НАДЁЖНОСТЬ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

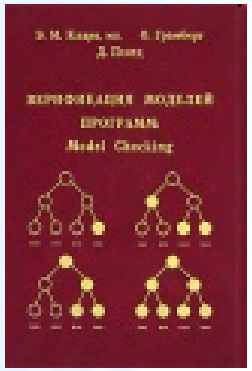
Лекция 7:
Статический анализ

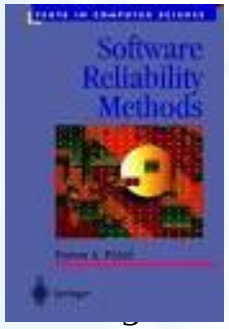
ВМиК МГУ им. М.В. Ломоносова,
Кафедра АСВК, Лаборатория Вычислительных Комплексов
Ассистент Волканов Д.Ю.

План лекции

- Статический анализ
  - Модели программ
  - Алгоритмы статического анализа
  - Обнаружение дефектов
- Другие статические методы
- Заключение

Литература





- Карпов Ю.Г. Model Checking. Верификация параллельных и распределенных программных систем, БХВ-Петербург, 2010.
- Кларк, Грумберг, Пелед. Верификация моделей программ: Model checking, МЦНМО, 2002.
- Peled. Software Reliability Methods, Springer, 2001.

О чём этот курс

- Концепции и связи
- Аналитические модели и средства поддержки
- Методы для увеличения надёжности ПО:
  - Верификация
  - **Статический анализ**
  - Тестирование
  - Расчёт надёжности
  - *Анализ статистики ошибок*
  - *Безопасность*

Статические методы обеспечения качества

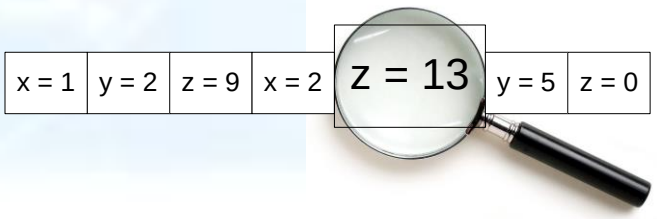
- Формальные инспекции, аудит
- Формальная верификация
  - Дедуктивная верификация
  - Model checking
- **Статический анализ**
- Трансформации
  - Рефакторинги
  - Модификации

Статический анализ

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

Статический анализ для обнаружения дефектов

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



Нефункциональные дефекты

- Ошибки этапа кодирования
- Не являются явным нарушением спецификации
- Являются следствием
  - сложности языков программирования
  - сложности библиотек
  - сложности алгоритмов и структур данных

Нефункциональные дефекты

- Выход за границу массива
- Переполнение буфера
- Использование неинициализированных объектов
- Работа с некорректным указателем
- Арифметические ошибки
- Утечки ресурсов
- Ошибки форматных строк
- Ошибки работы с ресурсами
- и т.д.

Проявление дефектов

- Без проявления
- Зависания
- Сбои
- Падения
- Деградация производительности
- …

