Математические модели и методы логического синтеза СБИС

Осень 2020

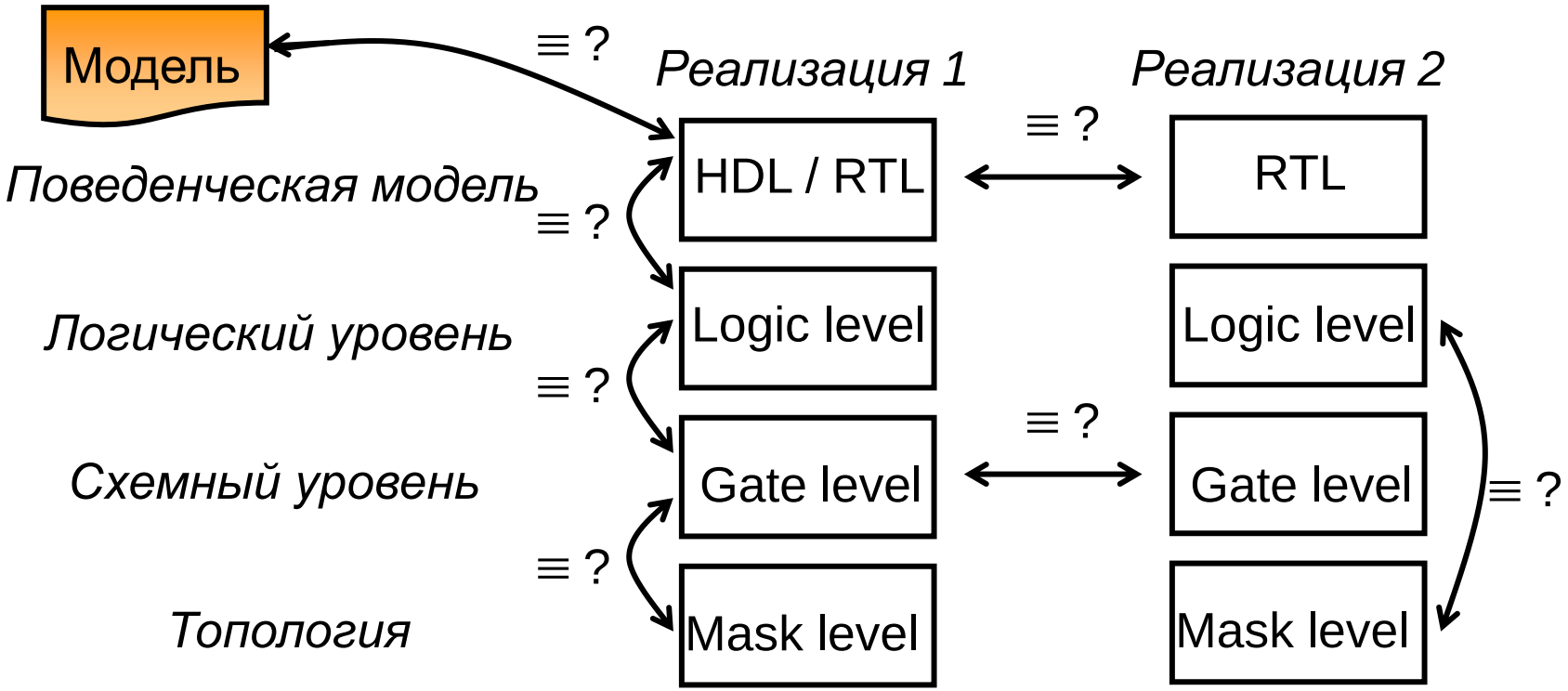
Лекция 11

План лекции

- Задачи верификации интегральных схем
- Методы решения задач верификации:
  - Формальные методы
  - Симуляция схем
  - Тестовый подход
- Задача проверки эквивалентности комбинационных схем

Верификация

- Верификация – проверка корректности
  - Модели и ее реализации (переход с одного уровня абстракции на другой)
  - Разных реализаций (на одном уровне абстракции)



Верификация - особенности

- Кризис верификации
  - Сложность устройств очень быстро растет
  - Верификация требует больше времени (> 70%), нежели само проектирование
  - Требуются методы автоматизации верификации и их интеграция в маршрут проектирования
- Последствия
  - Сбои в работе устройства, приводящие к катастрофам и угрожающие жизни людей
  - Неудобство (Ошибка Pentium FDIV)

Методы верификации

- Формальные методы
  - Дедуктивная верификация
  - Проверка моделей (model checking)
  - Проверка эквивалентности
- Симуляция – выполняется на модели
- Эмуляция, прототипирование - выполняется на макете устройства в рамках выбранного окружения
- Тестирование – выполняется на готовом устройстве (фабричное тестирование)

Функциональная верификация

- Задача: проверить работу устройства во всех возможных режимах работы
- Функциональная верификация RTL
  - Верифицировать схему RTL описания устройства
  - Зачастую нет модели с которой можно сравнить – функциональная симуляция
- Генерация функциональных тестов
  - Автоматическое построение тестов (проверка высоко-уровневых операций, синхронизации тактов и управляющих сигналов)
  - Методы на основе ВЫП

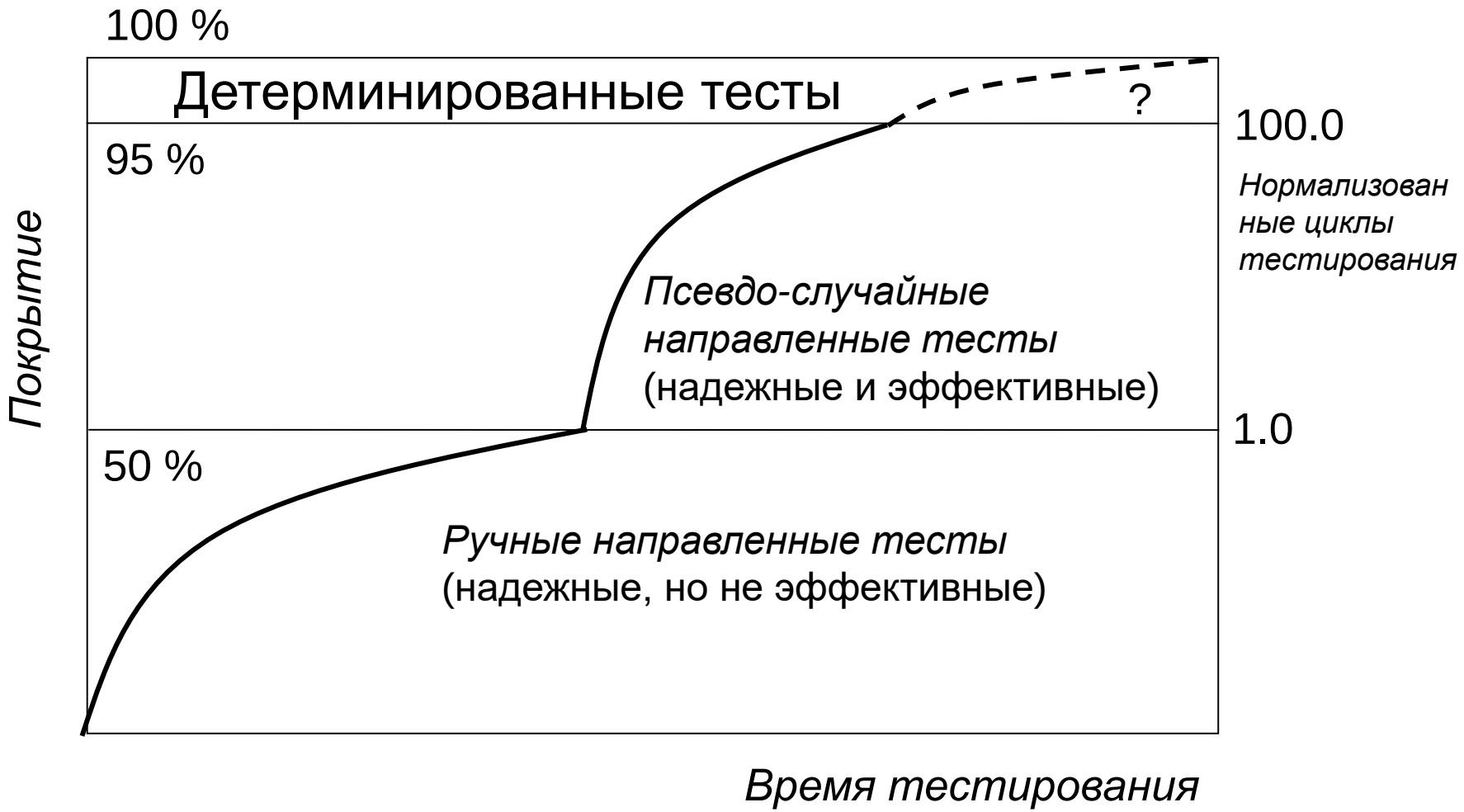
Оценка тестового покрытия

- Метрика тестового покрытия – методика измерения качества функциональной верификации
  - Мониторы: собирают данные о тестирование (покрытие, сценарии)
  - Низко-уровневые метрики (покрытие состояний, переходов, строк кода в аппаратной модели)
  - Высоко-уровневые метрики (покрытие событий, последовательностей состояний(транзакций), наборов данных
  - Само-тестирование (встроенное тестирование)

Построение функциональных тестов

- Задача: для заданной схемы и метрики покрытия достичь определенной цели покрытия
- Решение: функциональная симуляция
  - Направленные тесты
    - Зачастую пишутся вручную
    - Надежные (предсказуемое покрытие)
    - Неэффективные (покрывают только небольшую часть описания устройства)
  - Случайные тесты
    - Эффективные (быстрые), но ненадежные (непредсказуемое покрытие)
  - Детерминированные тесты
    - Генерируются автоматически
    - Учитывают ограничения (пользовательские, окружения, метрики покрытия
    - