ПОЛИТЕХ
Санкт-Петербургский
политехнический университет
Петра Великого

ИКНТ

Методы обеспечения качества программных систем

2016

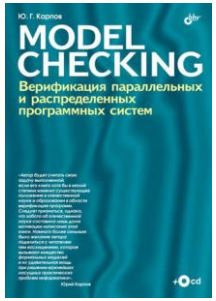
КСПТ
КОМПЬЮТЕРНЫЕ СИСТЕМЫ И ПРОГРАММНЫЕ ТЕХНОЛОГИИ

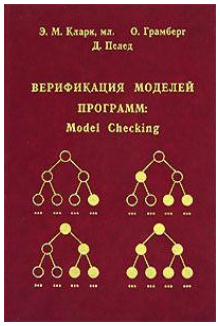
План курса

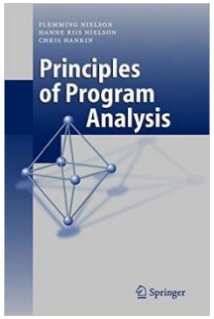
- Введение
- Качество ПО
- Модели программ
  - Виды моделей программ
  - Построение моделей программ
- Model Checking
  - Темпоральная логика
  - Верификация моделей программ
- Статический анализ
- Дедуктивная верификация
  - Метод Флойда
  - Метод Хоара
- Bounded Model Checking

Литература

- **Ю.Г. Карпов.** Model Checking. Верификация параллельных и распределенных программных систем. СПб:BHV, 2010, с. 552.
- **Д. Пелед, О. Грамберг, Э.М. Кларк.** Верификация моделей программ. Model Checking. МЦНМО, 2002, 416 с.
- **М.И. Глухих, В.М. Ицыксон.** Программная инженерия. Обеспечение качества программных средств методами статического анализа: уч. пособие. СПб: СПбГПУ, 2011, 150 с.
- **Н. И. Поликарпова, А.А. Шалыто.** Автоматное программирование. 2008. — 167 с.: ил.
- **F. Nielson, H. Nielson, C. Hankin.** Principles of Program Analysis. Springer: 1999, pp 452.







Качество программного обеспечения

- Качество ПО (ГОСТ Р ИСО/МЭК 9126, ISO 9126)
  - Весь объем признаков и характеристик программной продукции, который относится к её способности удовлетворять установленным и предполагаемым свойствам
- Характеристик качества ПО
  - Функциональность (Functionality)
  - Надежность (Reliability)
  - Практичность (Usability)
  - Эффективность (Efficiences)
  - Сопровождаемость (Maintainability)
  - Мобильность (Portability)

Качество ПО. Функциональность

- ***Функциональность*** - набор атрибутов характеризующий, соответствие функциональных возможностей ПО набору требуемой пользователем функциональности.
- Подхарактеристики:
  - Пригодность (соответствие требуемому набору функций)
  - Корректность (правильность, точность)
  - Способность к взаимодействию (с другими компонентами и системами)
  - Согласованность (соответствие стандартам)
  - Защищенность

Качество ПО. Надежность

- ***Надежность*** - набор атрибутов, относящихся к способности ПО сохранять свой уровень качества функционирования в установленных условиях за определенный период времени.
- Подхарактеристики:
  - Стабильность (число отказов при ошибках)
  - Устойчивость к ошибкам
  - Восстанавливаемость
  - Доступность/Готовность

Модели программ

- Модели программ:
  - Структурные модели
  - Поведенческие модели
- Структурные модели основаны на
  - Синтаксисе исходного языка
  - Структуре программы
- Поведенческие модели
  - Дополнительно используют семантику конструкций языков программирования

Модели программ

- Структурные модели
  - Дерево разбора (синтаксическое дерево, parsing tree)
  - Абстрактное синтаксическое дерево (abstract syntax tree, AST)
- Поведенческие модели
  - Граф потока управления (Control Flow Graph, CFG)
  - Статическое однократное присваивание (Single Static Assignment, SSA)
  - Граф зависимостей по данным (Data Dependency Graph, DDG)
  - Граф зависимостей программы (Program Dependency Graph, PDG)
- Гибридные модели
  - Абстрактный семантический граф (Abstract Semantic Graph, ASG)

