



Статический анализ программного обеспечения

**Санкт-Петербургский государственный политехнический
университет
Кафедра компьютерных систем и программных технологий
Лаборатория программно-аппаратных разработок**

Ицыксон В.М.

Санкт-Петербург 2011

План доклада

- Введение
- Качество ПО
- Методы обеспечения качества
- Статический анализ
 - Модели программ
 - Алгоритмы статического анализа
 - Обнаружение дефектов
 - Аннотирование
 - Статические анализаторы
- Место статического анализа
- Заключение

Тенденции индустрии разработки ПО

- ❑ Объем программ растет
- ❑ Время разработки новых версий ПО сокращается
- ❑ Все большее число задач решается программно
- ❑ ПО все больше используется при решении критически важных задач
- ❑ Существенная часть ПО является свободной и поставляется "as is"

Известные примеры программных ошибок

- **США, 1962 год.** Гибель несущего аппарата "Маринер-1". Причина – ошибка в одном символе программы
 - $DO\ 100\ I = 1, 10$
 - $DO100I = 1.10$
- **США, 1987 год.** Ускоритель Therac-25. Переоблучение пациентов онкокlinik. Причина – ошибка «race condition»
- **США, 1991 год.** Комплекс Patriot. Погибло 28 чел. Причина – ошибка округления
- **Европа, 1996 год.** Ракета Ариан-5. Ущерб 7 млрд. \$. Причина – использование унаследованного кода

Известные примеры программных ошибок

- **США, 2003 год.** Сбой в энергосистеме (Blackout). Ущерб 7-10 млрд.\$. Причина – ошибка «race condition»
- **Израиль.** Сбой навигационной системы самолетов F16 при полетах над Мертвым морем.
 - Высотомер выдавал значение ≤ 0 .
 - Ошибка деления на ноль (или переполнение)
- **Голландия, 2000 год.** Остановка доменной печи 29 февраля. Гибель 6 человек. Ошибка в процедуре расчета даты.
- ...

Качество программного обеспечения

- Характеристики качества ПО (по ISO 9126)
 - Функциональные возможности (Functionality)
 - Надежность (Reliability)
 - Практичность (Usability)
 - Эффективность (Efficiencies)
 - Сопровождаемость (Maintainability)
 - Мобильность (Portability)

Качество программного обеспечения

- Характеристики, относящиеся к качеству функционирования
 - **Функциональные возможности (Functionality)**
 - **Надежность (Reliability)**
 - Практичность (Usability)
 - Эффективность (Efficiencies)
 - Сопровождаемость (Maintainability)
 - Мобильность (Portability)

Методы обеспечения качества

- Методы, направленные на проектирование качественного ПО
 - Формальные спецификации
 - Синтез ПО на основе спецификаций и моделей
 - И т.п.
- **Методы, направленные на обеспечение качества существующего ПО**

Методы обеспечения качества

- **Динамические методы**
 - Подразумевают запуск ПО
- **Статические методы**
 - Не используют запуск ПО
 - Используют проектные артефакты (исходные коды, спецификации, аннотации и т.п.)
- **Комбинированные методы**

Динамические методы обеспечения качества

- Тестирование**
 - Ручное
 - Автоматизированное
 - Автоматическое
- Профилирование
- Динамический анализ
 - Мониторинг
 - Анализ трасс исполнения
- Контрактное программирование
- ...

Недостатки динамических методов

- Проверяют ограниченное число трасс выполнения
- Проверяют конечный набор входных данных
- Проверяют работоспособность в ограниченном числе окружений
- Число тестов, позволяющих исчерпывающе проверить программную систему, потенциально бесконечно!
- **Тестирование не гарантирует отсутствие ошибок!**
- **Тестирование может только показать наличие ошибок!**

*«Тестирование программ может служить для доказательства наличия ошибок, но никогда не докажет их отсутствия!»
Э.Дейкстра. Заметки по структурному программированию*

Статические методы обеспечения качества

- Формальные инспекции, аудит
- Формальная верификация
 - Дедуктивная верификация
 - Model checking
- **Статический анализ**
- Трансформации
 - Рефакторинги
 - Модификации

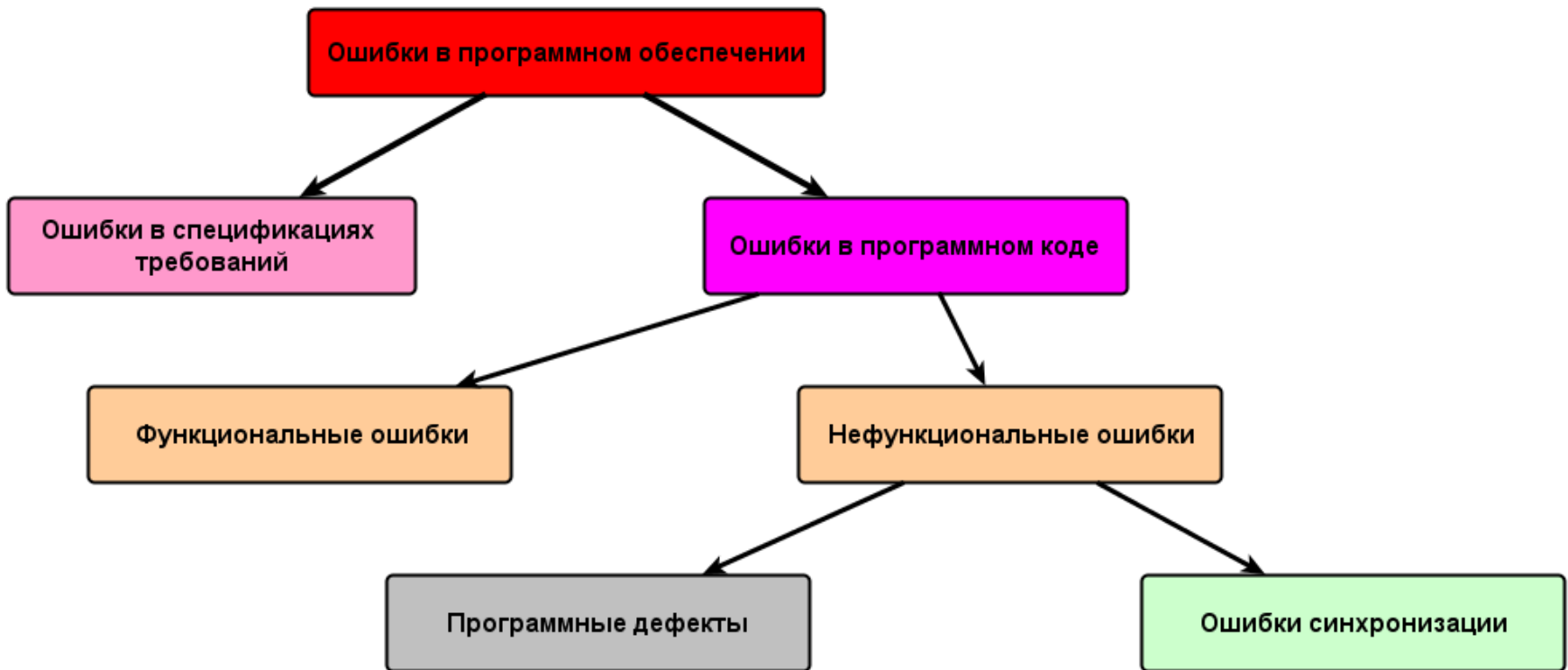
Статический анализ

- Использует исходный код ПО для анализа
- Применяется для
 - Форматирования программ
 - Вычисления программных метрик
 - Оптимизации программ
 - Распараллеливания программ
 - Преобразования программ
 - Обфускации/деобфускации программ
 - **Обнаружения дефектов**
 - ...