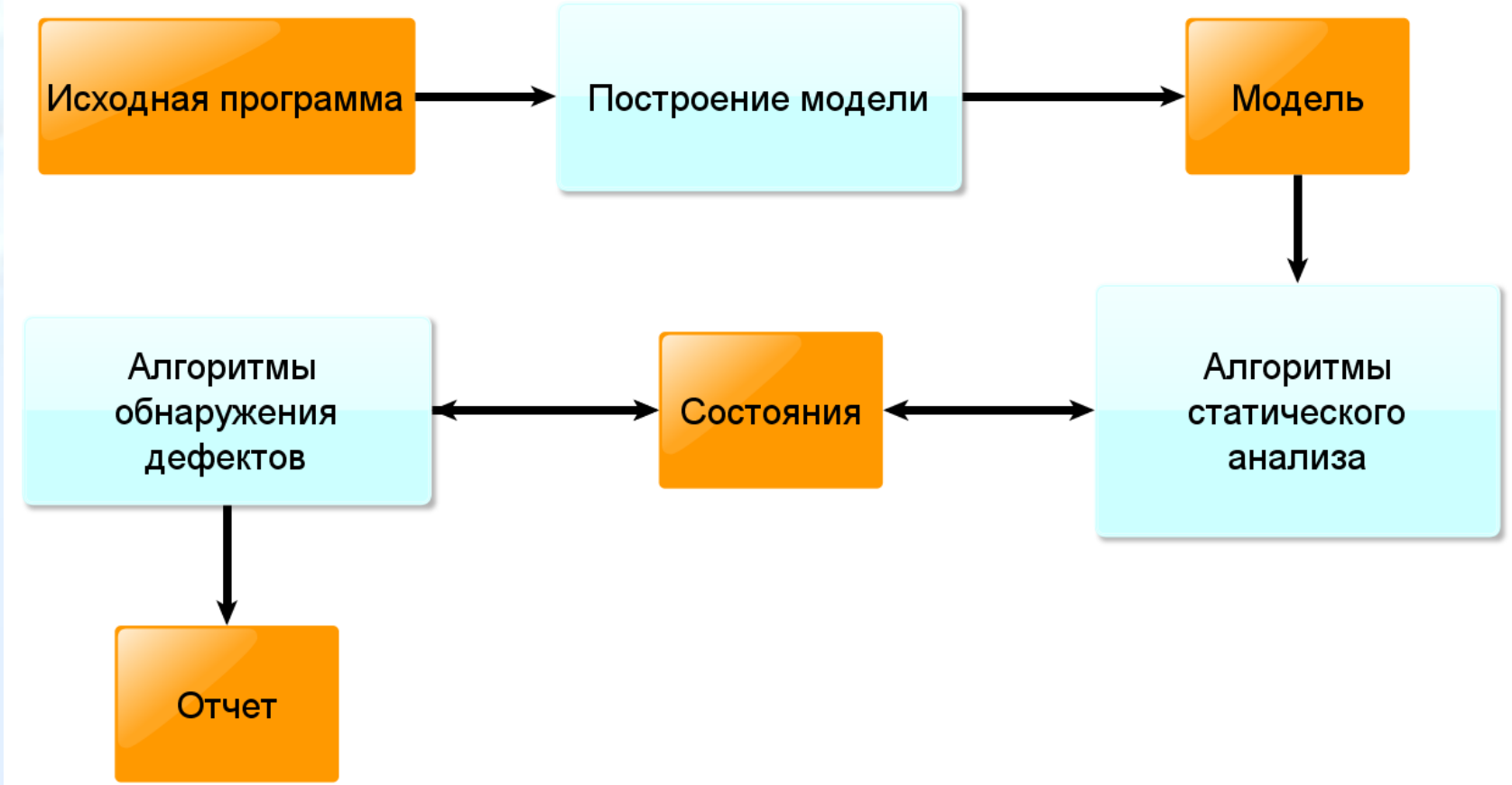
Опасные конструкции

- Язык C
  - Макроопределения
  - Указатели
  - Массивы
  - Адресная арифметика
  - Указатели на функции
  - Приведения типов
  - Объединения (union)

Характеристики анализа

- Полнота – доля найденных дефектов по отношению ко всем дефектам программы
- Точность – доля найденных истинных дефектов по отношению ко всем найденным
- Ресурсоемкость

Общая схема обнаружения дефектов



Модели программ. Требования

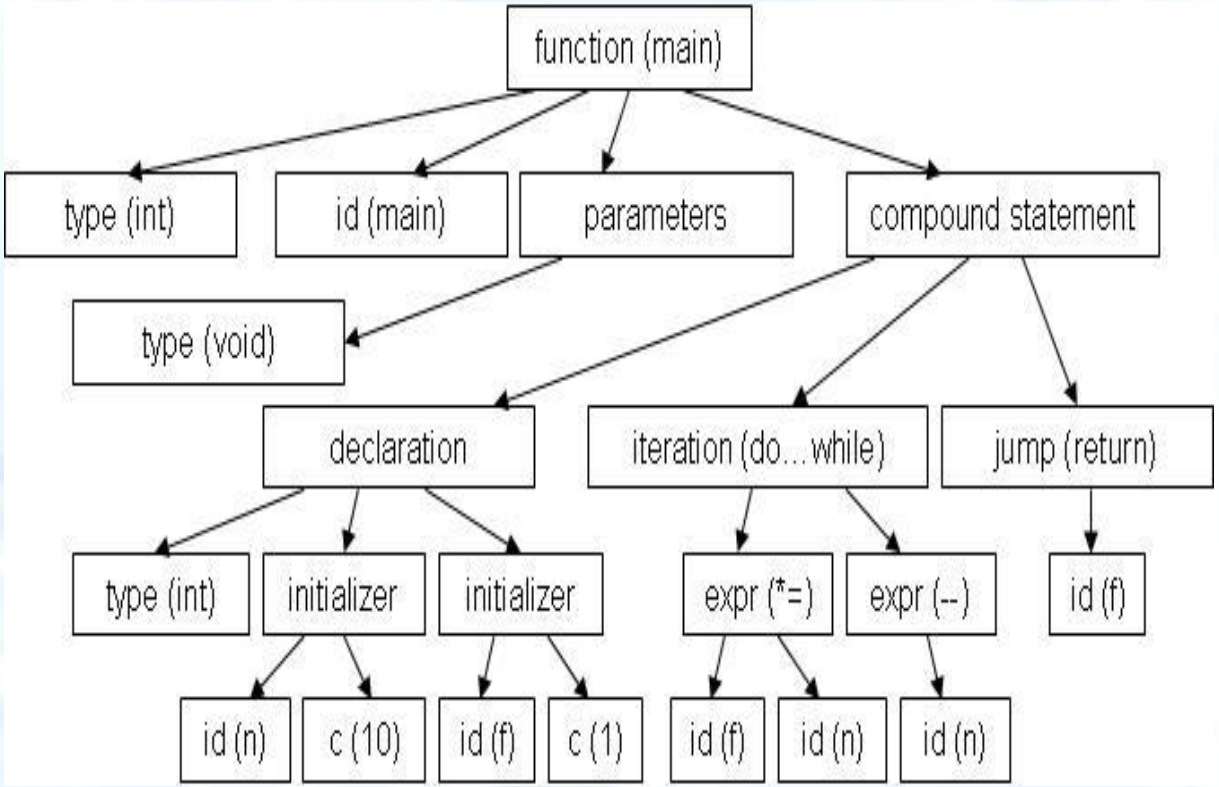
- Полнота представления объектов программы
- Восстанавливаемость
  - Идеально: возможность полного восстановления исходного кода
  - Желательно: связь с исходным кодом
- Эффективность поиска объектов

