall a, b \in \mathcal{P} \to ((a, b) \in E \Leftrightarrow \mathcal{V}_b \supseteq \mathcal{V}_a)$. Изначально

$$E = E_0 = \{(a, b) : a, b \in \mathcal{P} \text{ и имеется ограничение } b \supseteq a\}$$

ξ - анализ указателей, который сопоставляет каждой вершине p множество $\mathcal{V}_p$, изначально

$$\xi : \forall p \in \mathcal{P} \to \xi(p) = \{v : v \in \mathcal{V} \text{ и имеется ограничение } p \supseteq \{v\}\}$$

$\mathcal{C}_1$ - отображение $\mathcal{P} \to 2^{\mathcal{P}}$, такое что

$$\forall p \in \mathcal{P}\ \mathcal{C}_1(p) = \{q : q \in \mathcal{P} \text{ и имеется ограничение } q \supseteq *p\}$$

То есть каждой вершине графа p отображение $\mathcal{C}_1$ сопоставляет множество вершин, которые входят в сложные ограничения I вместе с $*p$

$\mathcal{C}_2$ - отображение $\mathcal{P} \to 2^{\mathcal{P}}$, такое что

$$\forall p \in \mathcal{P}\ \mathcal{C}_2(p) = \{q : q \in \mathcal{P} \text{ и имеется ограничение } *p \supseteq q\}$$

То есть $\mathcal{C}_2$ идентично $\mathcal{C}_1$ с той лишь разницей, что рассматриваются сложные ограничения II.

6.1.3 Разрешение ограничений на графе

Теперь следует рассмотреть данный граф с точки зрения решения построения анализа.

Рассмотрим несколько ситуаций, которые позволят понять как устроено построение анализа.

- Пусть $a, b \in \mathcal{P}$, $(a, b) \in E$ и $\xi(a) = \{c\}$. Так как $(a, b) \in E$, то $\xi(b) \supseteq \xi(a)$, то есть $\xi(b) \supseteq \{c\}$ и, если c не было в $\xi(b)$, то его необходимо добавить. Подобная операция называется *распространением значений по ребрам графа ограничений.*

  Если изобразить подобную ситуацию собственно в виде графа, то получим следующее:

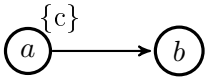


Рис. 5: До

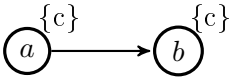


Рис. 6: После

- Пусть $a, b \in \mathcal{P}$, $\xi(a) = \{c\}$ и $\mathcal{C}_1(a) = \{b\}$. Из того как сформулировано $\mathcal{C}_1$ следует, что $\forall t \in \mathcal{V}_a \to \mathcal{V}_b \supseteq \mathcal{V}_t$, откуда следует, что $\mathcal{V}_b \supseteq \mathcal{V}_c$, и следует добавить ребро (c, b).

  Если изобразить подобную ситуацию собственно в виде графа, то получим следующее:

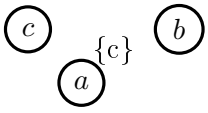


Рис. 7: До

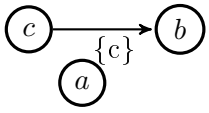


Рис. 8: После

- Пусть $a, b \in \mathcal{P}$, $\xi(a) = \{c\}$ и $\mathcal{C}_2(a) = \{b\}$. Из того как сформулировано $\mathcal{C}_2$ следует, что $\forall t \in \mathcal{V}_a \to \mathcal{V}_t \supseteq \mathcal{V}_b$, откуда следует, что $\mathcal{V}_c \supseteq \mathcal{V}_b$, и следует добавить ребро (b, c).

  Если изобразить подобную ситуацию собственно в виде графа, то получим следующее:

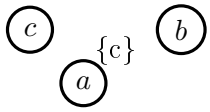


Рис. 9: До

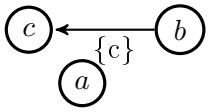


Рис. 10: После

На основании всего лишь этих трех правил запишем итеративный алгоритм решения, который также называют алгоритмом *динамического транзитивного замыкания.*

Алгоритм 4: (Динамическое транзитивное замыкание)

$W \leftarrow V$;
while $W \neq \emptyset$ **do**
  $n \leftarrow$ get from W;
  remove n from W;
  foreach $v \in \xi(n)$ **do**
    foreach $a \in \mathcal{C}_1(n)$ **do**
      if $(v, a) \notin E$ **then**
        $E \leftarrow E \cup \{(v, a)\}$;
        $W \leftarrow W \cup \{v\}$;
      end
    end
    foreach $b \in \mathcal{C}_2(n)$ **do**
      if $(b, v) \notin E$ **then**
        $E \leftarrow E \cup \{(b, v)\}$;
        $W \leftarrow W \cup \{b\}$;
      end
    end
  end
  foreach $(n, z) \in E$ **do**
    $\xi(z) \leftarrow \xi(z) \cup \xi(n)$;
    if $\xi(z)$ *changed* **then**
      $W \leftarrow W \cup \{z\}$;
    end
  end
end

6.1.4 Пример работы алгоритма

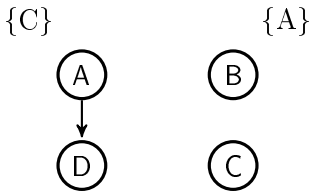
Пример 20. Рассмотрим работу алгоритма на следующем коде:

```
B = &A;
A = &C;
D = A;
*D = B;
```

```
A = *D;
```

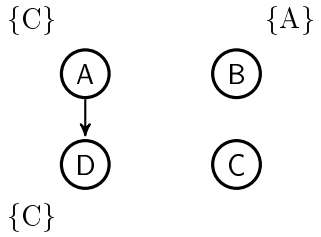
Для данного примера $\mathcal{P} = \{A, B, C, D\}$, $E = E_0 = (A, D)$, $\xi(B) = \{A\}, \xi(A) = \{C\}$, $\mathcal{C}_1(D) = \{A\}$ и $\mathcal{C}_2(D) = \{B\}$.

Изобразим данный граф:

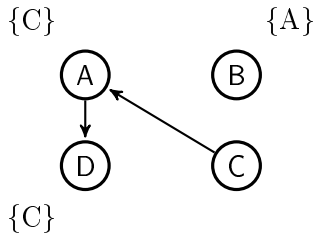


Начнем алгоритм

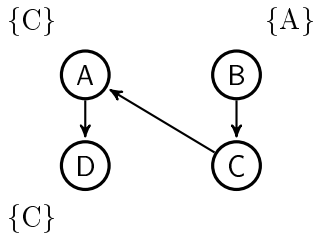
1. $W = \{A, B, C, D\}$;
2. $n = A$ и $W = \{B, C, D\}$;
3. $\mathcal{C}_1(A) = \mathcal{C}_2(A) = \emptyset$, поэтому ничего не делаем;
4. $(A, D) \in E$, поэтому $\xi(D) = \xi(D) \cup \xi(A) = \emptyset \cup \{C\} = \{C\}$;



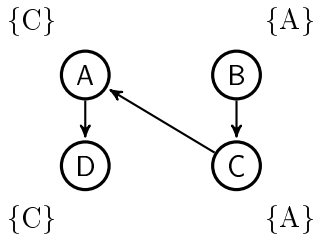
5. Добавляем D в W. $W = \{B, C, D\}$;
6. $n = B$ и $W = \{C, D\}$;
7. $\mathcal{C}_1(B) = \mathcal{C}_2(B) = \emptyset$, поэтому ничего не делаем;
8. Ребер из B нет, ничего не делаем;
9. $n = C$ и $W = \{D\}$;
10. $\mathcal{C}_1(C) = \mathcal{C}_2(C) = \emptyset$, поэтому ничего не делаем;
11. Ребер из C нет, ничего не делаем;
12. $n = D$ и $W = \emptyset$;
13. $v = C$ и $\mathcal{C}_1(D) = \{A\}$, поэтому добавляем ребро (C, A);