Статический анализ для обнаружения дефектов

- Использует исходный код ПО для анализа
- Теоретически позволяет проанализировать **все** возможные трассы исполнения
- Позволяет проанализировать **все** наборы входных данных
- Может быть **полностью** автоматизирован
- Позволяет обнаружить *нефункциональные* дефекты

Ошибки в программном обеспечении



Ошибки в программном коде

- Функциональные ошибки
 - являются следствием нарушения спецификаций
- Нефункциональные ошибки
 - не являются нарушением явных требований спецификаций
 - являются:
 - ошибками кодирования
 - нарушениями правил языка программирования
 - следствием некорректной работы с библиотеками
 - и т.п.
 - могут приводить к неправильному функционированию программы!

Программные дефекты

- Ошибки выхода за пределы объектов
 - выход за границу массива
 - переполнение буфера
- Использование неинициализированных объектов
- Ошибки работы с указателями
 - работа с нулевым указателем
 - работа с некорректным указателем
- Арифметические ошибки
 - переполнение
 - деление на ноль
- Ошибки форматных строк
- Ошибки работы с ресурсами
 - утечки ресурсов
 - повторное освобождение ресурсов
 - ошибки протокола работы с ресурсами
- «Мертвый» код

Статистика дефектов в open source проектах

- Отчет компании Coverity
- Проанализировано 280 проектов
- Объем - 60 млн. строк
- Найдено 38453 дефектов
- Плотность дефектов: 0.25-1 дефекта на 1000 строк кода

| Defect Type | Frequency |
|--|-----------|
| Разыменование нулевого указателя | 27.81% |
| Утечка ресурсов | 23.34% |
| «Мертвый» код | 9.71% |
| Небезопасное использование возвращаемого значения | 9.20% |
| Использование значения до проверки (NULL) | 8.35% |
| Использование объекта после освобождения | 5.91% |
| Переполнение буфера | 6.00% |
| Чтение неинициализированной переменной | 8.41% |
| Другие | 1.27% |

Примеры программных дефектов

- ▶ Примеры ошибок работы с указателями
 - ▶ разыменованние 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

- ▶ Примеры ошибок работы с неинициализированными объектами
 - ▶ использование неинициализированной переменной
 - ▶ разыменованное неинициализированное указатель

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

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

Примеры программных дефектов

- ▶ Примеры ошибок работы с ресурсами
 - ▶ утечка памяти
 - ▶ повторное освобождение ресурса

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

```
int main() {  
    FILE f = fopen(fname, "w");  
    ...  
    fclose(f);  
    ...  
    if (flag){  
        fclose(f);  
    }  
    ...  
}
```

Примеры программных дефектов

- ▶ Примеры ошибок работы со строками
 - ▶ использование в качестве форматной строки неконтролируемого значения
 - ▶ переполнение буфера при операции со строками

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

```
int main()
{
    char src[10] = "Hello";
    char dest[5];
    ...
    strcpy(dest, src);
}
```

Проявление дефектов

- Зависания
- Сбои
- Падения
- Деградация производительности
- Неправильное функционирование
- Без проявления!
- ...

Языки программирования

- Для каких языков наиболее актуально обнаружение дефектов?
- Почему С и С++?
 - Являются вместе с Java наиболее популярными языками
 - Наибольшее число дефектов и уязвимостей имеются в программах на С и С++
 - Язык С (и С++) является языком системного программирования и используется во многих критически важных приложениях
 - Методы статического анализа наиболее проработаны для С/С++
- А что с другими языками?

TIOBE Programming Community Index (Jun 2011)

| Position Jun 2011 | Position Jun 2010 | Delta in Position | Programming Language | Ratings Jun 2011 | Delta Jun 2010 | Status |
|-------------------|-------------------|-------------------|----------------------|------------------|----------------|--------|
| 1 | 2 | ↑ | Java | 18.580% | +0.62% | A |
| 2 | 1 | ↓ | C | 16.278% | -1.91% | A |
| 3 | 3 | = | C++ | 9.830% | -0.55% | A |
| 4 | 6 | ↑↑ | C# | 6.844% | +2.06% | A |
| 5 | 4 | ↓ | PHP | 6.602% | -2.47% | A |
| 6 | 5 | ↓ | (Visual) Basic | 4.727% | -0.93% | A |
| 7 | 10 | ↑↑↑ | Objective-C | 4.437% | +2.07% | A |
| 8 | 7 | ↓ | Python | 3.899% | -0.20% | A |
| 9 | 8 | ↓ | Perl | 2.312% | -0.97% | A |
| 10 | 20 | ↑↑↑↑↑↑↑↑↑↑ | Lua | 2.039% | +1.55% | A |
| 11 | 12 | ↑ | JavaScript | 1.501% | -0.58% | A |
| 12 | 11 | ↓ | Ruby | 1.484% | -0.61% | A |
| 13 | 9 | ↓↓↓ | Delphi/Object Pascal | 1.070% | -1.50% | A |
| 14 | 16 | ↑↑ | Lisp | 0.935% | +0.28% | A |
| 15 | 15 | = | Pascal | 0.731% | +0.00% | A |

Проблемы языков C/C++

- Опасные конструкции
 - Язык C
 - Макроопределения
 - Указатели
 - Массивы
 - Адресная арифметика
 - Указатели на функции
 - Приведения типов
 - Объединения (union)
 - Язык C++
 - Полиморфизм
 - Исключения
 - ...
- Неоднозначная семантика

Общая схема обнаружения дефектов



Модели программ. Требования

- Полнота представления объектов программы
- Восстанавливаемость
 - Идеально: возможность полного восстановления исходного кода
 - Желательно: связь с исходным кодом
- Эффективность поиска объектов