Дерево разбора

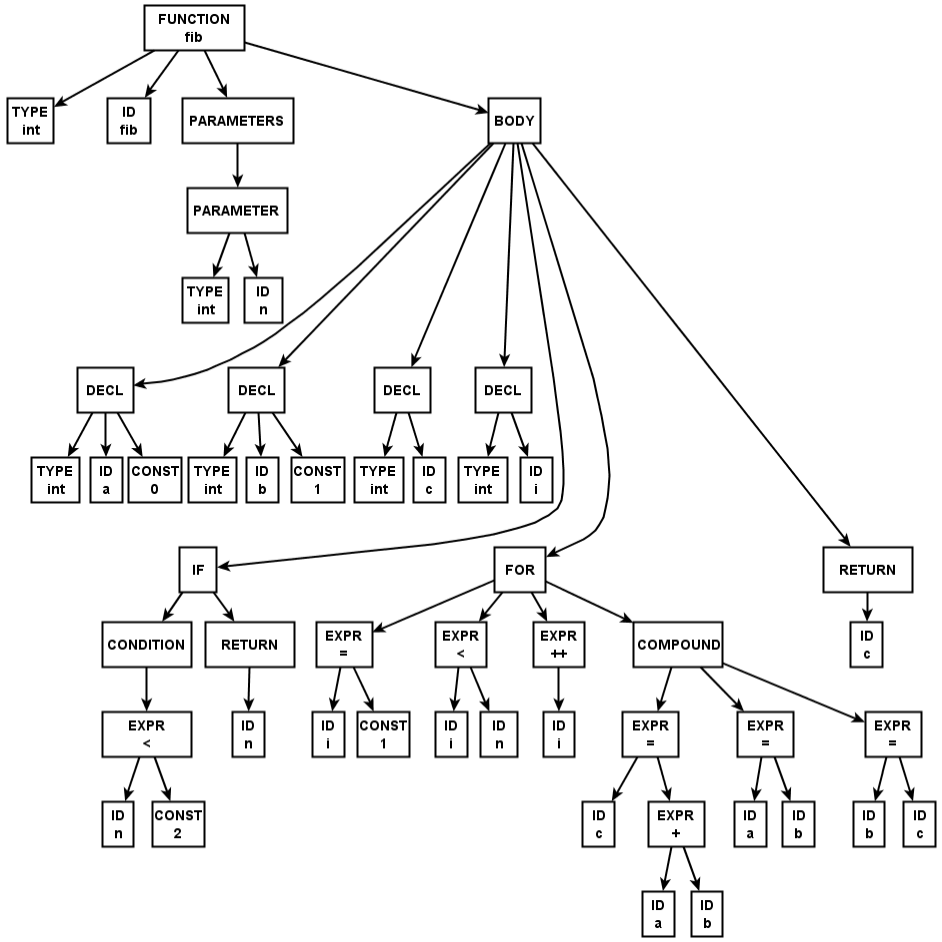
- Синтаксическое дерево, конкретное синтаксическое дерево, parsing tree
- Является формой представления программы, соответствующей грамматике исходного языка (например, БНФ или РБНФ)
- Строится парсерами написанными вручную или синтезированными генераторами парсеров
- Является наиболее детальным структурным представлением программы
- В исходном виде редко используется для анализа программ из-за громоздкости

Абстрактное синтаксическое дерево

- Структурное представление программы
- Отличие от синтаксического дерева (дерева разбора):
  - Удаление большинства нетерминальных узлов, имеющих одного потомка
  - Замена части терминальных символов атрибутами узлов
  - Модификация группирующих узлов
- По модели АСД легко восстановить исходную программу с точностью до форматирования и комментариев
- Строится из дерева разбора путем несложной постобработки