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)
  - SSA - static single assignment
- Гибридные
  - Абстрактный семантический граф (ASG - abstract semantic graph)

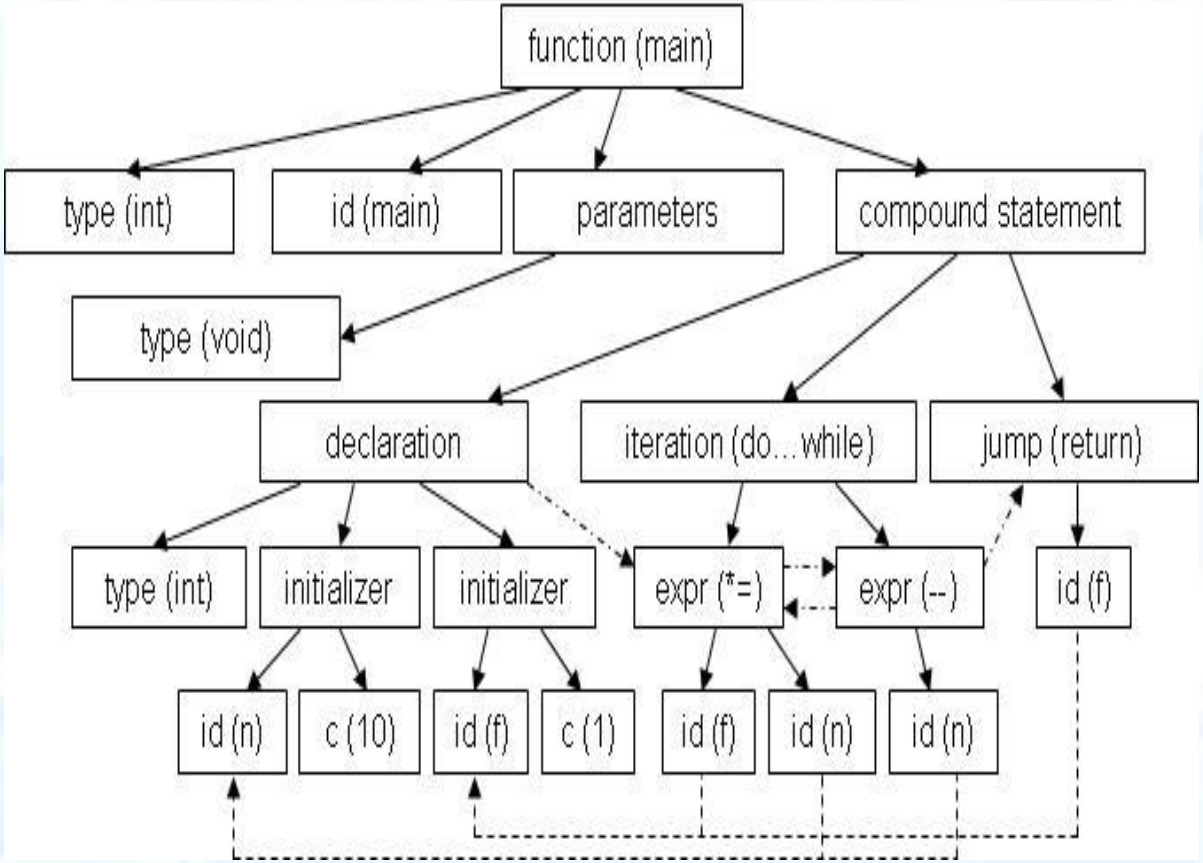
Модели программ: AST

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



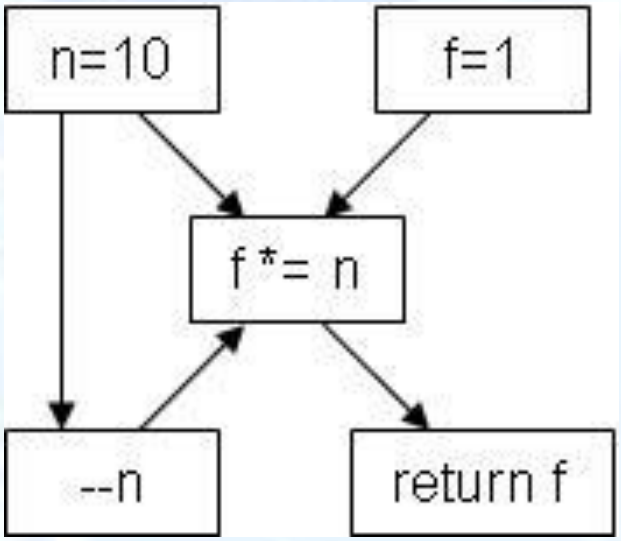
Модели программ: ASG

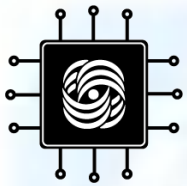
```
int main()
{
   int n = 10;
   int f = 1;
   while (n >
    0)
   {
      f *= n;
      --n;
   }
   return f;
}
```



Модели программ: DDG

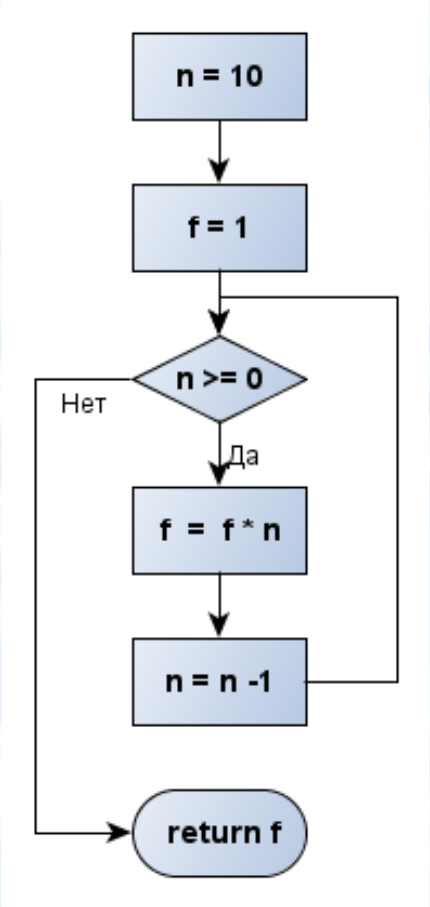
```
int main()
{
   int n = 10;
   int f = 1;
   while (n >
    0)
   {
      f *= n;
      --n;
   }
   return f;
}
```





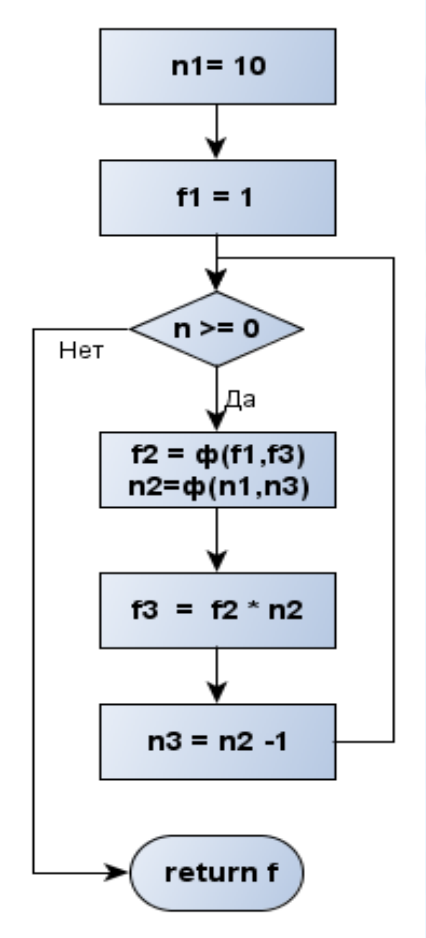
Модели программ: CFG

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



Модели программ: SSA

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



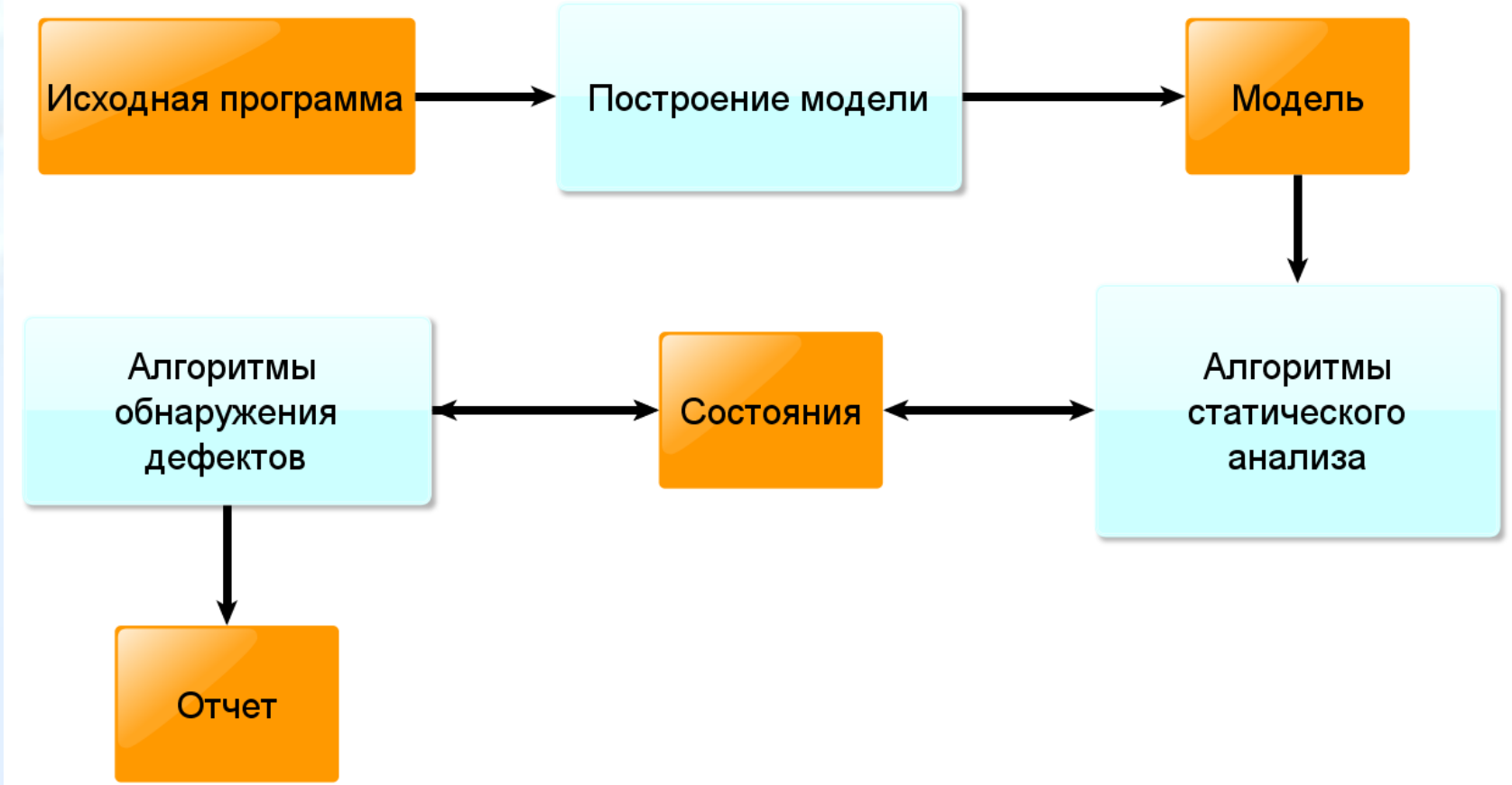
Модели программ. Резюме

- Для простейшего статического анализа достаточно структурных моделей (AST)
- Для обнаружения дефектов необходим учет семантики:
  - ASG
  - CFG
  - CFG в форме SSA

Модели программ. Построение

- Построение парсера вручную
- Использование генераторов парсеров (ANTLR, YACC, etc)
- Использование самостоятельных 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)

Общая схема обнаружения дефектов



Анализ на основе состояний программы

- Состояние программы – совокупность состояний всех объектов программы в какой-либо точке
- Цель – вычислить все возможные состояния программы во всех точках
- Обнаружение дефектов происходит на основе проверки вычисленных состояний

Алгоритмы статического анализа

- По типу алгоритму
  - Интервальный анализ
  - Анализ указателей
  - Ресурсный анализ
- По масштабу
  - Внутрипроцедурный
  - Межпроцедурный
- По консервативности
  - Пессимистический (консервативный) анализ
  - Оптимистический анализ

Интервальный анализ

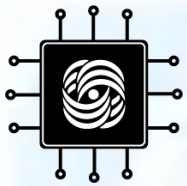
- Состояние переменных представляются интервальными значениями
  - a = [10;20]
  - b = (-40;20) U (30;100)
  - c = [0;∞)
- Вычисление состояния после выполнения оператора языка производится на основе интервальной арифметики
- Каждому оператору ставится в соответствие уравнение, связывающее состояние до выполнения и после

Интервальный анализ

Элемент модели	**Интервальное правило**
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;	$S^l = S^{l-1} \setminus U_i \langle a, i \rangle \cup U_j \langle a, b_j \rangle$
φ(…)	$S^l = U_i S^i$
if(cond) S^{true} else S^{false}	$S^{true} = S^{l-1} \cap cond$
	$S^{false} = S^{l-1} \cap \neg cond$