Модели программ. Типы

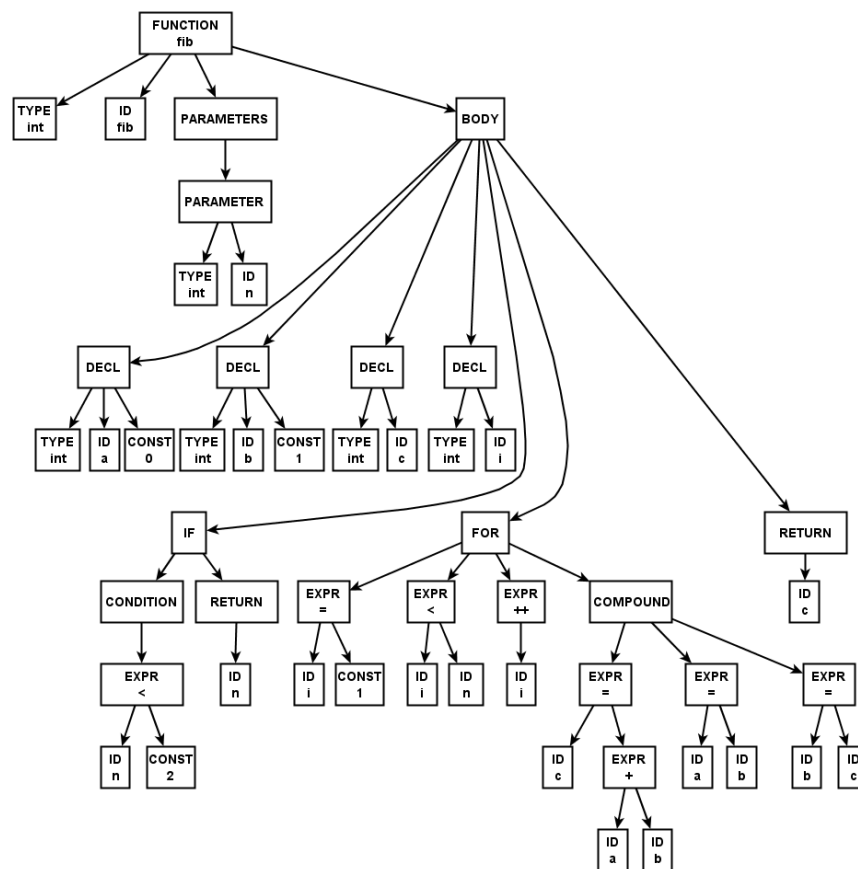
- Структурные
 - Синтаксическое дерево разбора
 - Абстрактное синтаксическое дерево (AST – abstract syntax tree)
- Поведенческие
 - Граф потока управления (CFG – control flow graph)
 - Граф зависимостей по данным (DDG – data dependence graph)
 - Граф зависимостей программы (PDG – program dependency graph)
 - Представление в виде однократного статического присваивания (SSA - static single assignment)
- Гибридные
 - Абстрактный семантический граф (ASG - abstract semantic graph)

Модели программ: AST

```

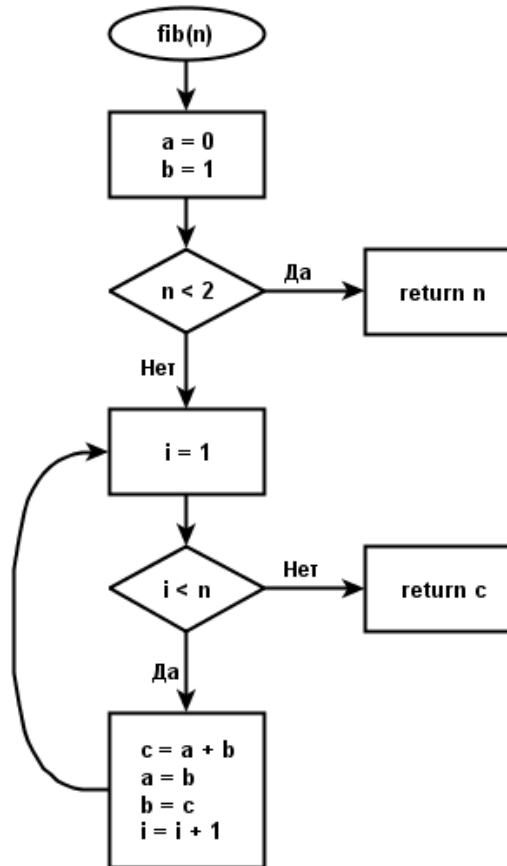
int fib(int n)
{
    int a = 0;
    int b = 1; c, i;
    if (n < 2) return n;
    for (i = 1; i < n; i++)
    {
        c = a + b;
        a = b;
        b = c;
    }
    return c;
}

```



Модели программ: CFG

```
int fib(int n)
{
    int a = 0;
    int b = 1; c, i;
    if (n < 2) return n;
    for (i = 1; i < n; i++)
    {
        c = a + b;
        a = b;
        b = c;
    }
    return c;
}
```

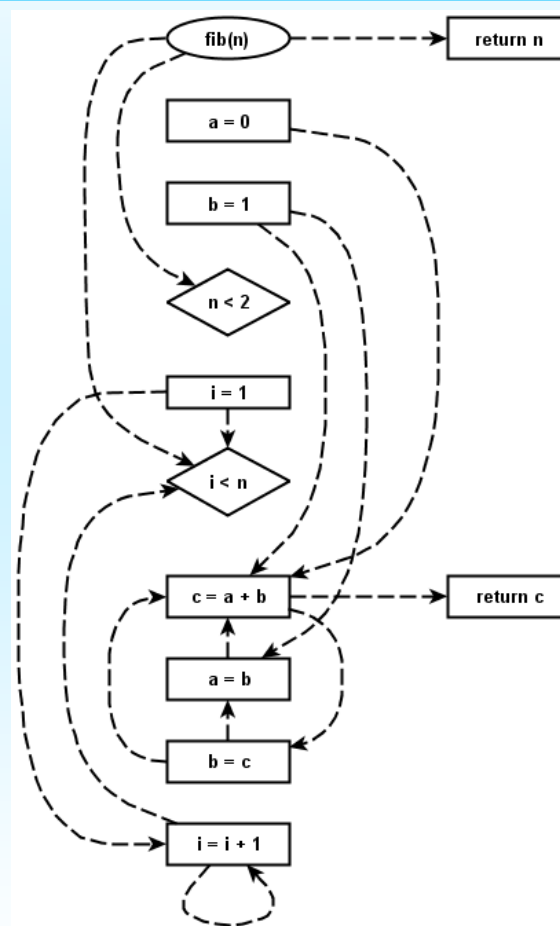


Модели программ: DDG

```

int fib(int n)
{
    int a = 0;
    int b = 1; c, i;
    if (n < 2) return n;
    for (i = 1; i < n; i++)
    {
        c = a + b;
        a = b;
        b = c;
    }
    return c;
}

```

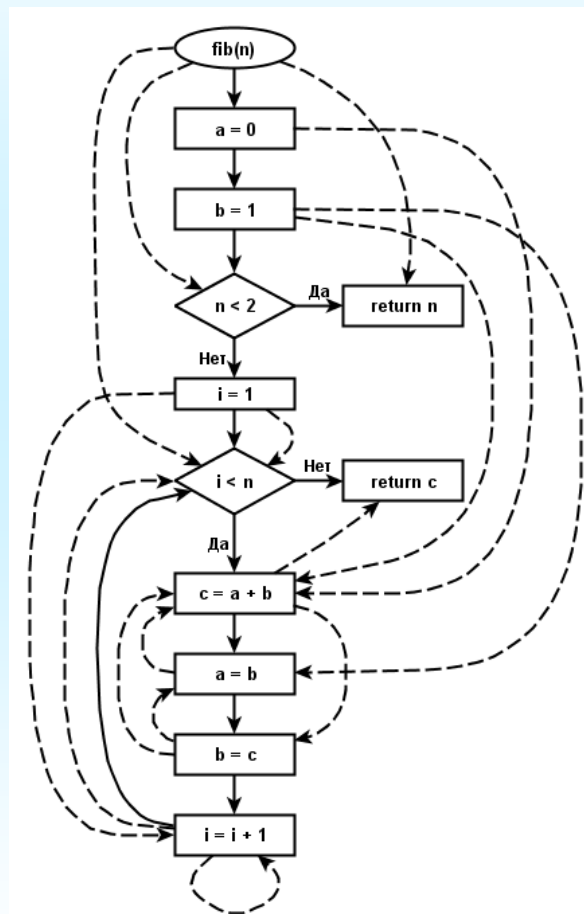


Модели программ: PDG

```

int fib(int n)
{
    int a = 0;
    int b = 1; c, i;
    if (n < 2) return n;
    for (i = 1; i < n; i++)
    {
        c = a + b;
        a = b;
        b = c;
    }
    return c;
}

```

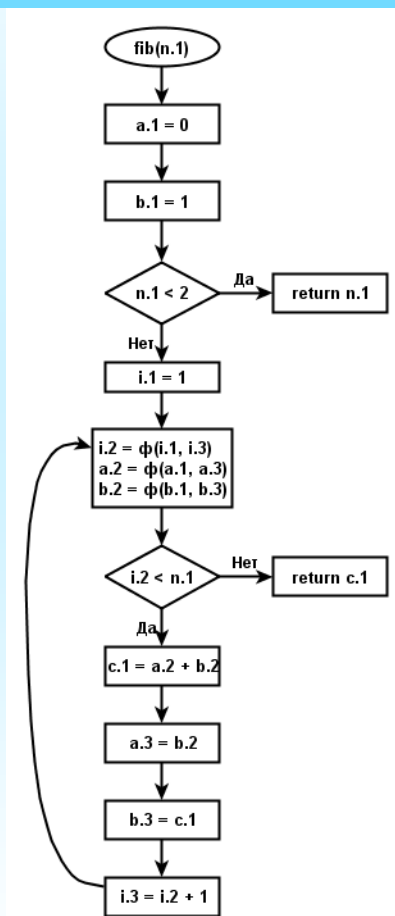


Модели программ: SSA

```

int fib(int n)
{
    int a = 0;
    int b = 1; c, i;
    if (n < 2) return n;
    for (i = 1; i < n; i++)
    {
        c = a + b;
        a = b;
        b = c;
    }
    return c;
}

```

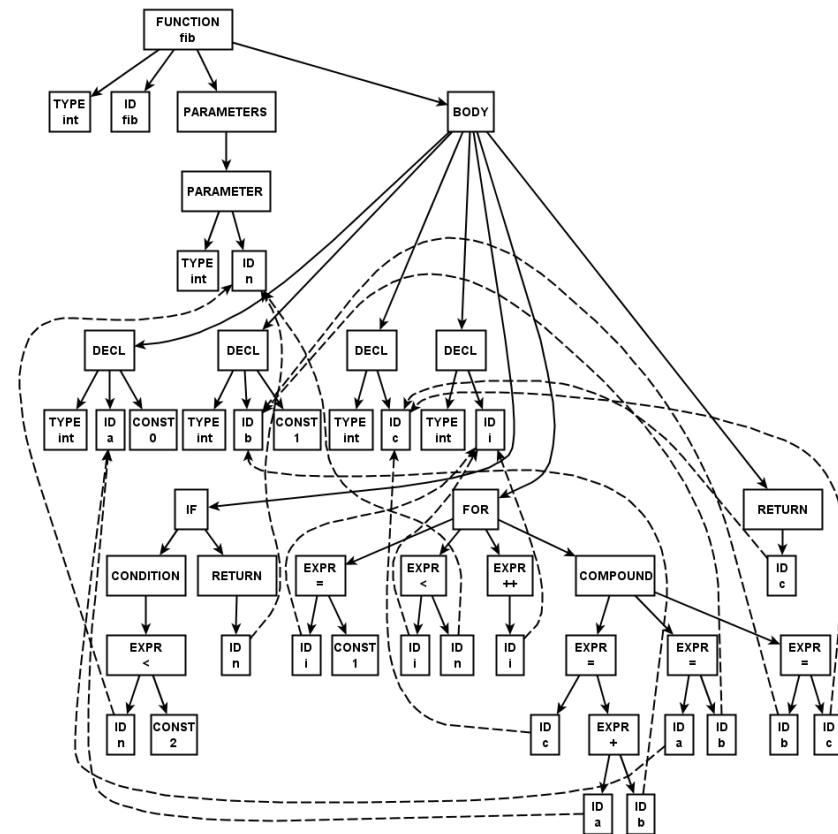


Модели программ: ASG

```

int fib(int n)
{
  int a = 0;
  int b = 1; c, i;
  if (n < 2) return n;
  for (i = 1; i < n; i++)
  {
    c = a + b;
    a = b;
    b = c;
  }
  return c;
}

```



Модели программ для СА

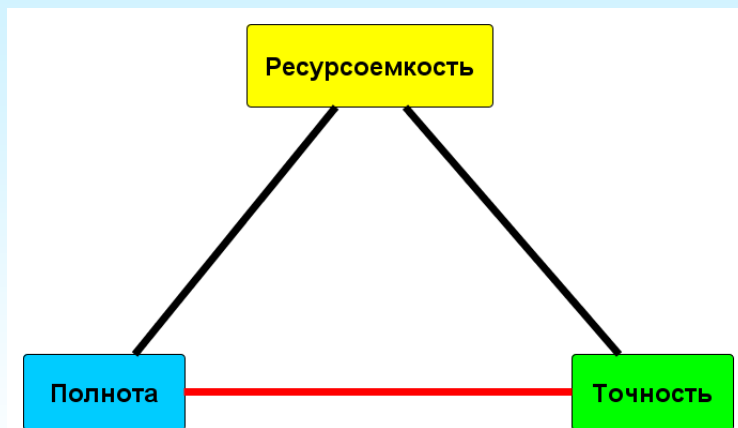
- Для простейшего статического анализа достаточно структурных моделей (AST)
- Для анализа потока данных необходим учет семантики:
 - CFG, SSA, PDG
 - ASG
- Для анализа зависимостей
 - DDG, PDG, SSA
 - ASG

Модели программ. Построение

- ❑ Построение парсера вручную
- ❑ Использование генераторов парсеров (JavaCC, ANTLR, yacc, etc)
- ❑ Использование готовых парсеров (Elsa)
- ❑ Использование парсеров в составе средств разработки (NetBeans, Eclipse CDT)
- ❑ Использование Front-End компиляторов (gcc)
- ❑ Использование фреймворков для статического анализа (SUIF, CIL, LLVM, etc)

Показатели эффективности обнаружения

- **Полнота (C)** – доля найденных дефектов среди всех
 - $C = T / A$
- **Точность (P)** – доля истинных дефектов среди найденных
 - $P = T / S$
- **Ресурсоемкость**



Общая схема обнаружения дефектов



Подходы к организации СА

- На основе систем уравнений
- На основе ограничений
- **На основе абстрактной интерпретации (abstract interpretation)**
- На основе систем типов и эффектов (type and effect systems)

СА на основе систем уравнений

- Состояние программы – совокупность состояний всех объектов программы в какой-либо точке
- Цель – вычислить все возможные состояния программы во всех точках
- Каждому оператору ставится в соответствие уравнение, связывающее состояние до выполнения и после
- Уравнения оперируют состояниями программы
- Решение системы уравнений – множество состояний программы во всех точках
- Обнаружение дефектов происходит на основе проверки вычисленных состояний

Построение системы уравнений

Элемент модели

Соответствующее уравнение

int a;

$$S^l = S^{l-1} \cup \langle a, \text{noninit} \rangle$$

a = C;

$$S^l = S^{l-1} \setminus U_i \langle a, i \rangle \cup \langle a, C \rangle$$

a = b;

$$S^l = S^{l-1} \setminus U_i \langle a, i \rangle \cup U_j \langle a, b_j \rangle$$

$\varphi(\dots)$

$$S^l = U_i S^i$$

if(cond) S^{true} else S^{false}

$$S^{\text{true}} = S^{l-1} \cap \text{cond}$$

$$S^{\text{false}} = S^{l-1} \cap \neg \text{cond}$$

Решение системы уравнений

- Система решается на решетке
- Уравнения могут решаться в любой последовательности
- Завершаемость процесса решения обуславливается
 - Монотонностью уравнений
 - Конечностью системы уравнений

Абстрактная интерпретация

- Аппроксимация конкретной семантики программы – приближенной (или абстрактной) семантикой
- Интерпретация программы – выполнение ее над конкретным доменом данных
- Абстрактная интерпретация – интерпретация программы над выбранным абстрактным доменом
- Абстрактная интерпретация позволяет снизить размерность задачи, не сильно снижая качество решения
- Patrick Cousot, 1975 год

Алгоритмы статического анализа

- Интервальный анализ
- Анализ указателей
- Ресурсный анализ
- Строковый анализ
- Анализ зависимостей
- ...

Интервальный анализ

- Состояние переменных представляются интервальными значениями
 - $a = [10; 20]$
 - $b = (-40; 20) \cup (30; 100)$
 - $c = [0; \infty)$
- Вычисление состояния после выполнения оператора языка производится на основе интервальной арифметики

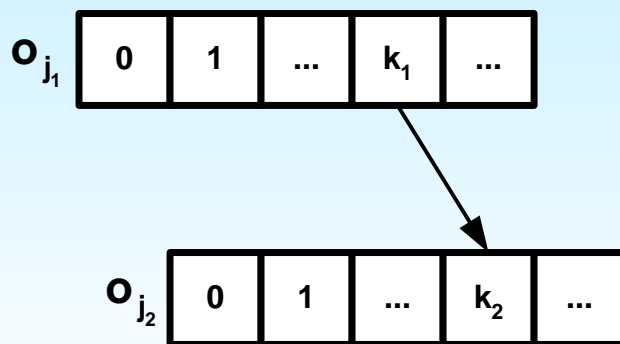
Интервальный анализ.

Пример

| | |
|--------------------------------|---|
| <code>int a;</code> | <code><a, noninit></code> |
| <code>a=10;</code> | <code><a, [10:10]></code> |
| <code>int b = f(...);</code> | <code><a, [10:10]>, <b, (-∞; ∞)></code> |
| <code>if (b < 100) {</code> | |
| <code>...</code> | <code><a, [10:10]>, <b, (-∞; 100)></code> |
| <code>a = a + b;</code> | <code><a, [-∞+10:110]>, <b, (-∞; 100)></code> |
| <code>}</code> | |
| <code>else {</code> | |
| <code>...</code> | <code><a, [10:10]>, <b, [100; ∞)></code> |
| <code>a = b / 2;</code> | <code><a, [50: ∞/2]>, <b, [100; ∞)></code> |
| <code>}</code> | |

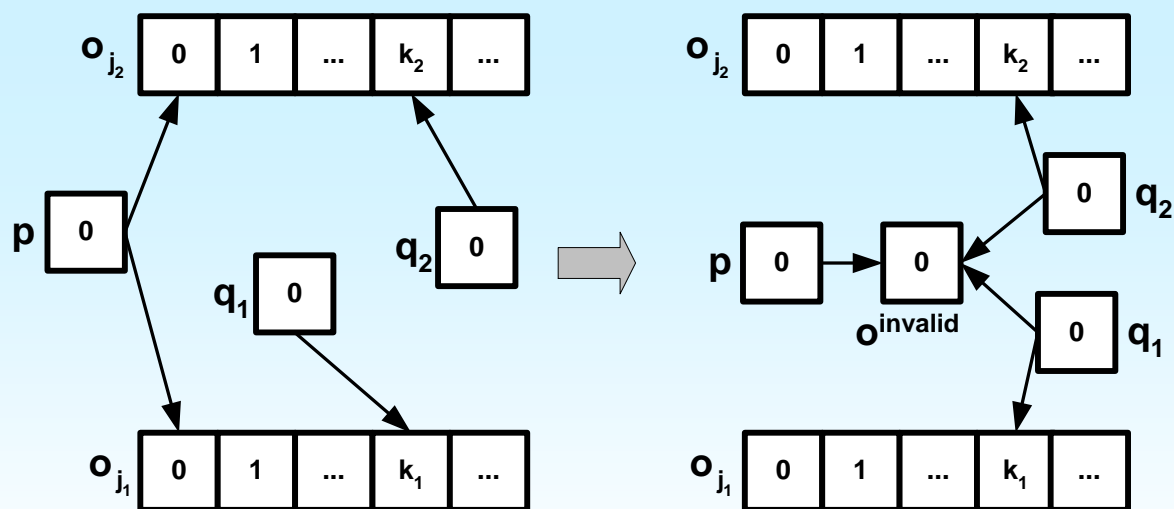
Анализ указателей

- Отличие от интервального анализа:
 - хранятся не интервалы значений, а списки связей "points-to"
 - значения одних объектов могут влиять на другие



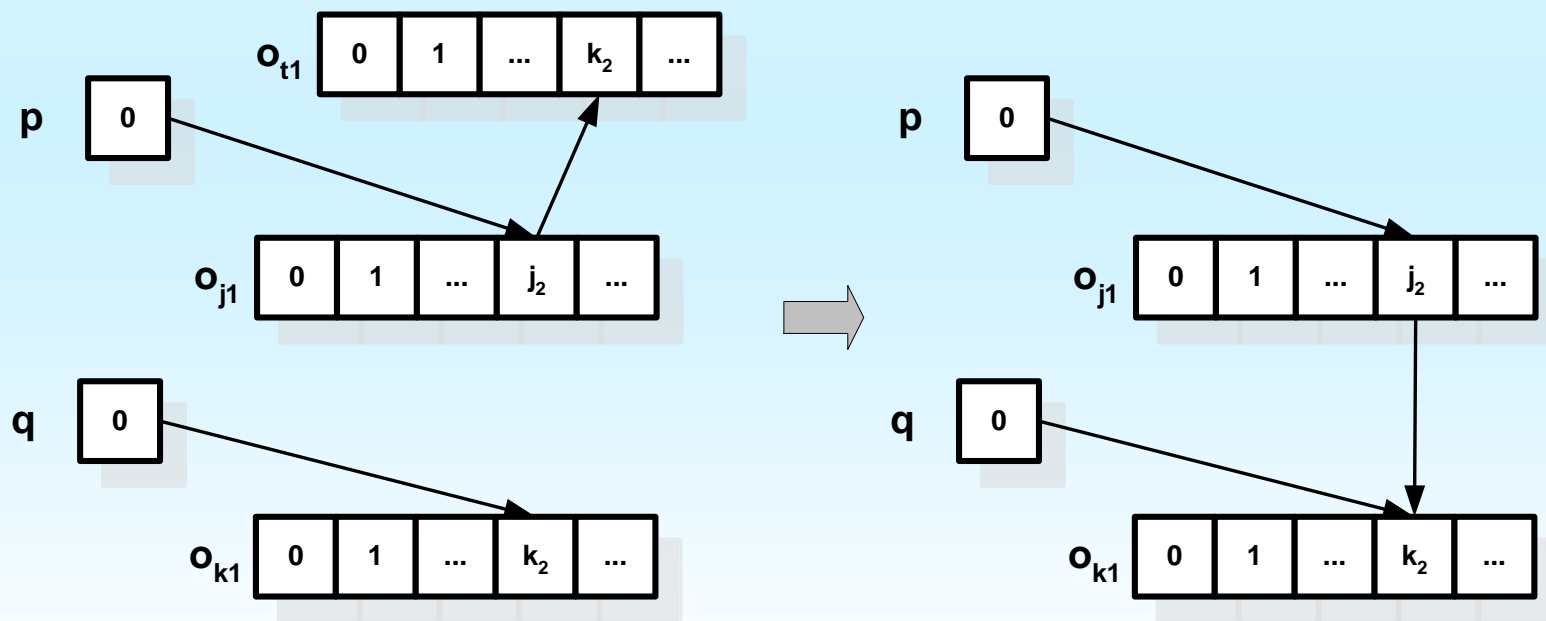
Анализ указателей. Пример

- p , q_1 и q_2 – указатели на динамическую память
- Пример для конструкции `free(p);`



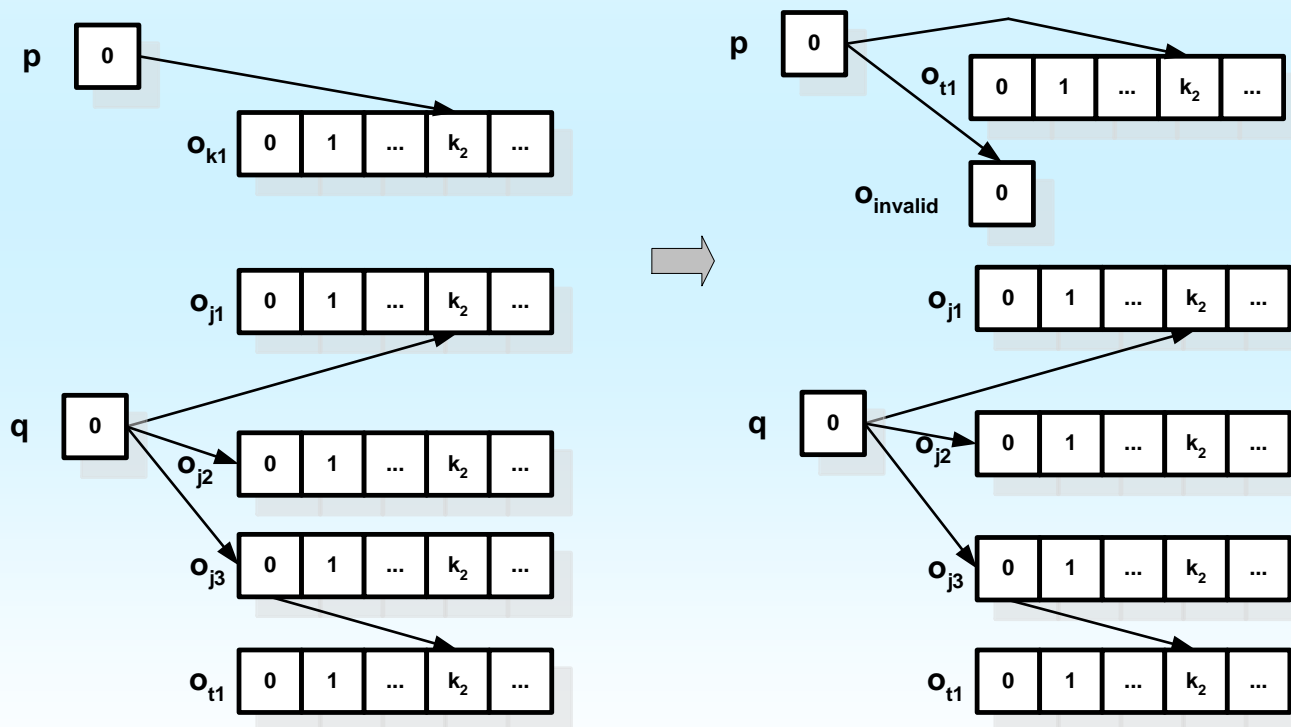
Анализ указателей. Пример

- p, q – указатели
- Пример для конструкции $*p = q$



Анализ указателей. Пример

- p, q – указатели
- Пример для конструкции $p = *q$



Ресурсный анализ

- Ресурсы – объекты программы, имеющие свой жизненный цикл
- Примеры ресурсов:
 - Выделенная динамическая память
 - Файлы
 - Семафоры
 - Мьютексы
 - Сокеты
 - Потоки
 - Нити
 - ...
- Обычно ресурсы характеризуются состоянием и атрибутами
- Каждый тип ресурса изменяет свое состояние по определенным правилам

Ресурсный анализ. Объект в куче

- Правила использования
 - Создается ($p = \text{malloc}(\dots)$)
 - Используется ($*p$)
 - Освобождается ($\text{free}(p)$)
- Примеры дефектов
 - Использование не созданного объекта
 - Повторное освобождение
 - Утечка

Ресурсный анализ. Файл

- Правила использования
 - Открыть (`f = fopen(..., "r")`)
 - Читать/писать (`fprintf(f, ...)`)
 - Заккрыть (`close(f)`)
- Примеры дефектов
 - Чтение из не открытого файла
 - Попытка писать в файл открытый на чтение
 - ...

Ресурсный анализ

- Отличие ресурсов от других объектов программы:
 - Управляются операционной системой
 - Управление сменой состояний и атрибутов происходит через функции API
- Необходимо анализировать не конструкции ЯП, а библиотечные вызовы
- Необходима информация о семантике библиотечных вызовов:
 - Семантика зашита в анализатор
 - Семантика описывается через аннотирование библиотек

Общая схема обнаружения дефектов



Алгоритмы статического анализа

- По масштабу
 - Внутрипроцедурный
 - Межпроцедурный
- По консервативности
 - Пессимистический (консервативный) анализ
 - Оптимистический анализ

Масштаб анализа

- Внутривпроцедурный анализ
 - Каждая функция анализируется отдельно
 - Информация не передается между функциями
 - Ресурсоемкость относительно низкая
 - Точность и/или полнота – снижаются
 - Реализуется большинством анализаторов

Масштаб анализа

- Межпроцедурный анализ
 - Все функция анализируется совместно
 - Информация передается между функциями
 - Ресурсоемкость очень высокая
 - Точность и/или полнота – выше
 - Реализовано в небольшом числе средств

Консервативность анализа

- Определяется правилами объединения ветвей
- Консервативный (пессимистический) анализ
 - При объединении ветвей объединяет интервалы
 - Сохраняет полноту (обнаруживаем все ошибки)
 - Теряет точность (много ложных обнаружений)
- Оптимистический анализ
 - При объединении ветвей пересекает интервалы
 - Теряет полноту (может пропускать ошибки)
 - Сохраняет точность (мало ложных обнаружений)

Консервативность анализа

```
int arr1[10];
int arr2[5];
if (...) {
...
} // <i, [1..8]>
else {
...
} // <i, [6..15]>
a = arr1[i]; // возможно ошибка
b = arr2[i]; // ошибка
```

- Консервативный анализ:
 - <i, [1..15]>
 - обнаружит обе ошибки (возможно первая – ложная)
- Оптимистический анализ
 - <i, [6..8]>
 - обнаружит вторую ошибку (пропустив первую)

Общая схема обнаружения дефектов



Обнаружение дефектов

- На основе анализа допустимости состояний в элементах модели
- Использование неинициализированной переменной:
 - Конструкция: $a = b$
 - Условие: $\langle a, \text{noninit} \rangle \in S^{l-1}$
- Повторное освобождение объекта
 - Конструкция: $\text{free}(p)$
 - Условие: $\langle p, \text{invalid} \rangle \in S^{l-1}$

Обнаружение дефектов

- Разыменование некорректного указателя
 - Конструкция: $a = *p$
 - Условие: $\langle p, \text{invalid} \rangle \in S^{l-1}$
- Арифметические операции над указателями на разные объекты
 - Конструкция: $a = p - q$;
 - Условие: $\langle p, o_1 \rangle, \langle q, o_2 \rangle \in S^{l-1}, o_1 \neq o_2$
- Выход за границы объекта
 - Конструкция: $a = *p$;
 - Условие: $\langle p, o, k \rangle \in S^{l-1}, k \geq \neq 0^{\text{size}}$

Аннотирование

- Что делать если:
 - часть исходных кодов отсутствует?
 - проект использует библиотечные вызовы?
 - вы знаете о коде нечто такое, что не способен выявить анализатор?
- Ответ: использовать механизмы внесения внешней семантики в процесс анализа - аннотирование

Аннотирование

- Способы аннотирования:
 - расширение целевого языка с помощью аннотирующих комментариев
 - расширение целевого языка с использованием его внутренних средств
 - внешние средства для описания поведения компонентов

Языки аннотаций

- На основе комментариев в C
 - ACSL (ANSI/ISO C Specification Language)
 - Аннотации в SPLint
- На основе внутренних средств C
 - CLANG
 - SAL (Source Annotation Language)
 - PREfast Annotations
- Оригинальный язык
 - Metal
 - *PanLang*

Языки аннотаций: цели

- ❑ Описание поведения функций
- ❑ Описание глобальных объектов
- ❑ Описание ресурсов
- ❑ Описание предусловий, инвариантов, ...
- ❑ Обнаружение новых типов дефектов
- ❑ Маскирование ложных дефектов
- ❑ Маскирование участков кода
- ❑ ...

Metal: описание ошибок работы с ресурсами

```
state decl any_pointer v;  
start: { kfree(v) } ==> v.freed;  
v.freed: { *v } ==> v.stop,  
{ err("Использование %s после освобождения!",  
      mc_identifier(v)); }  
| { kfree(v) } ==> v.stop,  
{ err("Двойное освобождение %s!",  
      mc_identifier(v)); };
```

ACSL: пример предусловия

```
/*@  
  requires 0 <= n;  
  requires \valid_range(a, 0, n-1);  
*/  
void example(value_type* a, size_type n);
```

RanLang: пример аннотации функции *realloc*

```
void* realloc(void* ptr, unsigned size) {
    if (size <= 0) {
        delete(Heap) ptr;
        return 0;
    }
    else {
        void* res = new Heap(Allocated, size, none, true);
        if (ptr->size <= size) {
            size = ptr->size;
        }
        copy(res, ptr, size);
        delete(Heap) ptr;
        return res;
    }
}
```

RanLang: пример аннотации функции *recv*

```
long recv ( int s ,void* buf ,unsigned len ,int flags) {
    if ( buf == 0 ) {
        defect (INI-03, "Второй аргумент функции - NULL");
    }
    if ( flags < 0 ) {
        defect (INI-01, "Некорректное значение аргумента
        функции");
    }
    if ( state(s) != Established) {
        defect (RES-06, "Получение данных из несоединенного
        сокета");
    }
    set ( buf, [-inf:+inf], len );
    return [0:len];
}
```


Проблемы статического анализа

- Анализ циклов
- Анализ рекурсии и глубокой вложенности вызовов
- Анализ больших типов данных
- Анализ программ с исключениями
- Анализ программ с полиморфизмом и указателями на функции
- Параллельные программы

Средства статического анализа

- Зарубежные анализаторы
 - Coverity Prevent SA
 - Klockwork Insight
 - Fortify SCA
 - Mathworks PolySpace
 - IBM RSA
 - Microsoft SCA
 - Microsoft PREFIX/PreFast
 - ParaSoft C++Test
 - Frama-C
 - SPLint
- Российские анализаторы
 - SVaCE Detector (ИСПРАН)
 - Viva64, VivaMP (СиПроВер)
 - AEGIS (СПбГПУ)

Статический анализатор Aegis

- Разработан анализатор Aegis:
 - Поддержка стандарта C (ISO/IEC 9899:2001)
 - Поддержка расширений GNU
 - Анализ многофайловых проектов
 - Обнаружение программных дефектов следующих типов:
 - выход за границу массива
 - переполнение буфера
 - использование неинициализированных переменных
 - работа с некорректными указателями
 - ошибки форматной строки
 - утечки ресурсов
 - некорректная работа с ресурсами
 - Поддержка языка аннотаций PanLang

Aegis: плагин к IDE Eclipse

The screenshot displays the Eclipse IDE interface with the Aegis plugin. The main editor shows the source code of `rmdir.c` from the `frebsd` project. The code includes a `rm_path` function that processes a path string.

The **Problems** view at the bottom shows a message: "Операция выполнена успешно (получение и разархивация проекта)".

The **Defects** view displays a table of detected issues:


| Тип | Стандартное описание | Описание | Файл | Стрс | Полож |
|---------------|--|---|---------|------|-------|
| (summary) | | | | | |
| BUF-05* [DEF] | Дефект выполнения арифметических операций или сравн | Операции с указателями char * p и char * path | rmdir.c | 100 | 8 |
| BUF-02* [DI] | Дефект разыменования указателя, выведенного за преде | Разыменование указателя char ** argv, выведё | rmdir.c | 79 | 19 |
| BUF-02* [DEF] | Дефект разыменования указателя, выведенного за преде | Разыменование указателя char * p, выведенног | rmdir.c | 102 | 2 |
| BUF-02* [DEF] | Дефект разыменования указателя, выведенного за преде | Разыменование указателя char * p, выведенног | rmdir.c | 100 | 8 |

Below the table, the **Вывод скриптов и программ (в скобках - размеры)** section shows the following files and their sizes:

- stderr (50178)
- stdout (809)
- cfg.dump (30365)
- defects.xml (2611)
- projectparser.log (18400)

Аегіs: свободный сервис

<http://digiteklabs.ru/aegis>



Главная | **Аегіs** | Партнеры | Контакты

Главная > Аегіs > Demo

Результаты анализа

| Путь | Код | Дефекты, контексты |
|------|---|---|
| | 01: // Удобный пример с несколькими дефектами | |
| | 02: | |
| | 03: void f(void) { | |
| | 04: int a1[10]; | |
| | 05: int a2[15]; | |
| | 06: int *pa1 = a1; | |
| | 07: int *pa2=a2+7; | |
| 1(4) | 08: int diff = (pa2 - pa1); //BUF-05 | <input checked="" type="radio"/> Операции с указателями int pa2 и int pa1 на разные объекты (BUF-05) 1 ▾ |
| | 09: } | |
| | 10: | |
| | 11: void g(void) { | |
| | 12: int a1[10]; | |
| | 13: int a2[15]; | |
| | 14: int *pa1 = a1; | |
| | 15: int *pa2=a2+7; | |
| | 16: int diff = (pa2 - pa1); //BUF-05 | <input type="radio"/> Операции с указателями int pa2 и int pa1 на разные объекты (BUF-05) |
| | 17: } | |
| | 18: | |
| | 19: main2(int argc, char* argv[]) { | |
| 3) | 20: if (argc == 1) f(); else g(); | |

Статический анализ. Резюме

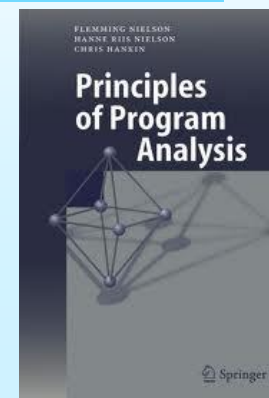
- Достоинства статического анализа
 - Возможность полного обнаружения дефектов определенного класса
 - Полная автоматизация процесса обнаружения
- Недостатки статического анализа
 - Наличие ложных обнаружений (false positives)
 - Высокая ресурсоемкость
 - Невозможность обнаружения функциональных ошибок

Место статического анализа

- Собственно статический анализ может использоваться
 - В составе сред разработки для оперативной проверки программ
 - В системах Continuous integration для постоянного контроля качества кода и своевременного обнаружения дефектов
- Статический анализ не заменяет тестирование, а дополняет его
- Хорошая практика - использование автоматизированного тестирования вместе со статическим анализом

Литература по СА

- M. Schwartzbach. Lecture Notes on Static Analysis. BRICS, Department of Computer Science University of Aarhus, Denmark
- D.Jackson, M.Rinard. Software Analysis: a Roadmap. 2000
- F. Nielson, H. Nielson, C. Hankin. Principles of Program Analysis
- Глухих М.И., Ицыксон В.М. Программная инженерия. Обеспечение качества программных средств методами статического анализа. СПб: СПбГПУ. 2011
- <http://digiteklabs.ru/research/defectdetection/>



Контактная информация

Санкт-Петербургский государственный политехнический университет

Факультет технической кибернетики

Кафедра компьютерных систем и программных технологий

Лаборатория программно-аппаратных разработок. Web: <http://digiteklabs.ru>



Ицыксон Владимир Михайлович, к.т.н, доц.

E-mail: vlad@ftk.spbstu.ru

Тел.: +7(812)297-22-38



Спасибо за внимание!