Абстрактное синтаксическое дерево

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



Граф потока управления

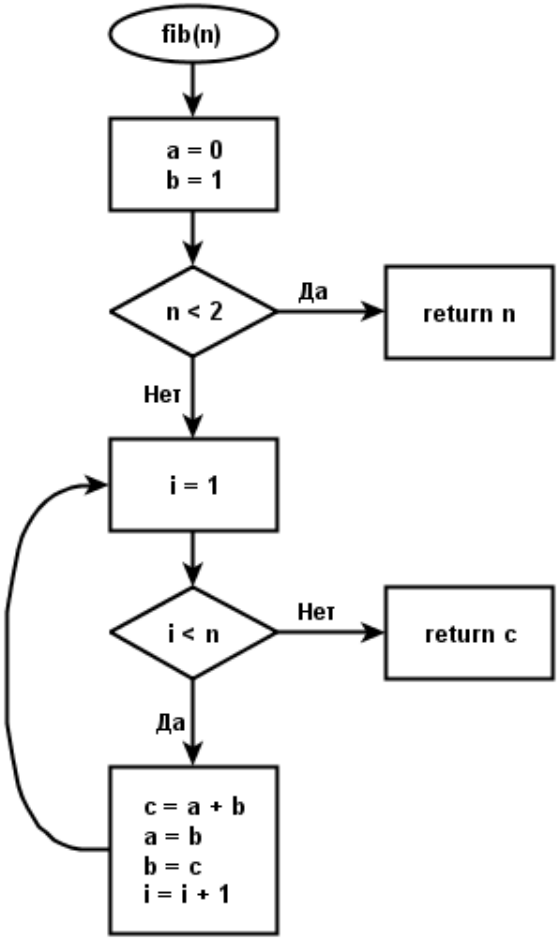
- Граф потока управления (ГПУ), Control Flow Graph (CFG)
- Потоки управления программы представляются в виде направленного графа
- Определяется не только структурой программы, но и семантикой операторов
- Два узла графа (A и B)связаны направленной дугой, если возможно выполнение B непосредственно после A
- ГПУ используется для:
  - Генерации кода
  - Оптимизации программ
  - Обнаружения ошибок в программе
  - И т.п.

Построение графа потока управления

- ГПУ строится с помощью анализа АСД на основе семантики языковых конструкций
  - Для большинства конструкций – дуга, соединяющая узлы, соответствующие следующим друг за другом операторам
  - Для части конструкций (ветвления, циклы, вызов функций, переходы и т.п.) – в соответствии с семантикой
- Проблемные конструкции языков программирования:
  - Вычисляемые безусловные переходы (goto)
  - Вызов функции по указателю
  - Полиморфизм
  - Структурная обработка исключений
  - Параллелизм
  - И т.п.

Граф потока управления

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



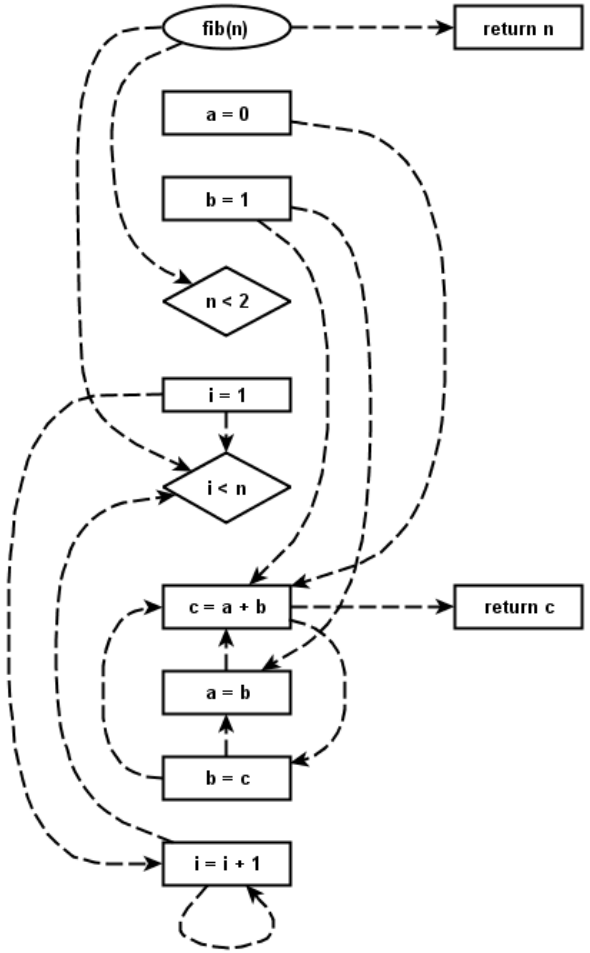
Ицыксон В.М. МОК © 2016

Граф зависимостей по данным

- Граф зависимости по данным (ГЗД), Data Dependency Graph (DDG)
- Отображает связи по данными между объектами программы
- Показывает зависимости между объявлениями, инициализациями, присваиваниями объектов и их использованиями
- Определяется структурой программы, семантикой конструкций
- Используется для:
  - Оптимизации программ
  - Автоматического распараллеливания программ
  - Обнаружения неинициализированных объектов
  - Обнаружения некорректно инициализированных объектов
  - и т.п.

Граф зависимостей по данным

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

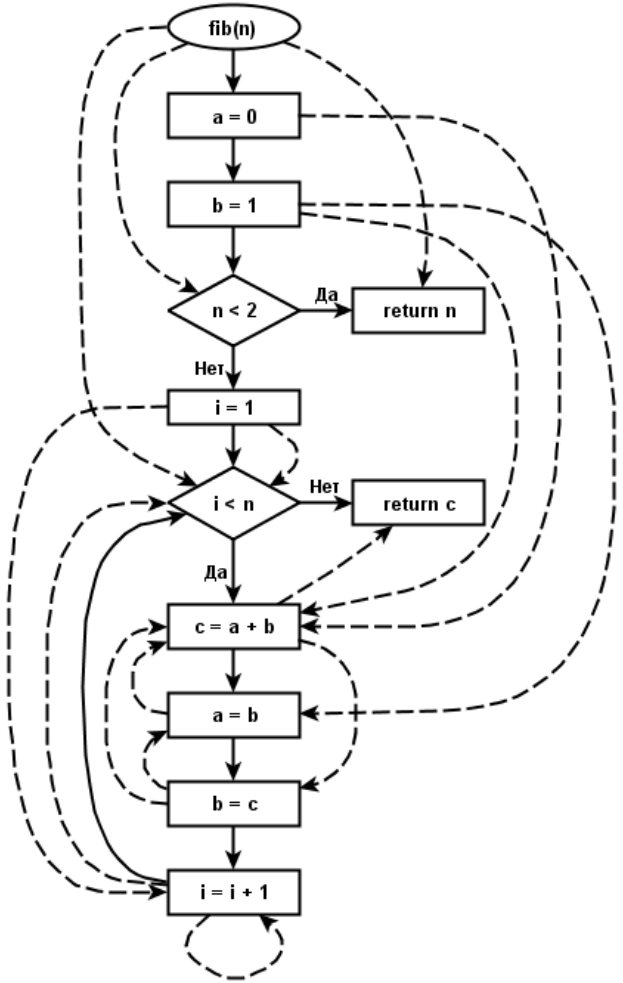


Граф зависимостей программы

- Является комбинированной моделью
- Содержит два типа связей:
  - Связи потока управления
  - Зависимости по данным
- Используется для
  - Оптимизации
  - Генерации кода
  - Анализа программ
  - Верификации
  - И т.п.