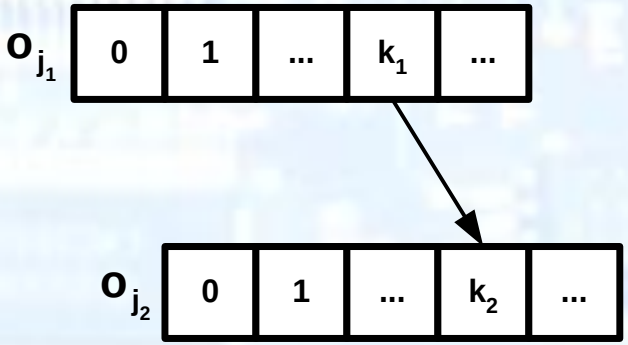
Интервальный анализ. Пример

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



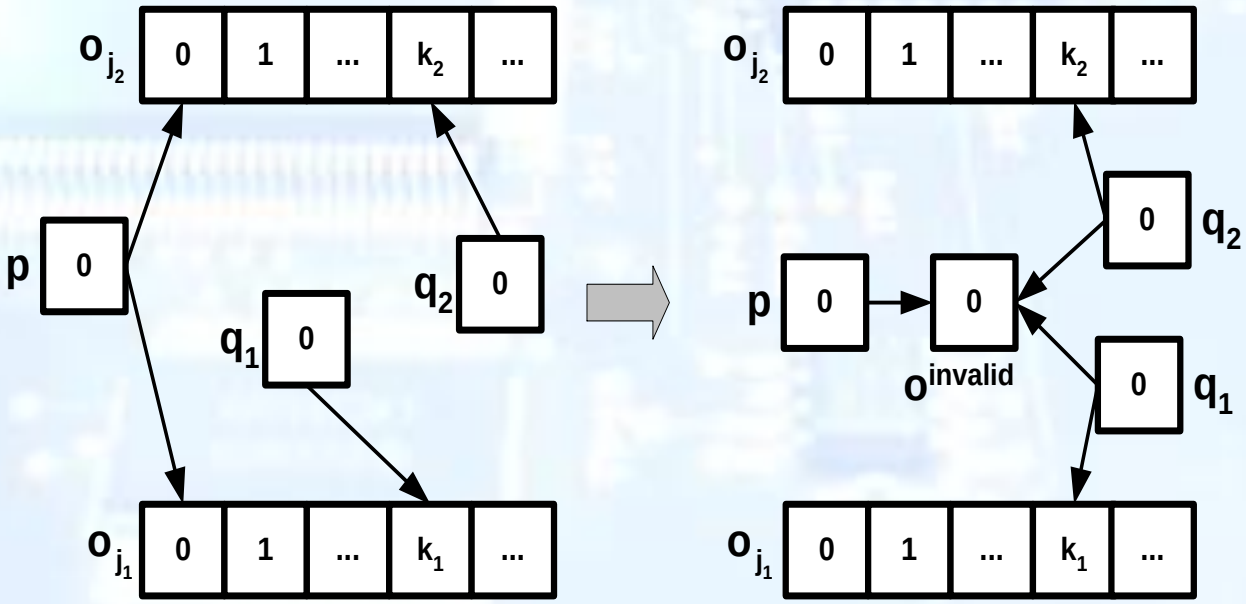
Анализ указателей

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



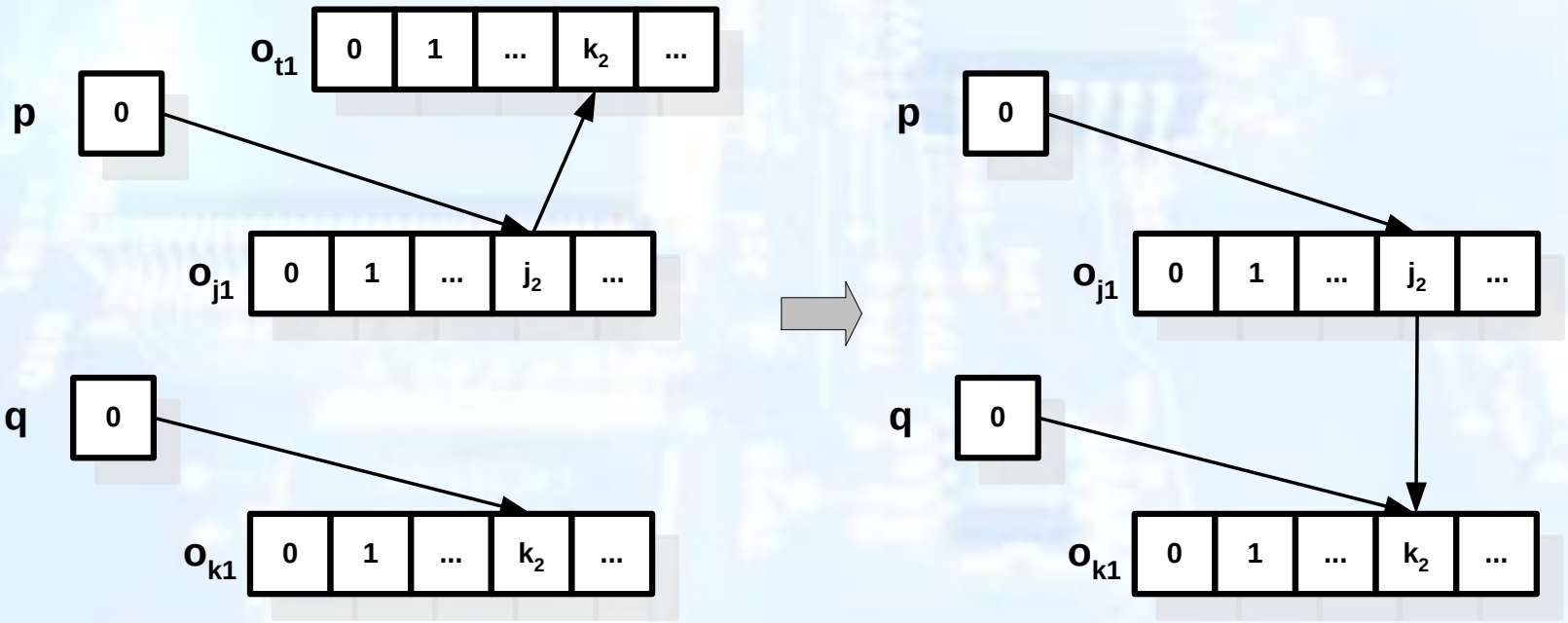
Анализ указателей. Пример

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



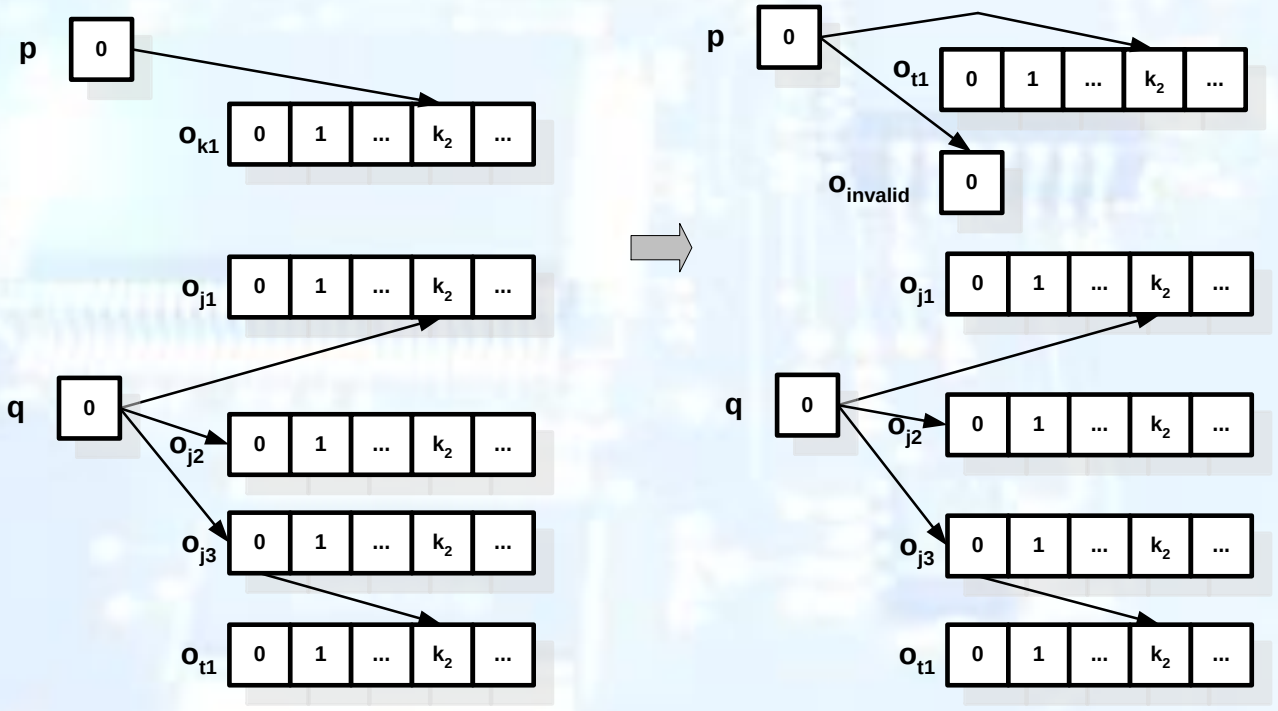
Анализ указателей. Пример

- 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 = malloc(...))
  - Используется (*p)
  - Освобождается (free(p))
- Примеры дефектов
  - Использование не созданного объекта
  - Повторное освобождение
  - Утечка

Ресурсный анализ. Файл

- Правила использования
  - Открыть (f = fopen(“…”, “r”)
  - Читать/писать (fprintf(f, …))
  - Закрыть (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

Масштаб анализа

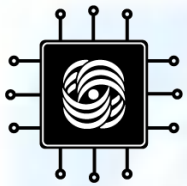
- Внутрипроцедурный анализ
  - Каждая функция анализируется отдельно
  - Информация не передается между функциями
  - Ресурсоемкость относительно низкая
  - Точность и/или полнота – снижаются