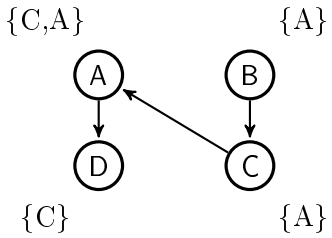
14. Добавляем C в W. $W = \{C\}$;
15. $v = C$ и $\mathcal{C}_2(D) = \{B\}$, поэтому добавляем ребро (B, C);



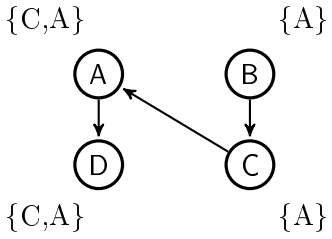
16. Добавляем B в W. $W = \{B, C\}$;
17. Ребер из D нет, ничего не делаем;
18. $n = B$ и $W = \{C\}$;
19. $\mathcal{C}_1(B) = \mathcal{C}_2(B) = \emptyset$, поэтому ничего не делаем;
20. $(B, C) \in E$, поэтому $\xi(C) = \xi(C) \cup \xi(B) = \emptyset \cup \{A\} = \{A\}$;



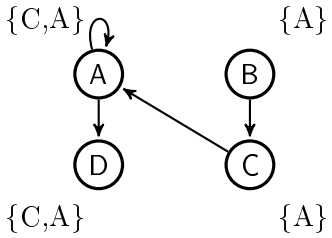
21. Добавляем C в W. $W = \{C\}$;
22. $n = C$ и $W = \emptyset$;
23. $\mathcal{C}_1(C) = \mathcal{C}_2(C) = \emptyset$, поэтому ничего не делаем;
24. $(C, A) \in E$, поэтому $\xi(A) = \xi(A) \cup \xi(C) = \{C\} \cup \{A\} = \{C, A\}$;



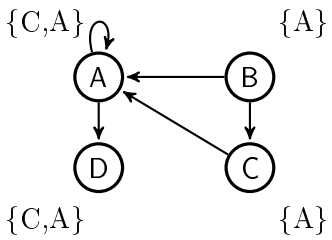
25. Добавляем A в W. $W = \{A\}$;
26. $n = A$ и $W = \emptyset$;
27. $\mathcal{C}_1(A) = \mathcal{C}_2(A) = \emptyset$, поэтому ничего не делаем;
28. $(A, D) \in E$, поэтому $\xi(D) = \xi(D) \cup \xi(A) = \{C\} \cup \{C, A\} = \{C, A\}$;



29. Добавляем D в W. $W = \{D\}$;
30. $n = D$ и $W = \emptyset$;
31. $v = C$, так как ребра (C, A) и (B, C) уже есть, ничего не делаем;
32. $v = A$ и $\mathcal{C}_1(D) = \{A\}$, поэтому добавляем ребро (A, A);



33. Добавляем A в W. $W = \{A\}$;
34. $v = A$ и $\mathcal{C}_2(D) = \{B\}$, поэтому добавляем ребро (B, A);



35. Добавляем A в W. $W = \{A\}$;
36. Ребер из D нет, ничего не делаем;
37. $n = A$ и $W = \emptyset$;
38. $\mathcal{C}_1(A) = \mathcal{C}_2(A) = \emptyset$, поэтому ничего не делаем;
39. $(A, D) \in E$, поэтому $\xi(D) = \xi(D) \cup \xi(A) = \{C, A\} \cup \{C, A\} = \{C, A\}$;
40. Множество $\xi(D)$ не изменилось, поэтому ничего не делаем;
41. Множество $W = \emptyset$, поэтому завершаем работу алгоритма.