Граф зависимостей программы

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

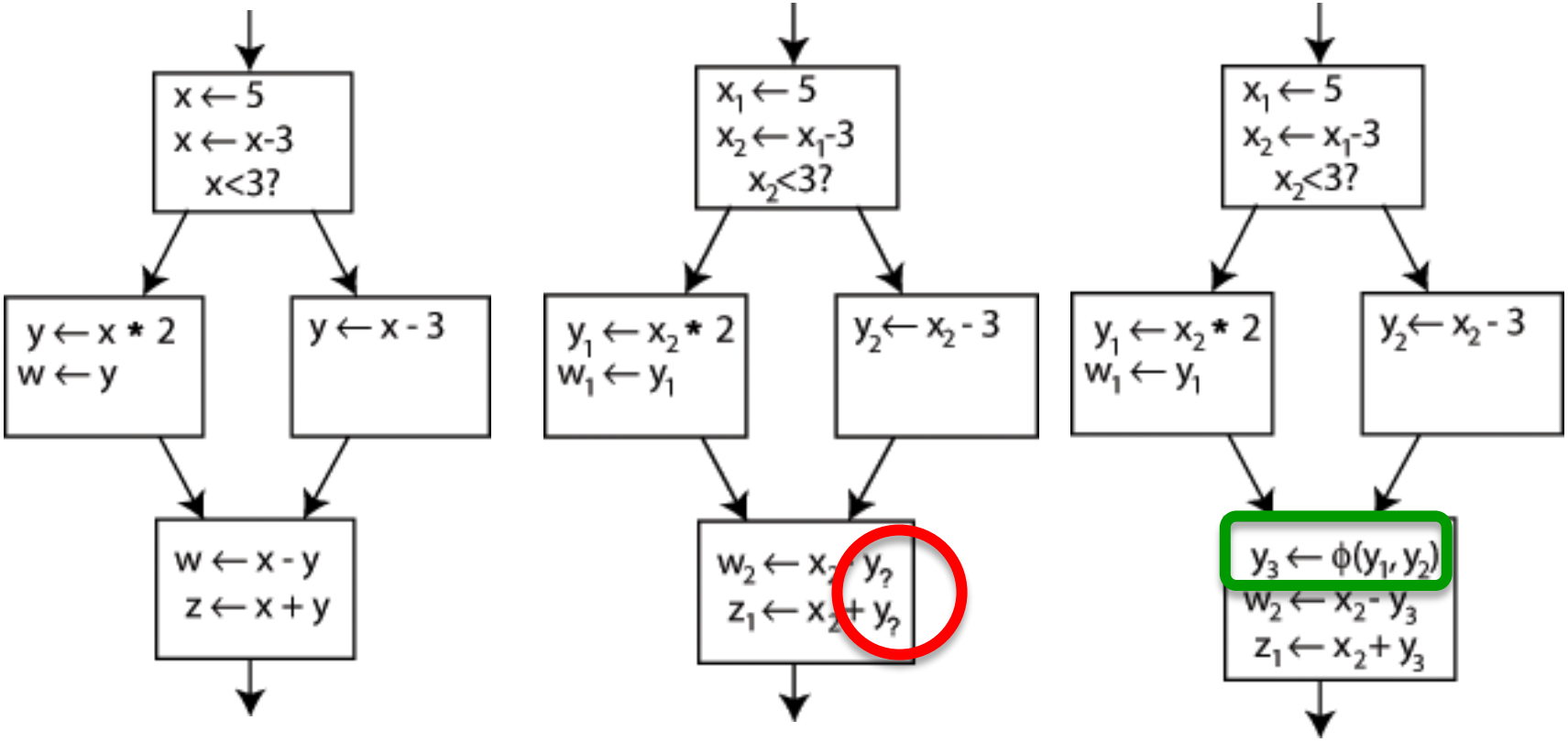


Ицыксон В.М. МОК © 2016

Статическое однократное присваивание

- Статическое однократное присваивание, Static Single Assignment form(SSA)
- Разработано IBM в 1981 году
- Модель, объединяющая граф потока управления и элементы зависимости по данным
- Значения всем объектам программы присваиваются один раз
- Вводится версионирование для каждого очередного присваивания объекта
- При обращении к значению объекта используется последняя версия по ходу графа потока управления
- Неоднозначность разрешается с помощью ϕ-функции

Статическое однократное присваивание

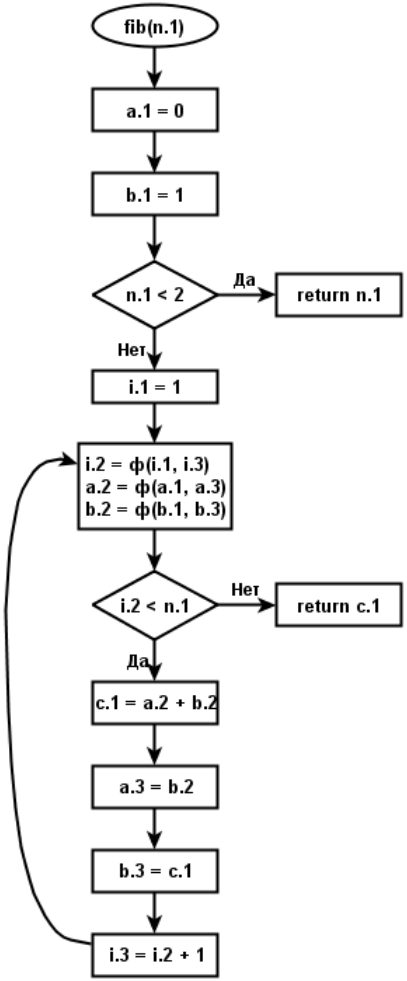


Статическое однократное присваивание

- Используется в компиляторах, статических анализаторах, верификаторах:
  - Распространение констант (constant propagation)
  - Обнаружение удаление мёртвого кода
  - Обнаружение неинициализированных объектов
  - Распараллеливание
  - Генерация кода
  - Преобразование программы / части программы в формулы для последующей верификации
  - И т.п.

Статическое однократное присваивание

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

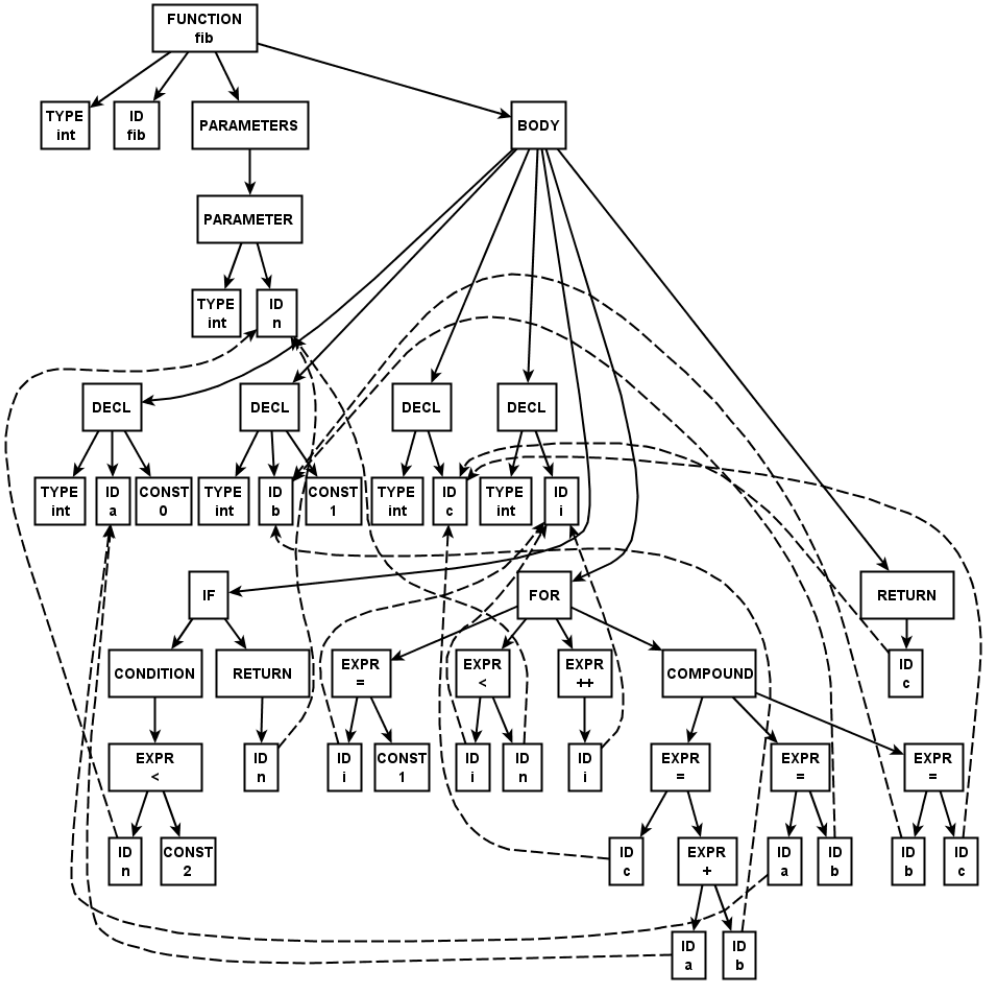


Абстрактный семантический граф

- Наиболее общая модель, объединяющая структурные и поведенческие компоненты
- Абстрактный синтаксический граф обогащается семантической информацией:
  - Связи по данным
  - Связи по управлению
  - Типами данных
  - И т.п.

Абстрактный семантический граф

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



Задачи обеспечения качества ПО

- Оценка качества программного обеспечения
- Проверка соответствия программы спецификации
- Обнаружение программных ошибок
- Проверка выполнимости свойств программы