  - Реализуется большинством анализаторов

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)

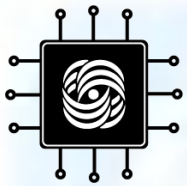
Решение уравнений

- В результате применения алгоритмов в явном или неявном виде строится система уравнений
- Решение системы
  - С использованием теории решеток
  - Методами абстрактной интерпретации
- Завершаемость процесса решения обуславливается
  - Монотонностью уравнений
  - Конечностью системы уравнений

Общая схема обнаружения дефектов

Обнаружение дефектов

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



Обнаружение дефектов

- Разыменование некорректного указателя
  - Конструкция: a = *p
  - Условие: $\langle p, invalid \rangle \in S^{l-1}$
- 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: $\langle p, o, k \rangle \in S^{l-1}$, $k \geq \neq o^{size}$

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

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

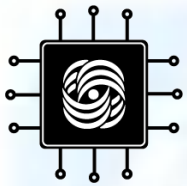
- Собственно статический анализ может использоваться
  - В составе сред разработки для оперативной проверки программ
  - В системах Continuous integration для постоянного контроля качества кода и своевременного обнаружения дефектов
- Статический анализ не заменяет тестирование, а дополняет его
- Хорошая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

Другие статические методы

- Инспекции
  - Формальные инспекции
  - Аудит кода
  - Парное программирование
- Дедуктивная верификация
- Model Checking

Применение различных методов обеспечения качества

	Хар-ка качества	Проблема	Обеспечение качества
1	Функциональность	Функциональные ошибки, несоответствие спецификации	Тестирование Верификация
2	Надежность	Низкая надежность Наличие уязвимостей	Статический анализ Тестирование
3	Практичность	Сложность использования	Тестирование
4	Эффективность	Проблемы с производительностью, ресурсами	Тестирование Профилирование Динамический/статический анализ
5	Сопровождаемость	Сложность сопровождения, модификации	Рефакторинг Документирование
6	Мобильность	Несоответствие стандартам, Сложность адаптации	Аудит, рефакторинг Статический анализ



Спасибо за внимание!