САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Методы анализа и обеспечения качества ПО

Михаил Моисеев

Обнаружения нефункциональных ошибок в последовательных программах на основе статического анализа

Санкт-Петербург
2012

Проверочная работа 1

1. Что такое СА, чем он отличается от других статических методов?
2. Перечислите виды Data Flow анализа.
3. Перечислите алгоритмы Data Flow анализа.
4. Что такое решетка?
5. Как используется теория решеток для решения систем уравнений?
6. В чем заключается основная идея абстрактной интерпретации?
7. Перечислите показатели эффективности СА.
8. От чего зависит эффективность СА?
9. В чем выражается влияние свойств ЯП на эффективность СА?
10. Почему внешние воздействия оказывают влияние на СА?

Нефункциональные ошибки в программах на языках C/C++

- Нефункциональные ошибки
  - В последовательных программах – **программные дефекты**
  - В параллельных программах программные дефекты и ошибки синхронизации
- Классификация программных дефектов
  - CERT https://www.securecoding.cert.org/confluence/display/seccode/CERT+C+Secure+Coding+Standard
  - Coverity Open Source reports http://www.scan.coverity.com/report/

Классификация программных дефектов

#	**Наименование**	**Краткое описание**
RES	Ошибки управления ресурсами и памятью	Множественное освобождение, нарушение протокола работы с ресурсом
LEAK	Утечки ресурсов и динамической памяти	Потеря последней ссылки на выделенный ресурс или память
BUF	Ошибки работы с буферами/ массивами	Выход за границы объекта при чтении или записи, некорректные операции с указателями
INI	Ошибки отсутствия инициализации	Использование неинициализированных переменных или разыменование указателей
EXP	Ошибки в арифметических операциях	Деление на ноль, переполнение
DC	Мертвый код	Наличие недостижимых частей программы
IO	Ошибки ввода/вывода	Ошибки форматной строки, использование опасных функций ввода

Примеры программных дефектов (RES)

- Примеры ошибок управления ресурсами/памятью
  - повторное освобождение памяти
  - нарушение протокола работы с ресурсом

```
int main() {
  int *p = malloc(SIZE);
  …
  free(p);
  …
  free(p);
}
```

```
int main() {
  FILE f = fopen(fname, “r”);
  …
  fwrite(…, f);
}
```

Примеры программных дефектов (LEAK)

- Примеры утечек ресурсов/памяти
  - утечка динамической памяти при выходе из функции
  - утечка динамической памяти при выделении нового объекта

```
int f() {
  int *p = malloc(SIZE);
  …
  return 0;
}
```

```
int main() {
  int *p = malloc(SIZE);
  int *q = malloc(SIZE);
  …
  if (flag){
    p = q;
  }
  …
}
```

Примеры программных дефектов (BUF)

- Примеры ошибок работы с буферами/массивами
  - разыменование указателя за границей объекта
  - операции над указателями на разные объекты

```
int main()
{
  int arr[10];
  int *p;
  …
  p = arr;
  for(int i=0; i<=10; i++)
     printf("%d" , *p++);
}
```

```
int main()
{
  int a[10];
  int b[15];
  int *p = a;
  int *q = b+10;
  …
  int diff = (p - q);
}
```

Примеры программных дефектов (INI)

- Примеры ошибок отсутствия инициализации
  - использование неинициализированной переменной
  - разыменование неинициализированного указателя

```
int main() {
  int i;
  …
  if (i > 0){
    …
  }
  …
}
```

```
int main() {
  int *p;
  …
  if (flag){
    int i =*p;
  }
  …
}
```

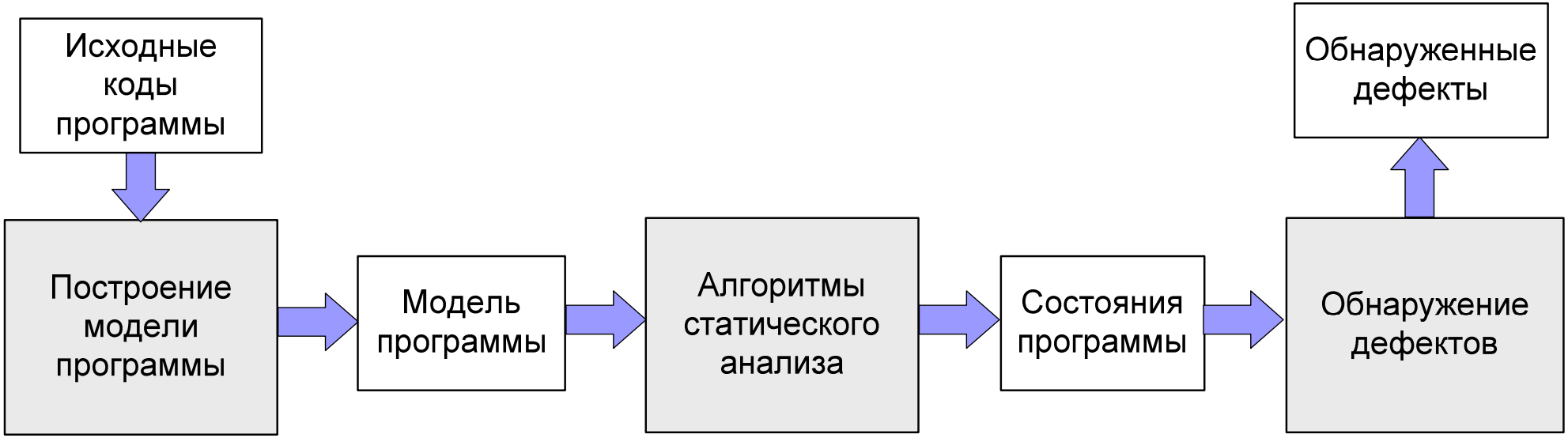
Примеры программных дефектов (IO)

- Примеры ошибок ввода/вывода
  - использование неконтролируемого значения в качестве форматной строки
  - использование опасных функций ввода

```
int main()
{
  char value[50];
  scanf("%49s", value);
  …
  printf(value);
}
```

```
int main()
{
  char* s = malloc(10);
  …
  gets(s);
}
```

Общая структура статического анализа



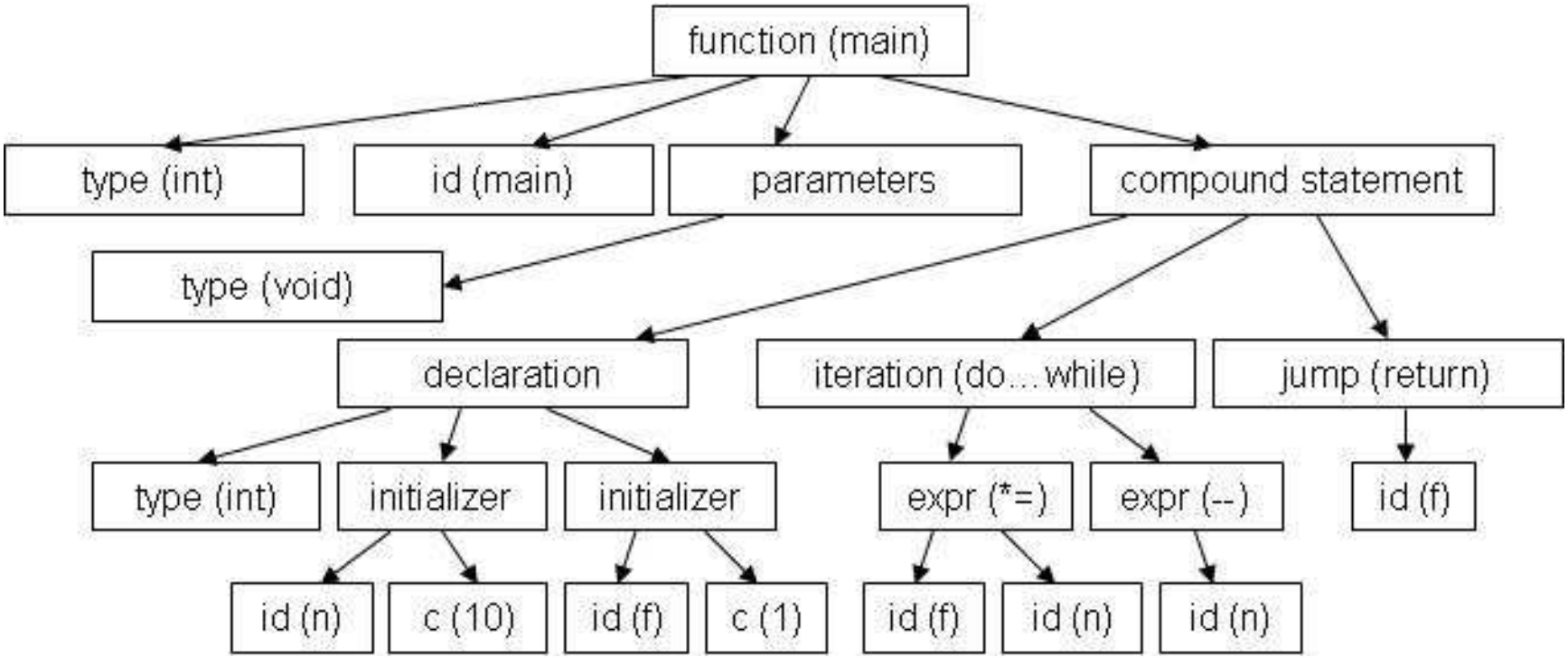
Модели программы

- Причины использования модели программ
- Модель исходного кода должна
  - обеспечивать доступ ко всем объектам программного кода и их атрибутам
  - поддерживать эффективный поиск объектов
  - снижать сложность разработки методов статического анализа
- Используемые модели
  - абстрактное синтаксическое дерево (AST)
  - абстрактный семантический граф (ASG)
  - граф потока управления (CFG)
  - граф зависимостей по данным (DDG)
  - представление на основе статического однократного присваивания (SSA)

Абстрактное синтаксическое дерево

- Дерево разбора (Parse tree) – вершины сопоставлены с нетерминалами формальной грамматики, а листья – с терминалами
- **Abstract Syntax Tree (AST)** – упрощенное дерево разбора, в котором нетерминальные узлы с единственным нетерминальным потомком преобразуются в атрибуты узлов
- Могут вводится новые узлы с большей семантической нагрузкой

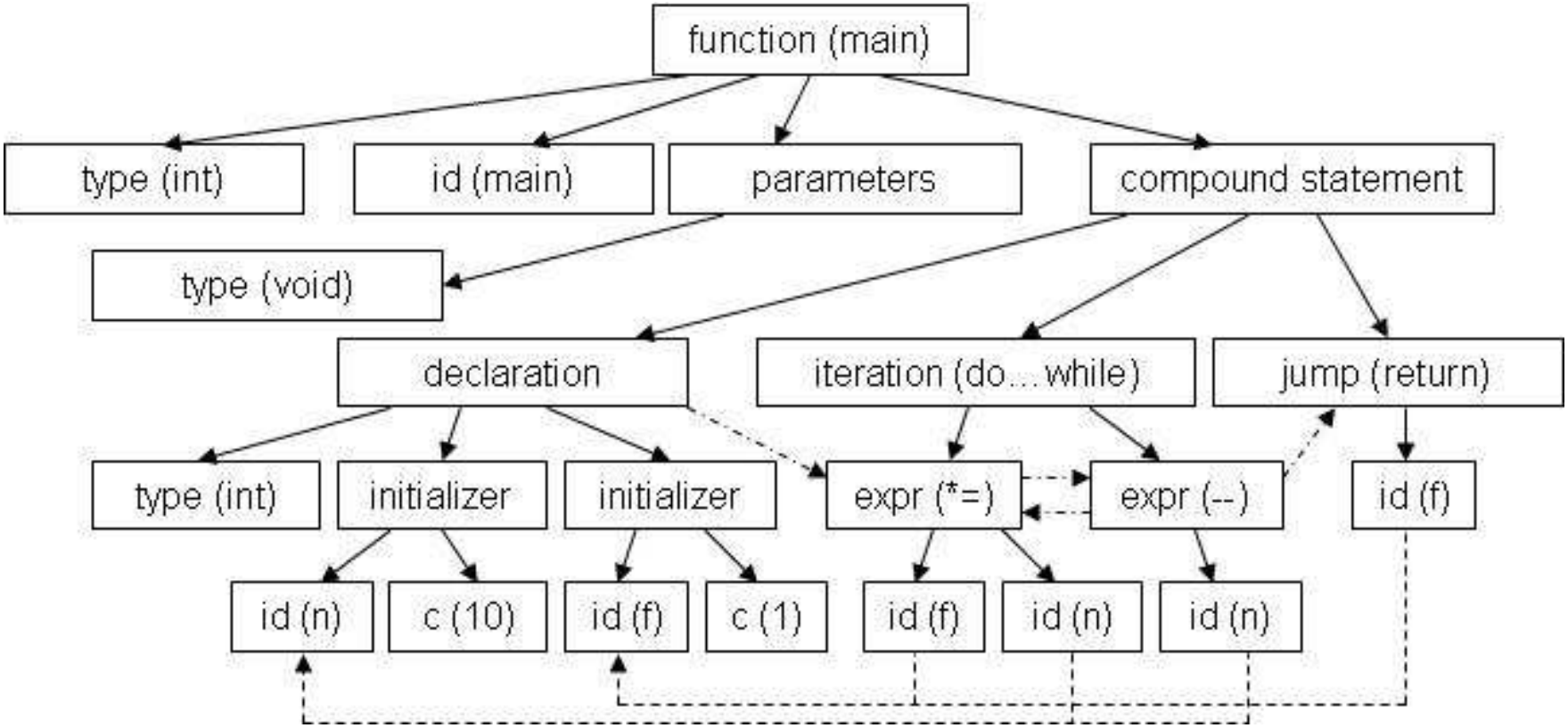
Абстрактное синтаксическое дерево



Абстрактный семантический граф

- **Abstract Semantic Graph (ASG)** – AST дополненное семантической информацией
- Семантическая информация получается в результате Control Flow или Data Flow анализа
- Семантическая информация представляется в виде дуг
  - дуги текущей инструкции к следующей
  - дуги от использования функции к ее определению
  - дуги от определения значения переменной к использованию

Абстрактный семантический граф

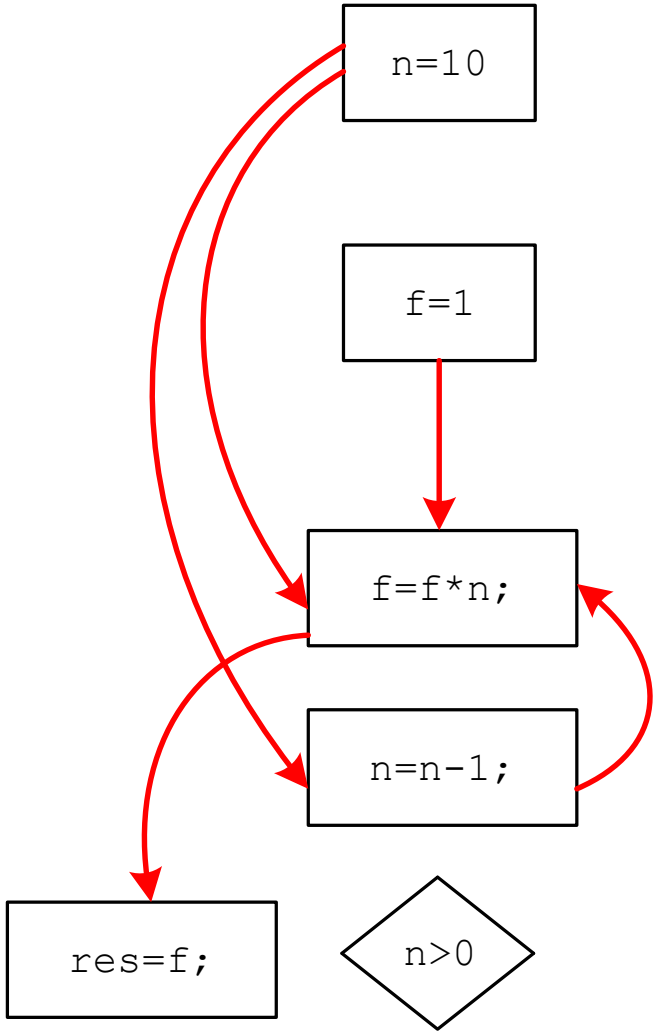


Граф зависимостей по данным

- **Data Dependency Graph (DDG)** – зависимости по данным между узлами-инструкциями представляются в виде направленных дуг
- Дуга связывает два узла тогда и только тогда, когда между ними есть зависимость по данным
  - запись-чтение
  - чтение-запись
  - запись-запись

Пример DDG

```
int n = 10;
int f = 1;
while (n > 0) {
   f = f * n;
   n = n – 1;
}
res = f;
```

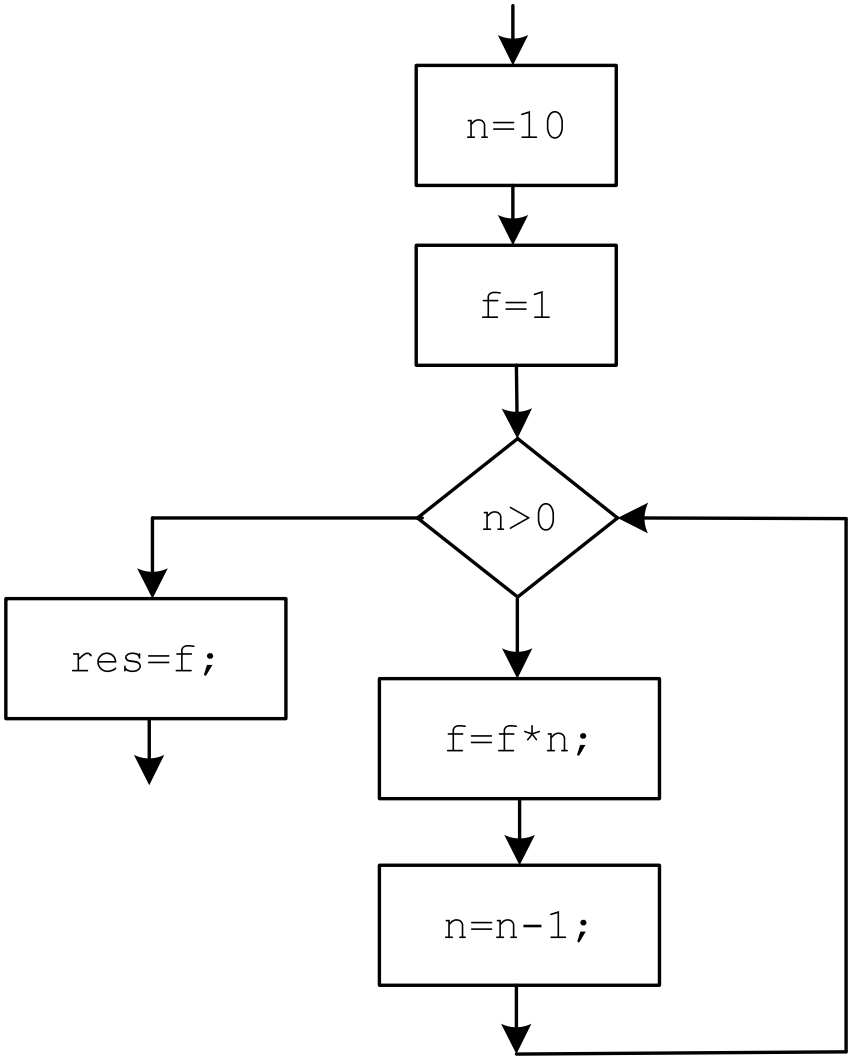


Граф потока управления

- **Control Flow Graph (CFG)** – поток управления представляется в виде ориентированного графа
- Сохраняется информация о конструкциях программы и переходах между ними
- При построении CFG учитываются
  - безусловные переходы
  - ветвления
  - циклы
  - вызовы функций
  - исключения

Пример CFG

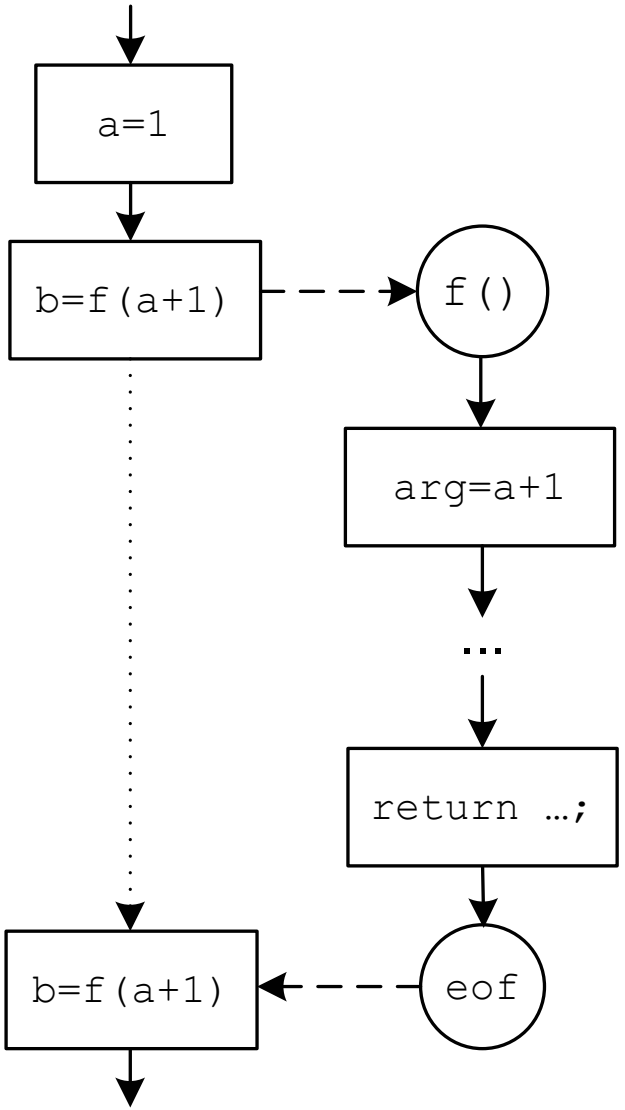
```
int n = 10;
int f = 1;
while (n > 0) {
   f = f * n;
   n = n - 1;
}
res = f;
```



Пример CFG

```
int a = 1;
int b = f(a+1);


float c = g(f(b)*a);
```

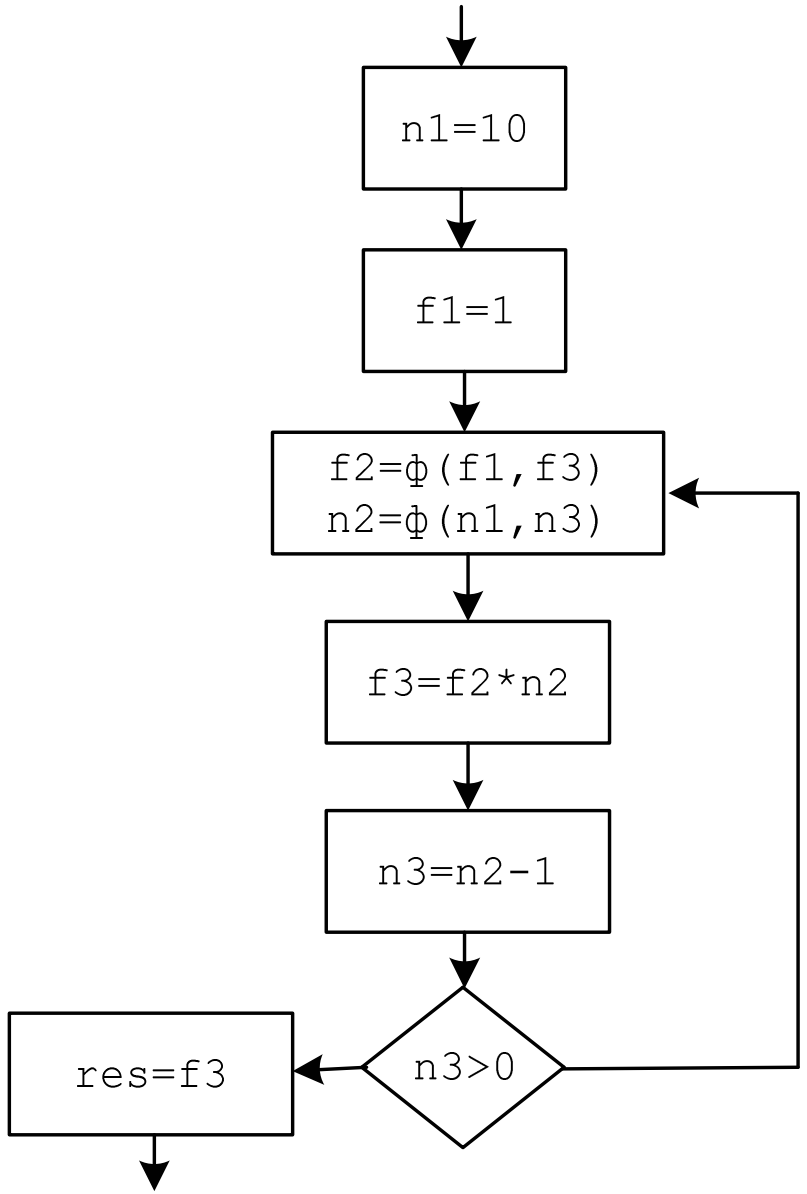


Представление на основе статического однократного присваивания

- Static single assignment form (SSA) – представление программы похожее на CFG
  - каждой переменной значение присваивается только один раз
  - в случае присвоения переменной нескольких значений используется версионирование
  - вводятся ***Φ***-функции, объединяющие ветки программы
  - циклы заменяются инструкциями ветвления и безусловных переходов
  - все выражения находятся в трёхоперандной форме

Пример SSA

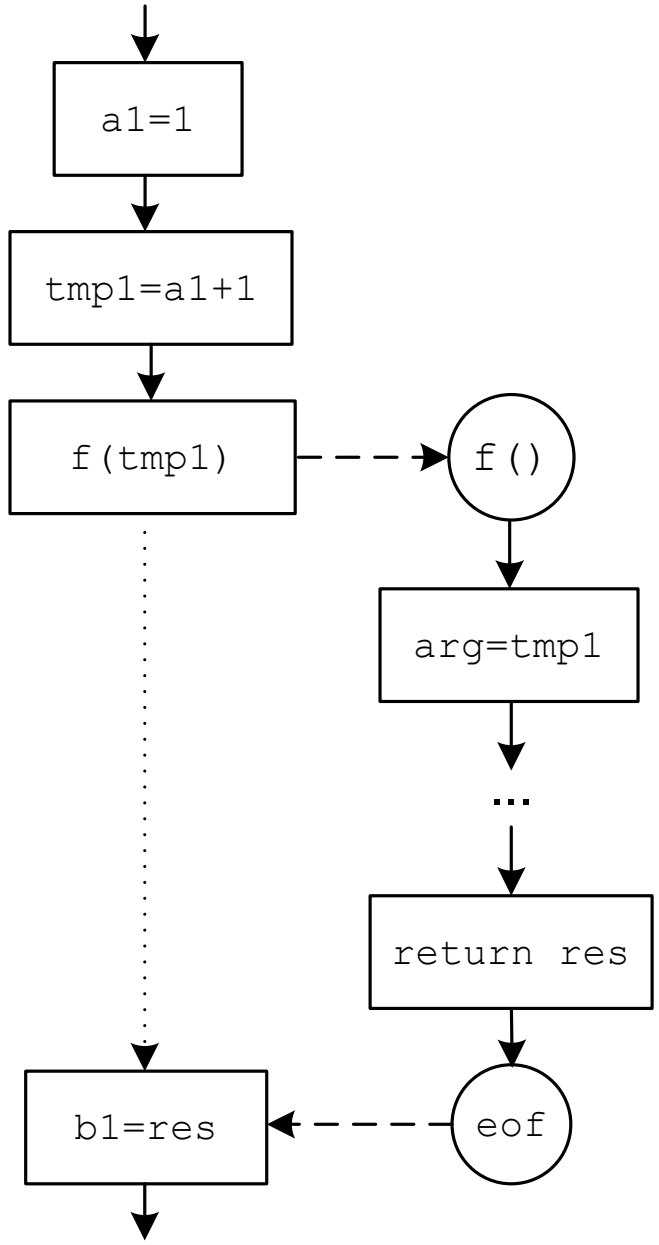
```
int n = 10;
int f = 1;
while (n > 0) {
   f = f * n;
   n = n - 1;
}
res = f;
```



Пример SSA

```
int a = 1;
int b = f(a+1);


float c = g(f(b)*a);
```



Построение модели программы

- Средства построения модели программы
  - использование генераторов парсеров (JavaCC, ANTLR, Bison, YACC)
  - использование самостоятельных парсеров (Elsa)
  - использование парсеров в составе средств разработки (NetBeans, CDT)
  - использование средств компиляции (gcc)
  - использование фреймворков для статического анализа (SUIF, CIL, LLVM, IRE)

Анализ программ на основе шаблонов

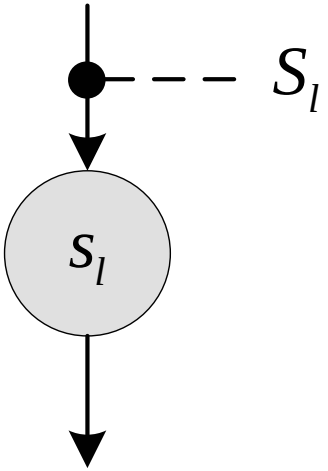
- Поиск параметризируемых шаблонов в исходном коде или построенной модели программы
- Шаблон представляет собой конструкции программы и переменные в качестве параметров
- Шаблоны для обнаружения дефектов
  - дефекты должны представляться 1-2 последовательными конструкциями
  - конструкции представляющие дефект должны быть локализированы

Анализ программ на основе шаблонов

- Достоинства и недостатки
  - простота реализации и низкая ресурсоемкость
  - возможность обнаруживать «тонкие» нарушения стиля кодирования и правил использования библиотечных функций
  - невозможность обнаружения широкого класса дефектов в произвольных ситуациях
- Средство анализа на основе шаблонов Viva64 http://www.viva64.com/

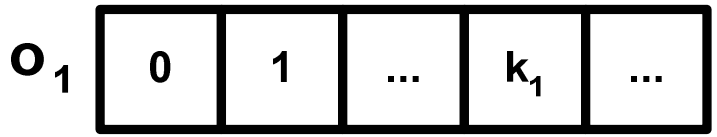
Анализ программ на основе состояний

- Определяются состояния объектов программы в отдельных конструкциях
- Состояние программы – возможные значения всех объектов, видимых в данной точке
- Состояние программы представляет как множество кортежей



$$S_l = \{tuple_i\}$$

$$tuple = \langle (o_1, k_1), value \rangle$$

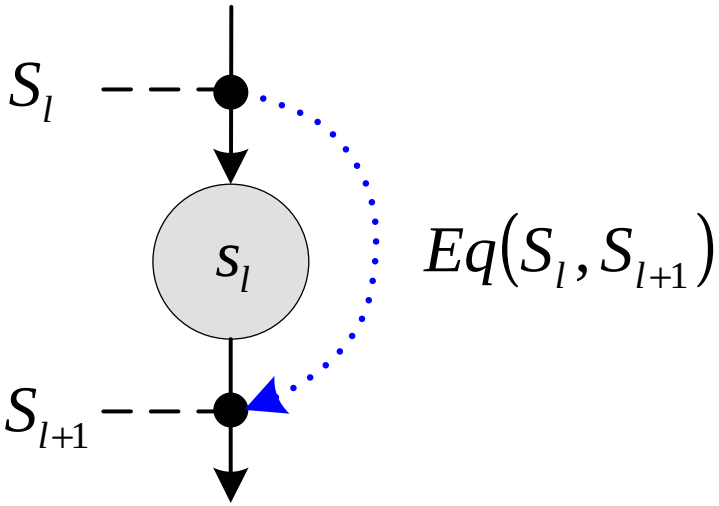


Алгоритмы определения состояний

- Различные типы объектов программы – различные правила и различные значения в правой части кортежей
- Анализ потока данных (анализ конструкций программы внутри процедур/функций)
  - алгоритм интервального анализа
  - алгоритм анализа указателей
  - алгоритм ресурсного анализа
- Анализ потока управления (межпроцедурный анализ)

Общая организация алгоритмов

- Алгоритмы анализа представлены в виде правил для конструкций программы (правила анализа д.б. монотонны)
- Каждое правило формирует уравнение, переменными являются состояния объектов программы



$$Rule(s_l) \rightarrow Eq(S_l, S_{l+1})$$

- Строится система уравнений, в результате решения которой получаются состояния объектов программы

Алгоритм интервального анализа

- Правила для конструкций, результат выполнения которых имеет интервальный тип (`char`, `int`, `float` и т.д.)
- Значения для интервальных переменных

$$value = i, i = (i_{\min}, i_{\max})$$

$$\langle\langle o_1, k_1 \rangle (i_{\min}, i_{\max}) \rangle\rangle$$

- Для учета неинициализированных переменных используется специальный интервал $i_{noninit}$

Правила интервального анализа

$$[declare(T\ a)]^l: \quad S_{l+1} = S_l \cup \left\langle \left\langle a, 0 \right\rangle, i_{noninit} \right\rangle$$

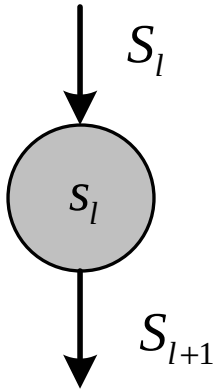
$$\left\langle a, 0 \right\rangle \Leftrightarrow a$$

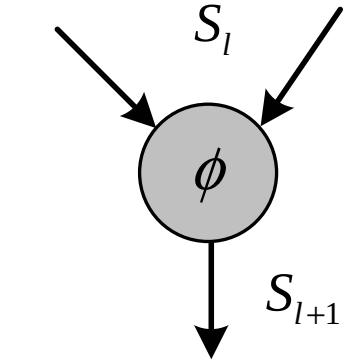
$$[undeclare(a)]^l: \quad S_{l+1} = S_l \setminus \bigcup_{\forall \left\langle x,i \right\rangle \in S_l : x = a} \left\langle a, i \right\rangle$$

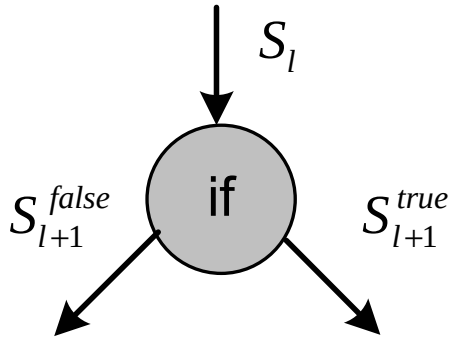
$$[\phi(\ldots)]^l : S_{l+1} = \bigcup S_l$$

$$[if\ (cond)\ s_1; s_2]^l: \quad S^{true}_{l+1} = S_l, \ S^{false}_{l+1} = S_l$$

$$S^{true}_{l+1} = \eta(S_l, cond), \ S^{false}_{l+1} = \eta\left(S_l, \overline{cond}\right)$$







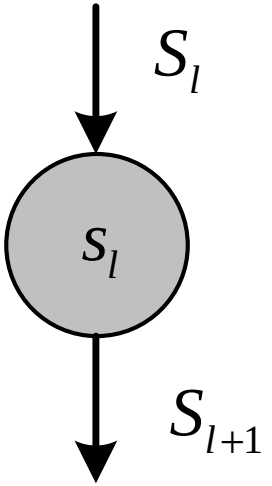
Правила интервального анализа

$$[a = C]^l : \quad S_{l+1} = S_l \setminus \bigcup_{\forall \langle x,i \rangle \in S_l : x = a} \langle a,i \rangle \cup \langle a,C \rangle \ \ (C - const)$$

$$[a = b]^l : \quad S_{l+1} = S_l \setminus \bigcup_{\forall \langle x,i \rangle \in S_l : x = a} \langle a,i \rangle \cup \bigcup_{\forall \langle x,j \rangle \in S_l : x = b} \langle a,j \rangle$$

$$[a = b + d]^l : \quad S_{l+1} = S_l \setminus \bigcup_{\forall \langle x,i \rangle \in S_l : x = a} \langle a,i \rangle \cup$$

$$\bigcup_{\forall \langle b,k \rangle, \langle d,j \rangle \in S_l : k,\, j \neq i_{nonini}} \langle a, \gamma(a,k,j) \rangle \cup \bigcup_{\exists \langle b,k \rangle, \langle d,j \rangle \in S_l : k \vee j = i_{nonini}} \langle a, i_{noninit} \rangle$$



$$\forall \langle a, (i_{\min}, i_{\max}) \rangle$$

$$\gamma(a,k,j) = \left(\min(k,j) + i_{\min}, \max(k,j) + i_{\max} \right)$$

Алгоритм анализа указателей

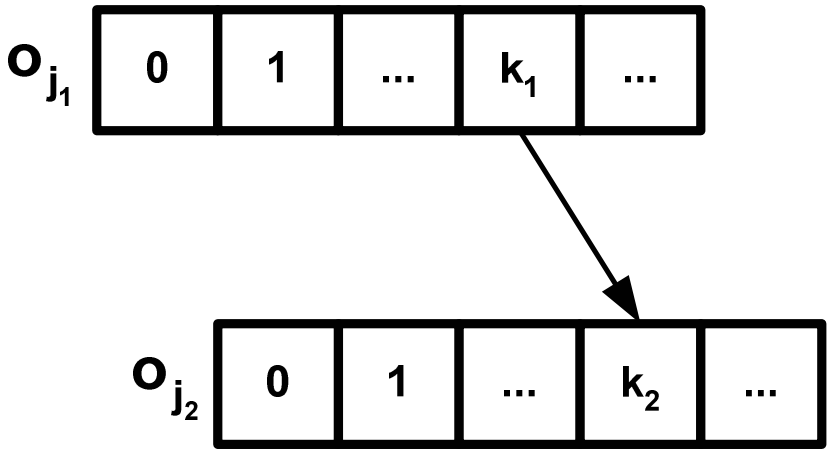
- Правила для конструкций, результат выполнения которых является указателем на объект или функцию
- Значения для переменных-указателей на объекты

$$value = \langle o, k \rangle$$

- Для учета неинициализированных переменных используется специальный объект $o_{invalid}$
- Значение NULL представляется с помощью специального объекта o_{null}

Представление состояния памяти

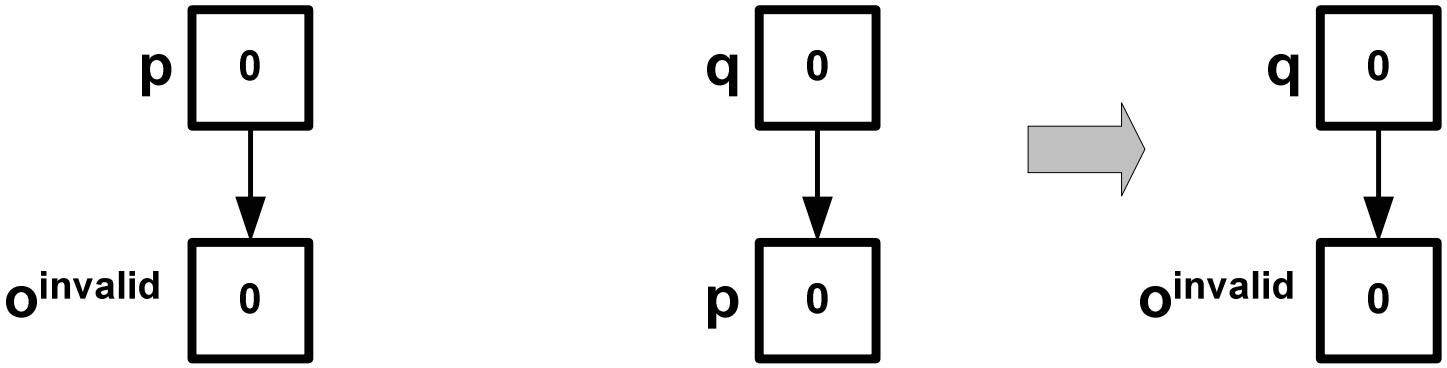
- Состояния переменных-указателей представляются в виде кортежей $\langle\langle o_1, k_1\rangle, \langle o_2, k_2\rangle\rangle$
- Представление произвольных отношений между объектами



Правила алгоритма анализа указателей

$$[declare(T * p)]^l : \quad S_{l+1} = S_l \cup \left\langle \left\langle p, 0 \right\rangle, o_{invalid} \right\rangle$$

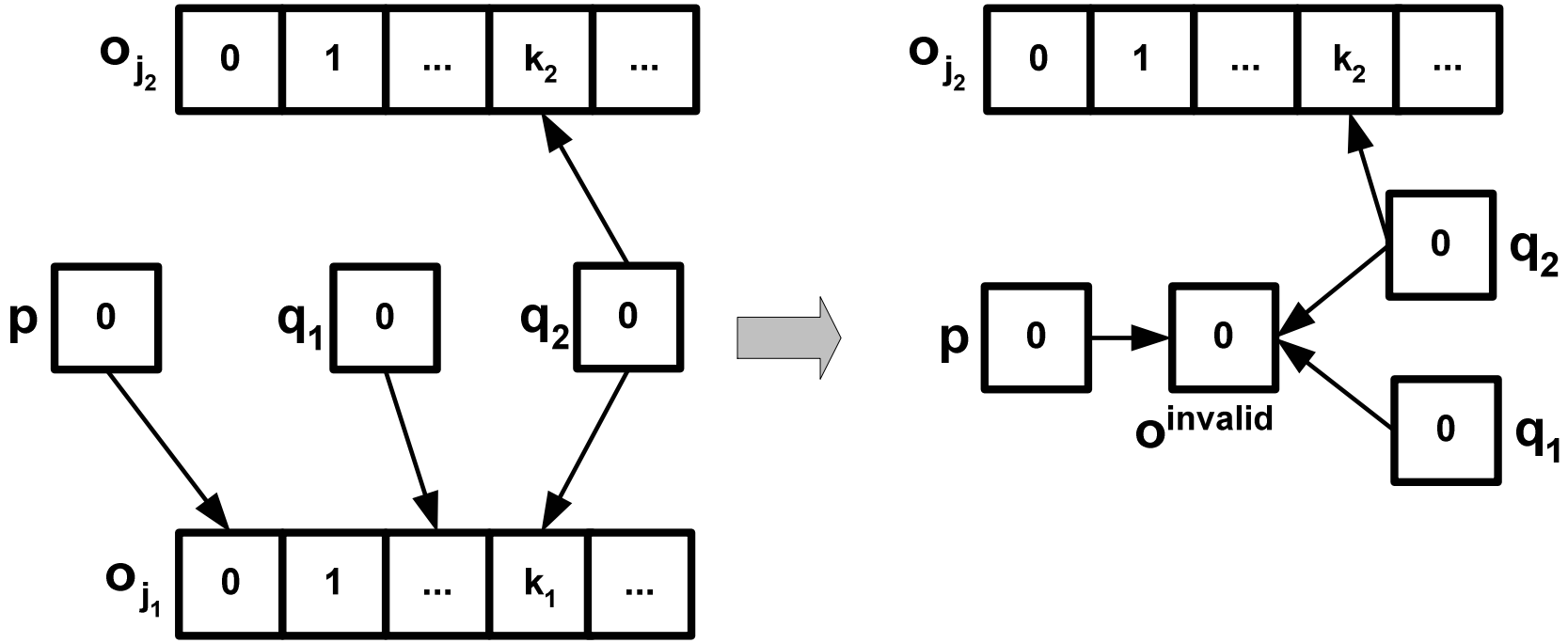
$$[undeclare(p)]^l : \quad S_{l+1} = S_l \setminus \bigcup_{\forall \left\langle \left\langle q,k_1 \right\rangle \left\langle p,0 \right\rangle \right\rangle \in S_l} \left\langle \left\langle q, k_1 \right\rangle \left\langle p, 0 \right\rangle \right\rangle \cup \bigcup_{\forall \left\langle \left\langle q,k_1 \right\rangle \left\langle p,0 \right\rangle \right\rangle \in S_l} \left\langle \left\langle q, k_1 \right\rangle, o_{invalid} \right\rangle$$



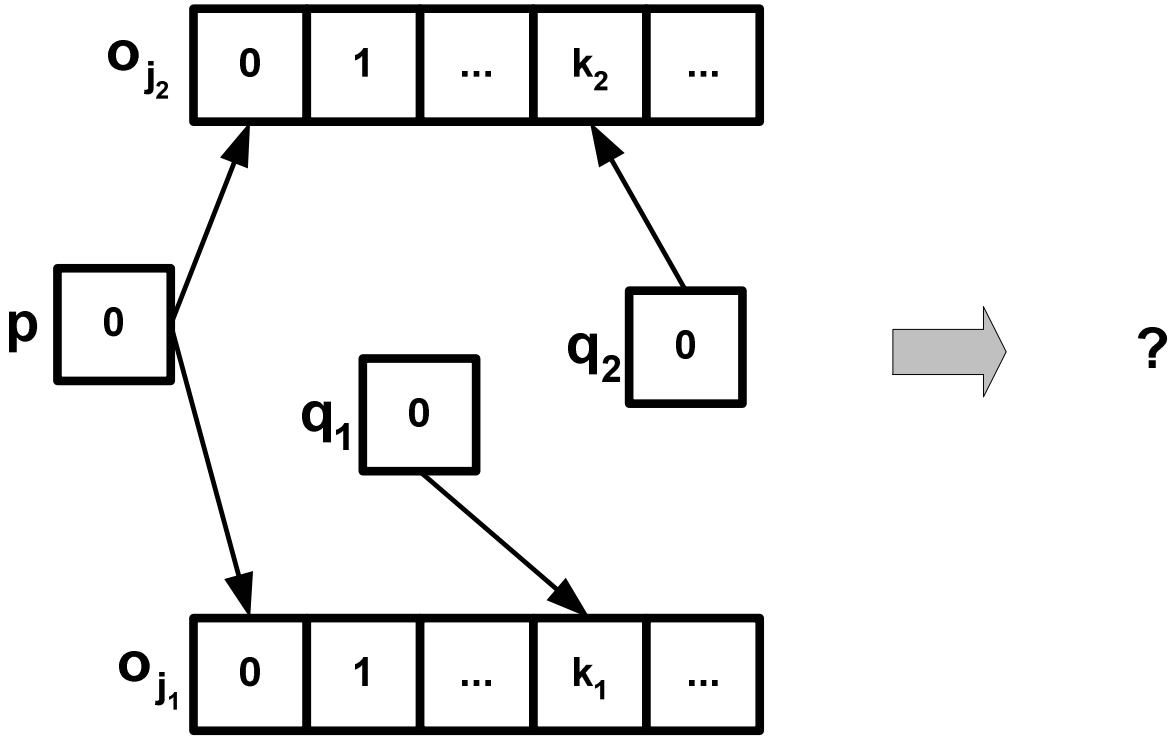
Примеры правил анализа указателей

$$[free(p)]^l: \quad S_{l+1} = S_l \setminus \bigcup_{\forall \langle\langle p,0\rangle\langle o_j,k_1\rangle\rangle \in S_l} \langle\langle p,0\rangle\langle o_{j_1},k_1\rangle\rangle \setminus \bigcup_{\forall \langle\langle p,0\rangle\langle o_{j_1},k_1\rangle\rangle, \langle\langle q,k_2\rangle\langle o_j,k_3\rangle\rangle \in S_l} \langle\langle q,k_2\rangle\langle o_j,k_3\rangle\rangle \cup$$

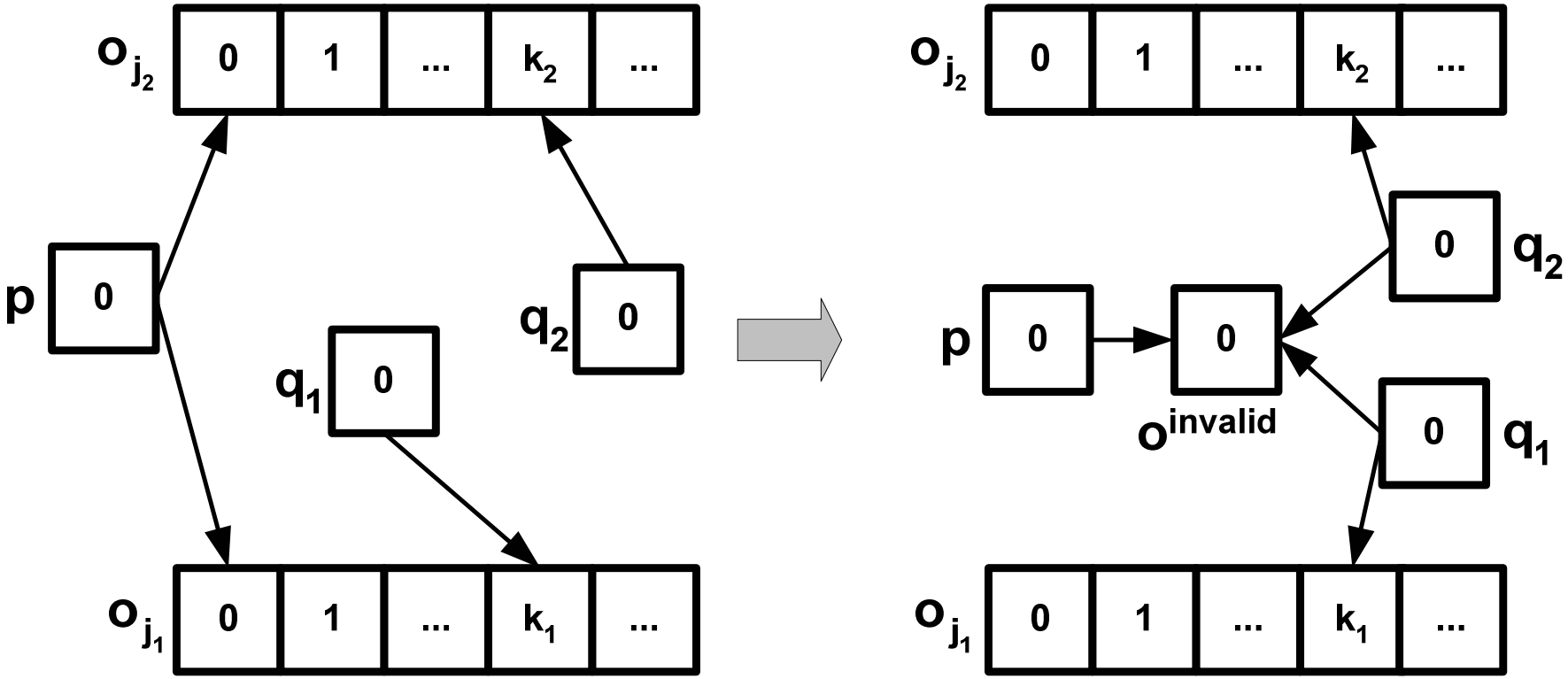
$$\langle\langle p,0\rangle, o_{invalid}\rangle \cup \bigcup_{\forall \langle\langle p,0\rangle\langle o_{j_1},k_1\rangle\rangle, \langle\langle q,k_2\rangle\langle o_j,k_3\rangle\rangle \in S_l} \langle\langle q,k_2\rangle, o_{invalid}\rangle$$



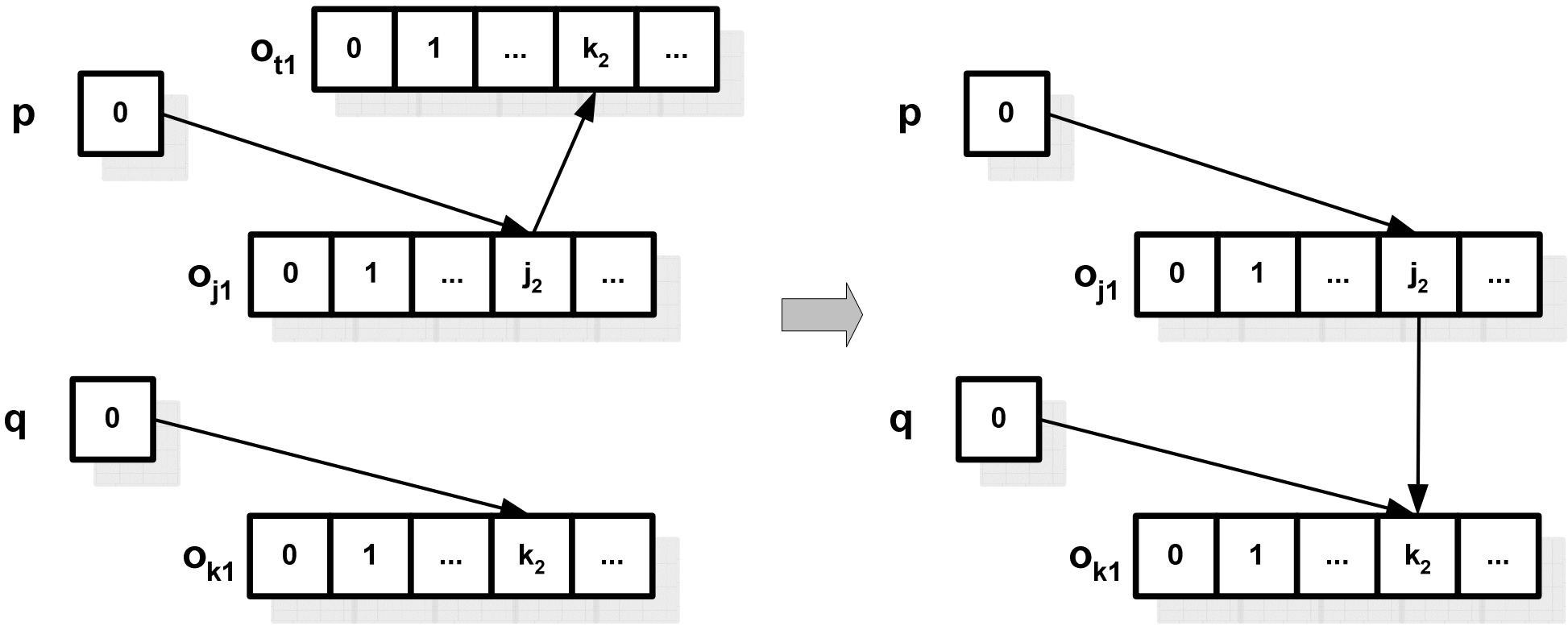
Примеры правил анализа указателей



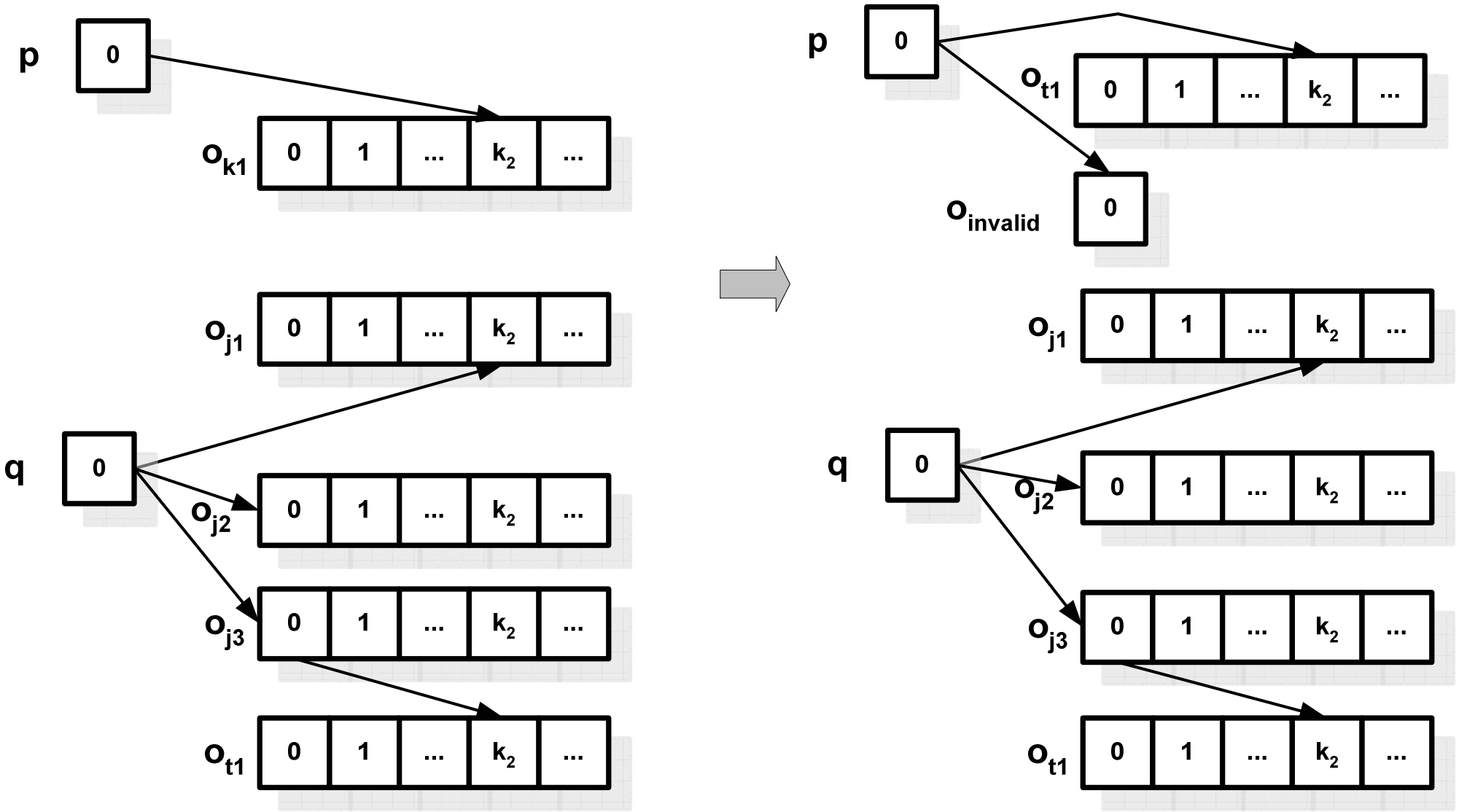
Примеры правил анализа указателей



Примеры правил анализа указателей



Примеры правил анализа указателей



Алгоритм ресурсного анализа

- Ресурсный анализ применяется для конструкций, оперирующих с ресурсами
- Примеры ресурсов: файлы, сокеты, потоки, мьютексы
- Правила ресурсного анализа определяют состояние ресурса

$$value = state,\ state = [s_1, \ldots s_n]$$

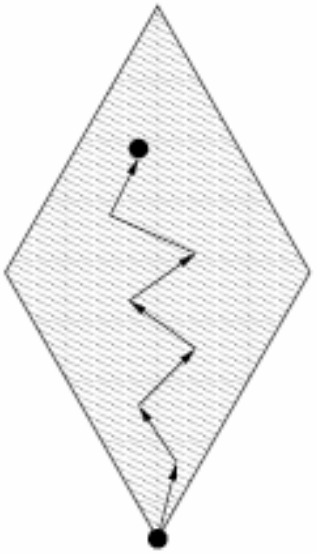
- Обнаружение нарушения протокола работы с ресурсами – нечто среднее между нефункциональными и функциональными ошибками

Решение системы уравнений

- В результате применения правил строится система уравнений
- Монотонность уравнений обеспечивается правила анализа, при каждом следующем решении уравнения результат на выходе по крайнем мере не уменьшается
- Решение системы уравнений
  - с использованием теории решеток
  - с использованием абстрактной интерпретации

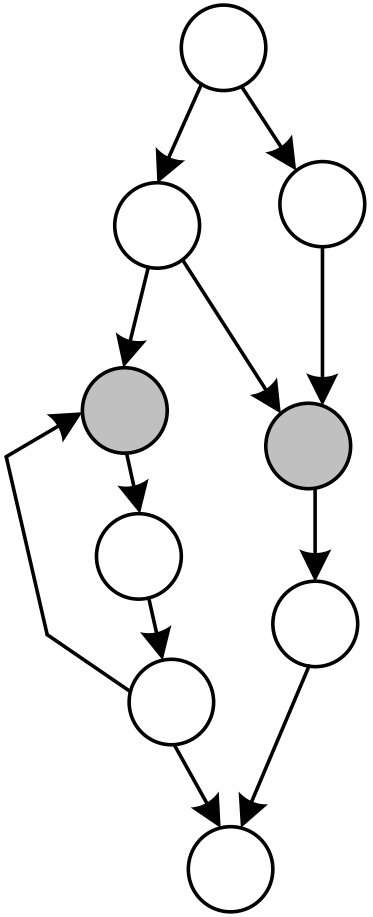
Использование теории решеток

- В узлах решетки – множества кортежей для переменных программы
- Начальное состояние ($\perp$ решетки) – соответствует пустым множествам кортежей или неинициализированным значениям переменных
- В результате решения СУ определяются множества кортежей в каждой точке программы
- Алгоритмы решения СУ
- Анализ циклов
- Техника расширения



Использование абстрактной интерпретации

- Выполняется решение уравнений в соответствии с последовательностью выполнения конструкций программы
- Последовательность решения влияет на ресурсоемкость алгоритма
- Анализ ветвлений
- Анализ циклов
- Анализ рекурсивных вызовов



Правила обнаружения дефектов

- Использование неинициализированной переменной

$$\frac{\langle a, i_{noninit} \rangle \in S_l}{[x = a]^l : defect}$$

- Повторное освобождение памяти

$$\frac{\langle \langle p,0 \rangle, \langle o_{invalid},0 \rangle \rangle \in S_l}{[\, free(p)]^l : defect}$$

- Разыменование некорректного указателя

$$\frac{\langle \langle p,0 \rangle, \langle o_{invalid},0 \rangle \rangle \in S_l}{[a = *p]^l : defect}$$

Правила обнаружения дефектов

- Выход за границы объекта

$$\frac{\langle\langle p,0\rangle,\langle o_j,k\rangle\rangle \in S_l : (k<0)\vee(k\geq o_j.size)}{[a=*p]^l : defect}$$

- Арифметические операции с указателями на разные объекты

$$\frac{\langle\langle p,0\rangle,\langle o_{j1},k_1\rangle\rangle,\langle\langle q,0\rangle,\langle o_{j2},k_2\rangle\rangle \in S_l : o_{j1}\neq o_{j2}}{[a=p-q]^l : defect}$$

- Утечка динамической памяти

Способы снижение ресурсоемкости

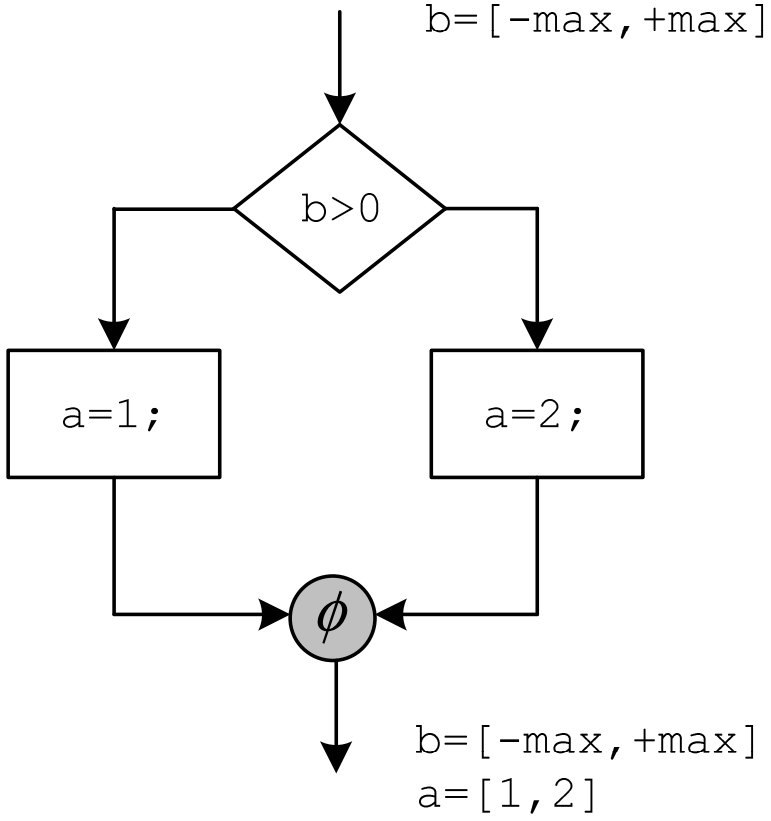
- СА при высокой точности и полноте имеет высокую ресурсоёмкость
  - большое число анализируемых путей
  - большая длина путей
  - большой объем данных, связанных с каждым путём

Способы снижение ресурсоемкости

- Сокращение числа путей
  - объединение состояний объектов программы в *ϕ*-функциях и далее проведение общего анализа
- Сокращение длины путей
  - ограничение числа итераций циклов
  - ограничение глубины стека вызова функций
  - замена некоторых функций их аппроксимациями
- Уменьшение количества данных
  - ограничение размера выделяемых и анализируемых объектов
  - компактное представления информации

Способы повышения точности

- Частичное неслияние путей, необходимо определить эффективный критерий неслияния
  - ограничение на число отдельных путей
  - время жизни пути
- Анализ зависимостей между значениями переменных

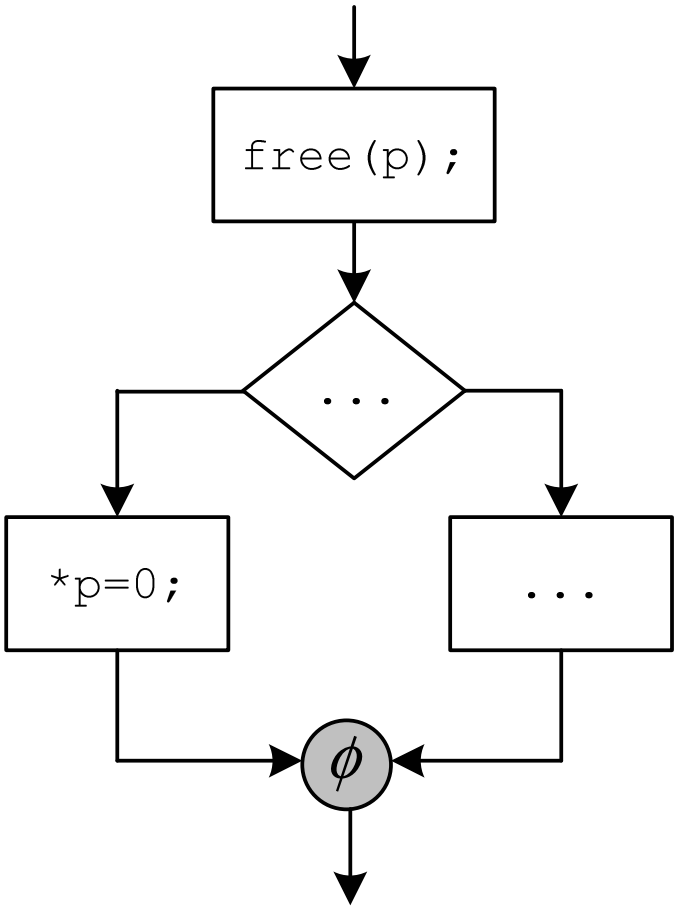


Пример работы зависимостей

```
int fd;
int f() {
    if (...) {
        return -1;
    fd = fopen(fname, “rw”);
    return 0;
}
int main() {
    ...
    int result = f();    // fd == file, result == 0
    if (result < 0) {
         exit;
    }
    fprintf(fd, ...);   // fd =file
```

Обнаружение множественных дефектов

- Программа может содержать несколько дефектов
- Возникновение наведенных дефектов
- Необходимо устранить влияние обнаруженных дефектов – в общем случае невозможно
- Удаление путей, на которых обнаружен дефект



Анализ программ с неполными исходными кодами

- Библиотечные функции
- **Аннотации** – упрощенных описания поведения отсутствующих функций, результаты работы функции
  - возвращаемое значение
  - переменные, изменяемые через указатели
  - изменяемые глобальные переменных
- Применение аннотация для определения значений входных данных, значений внутренних переменных и т.п.

Средства статического анализа

- Средства для обнаружения широкого класса дефектов
  - Coverity Prevent SA
  - Klockwork Insight
  - Mathworks PolySpace
  - IBM RSA
  - Fortify SCA
  - **Microsoft CA**
  - Frama-C
  - SPlint
- Отечественные разработки
  - SVaCE Detector
  - Viva64, VivaMP
  - **AEGIS**

Открытый сервис www.digiteklabs.ru/aegis

Результаты анализа

Длительность анализа: 6 seconds and 882 milliseconds

Путь	Код	Дефекты, контексты
	01: // Several for-cycles with function call	
	02:	
	03: #include <stdio.h>	
	04:	
	05: int arr[2];	
	06:	
	07: int f1(float par) {	
	08: int tmp = par;	
	09: printf("%d \n",tmp);	
*(3)	10: return arr[tmp]=tmp; // BUF-02	Разыменование указателя int * arr, tmp выведенного за границу объекта (BUF-02)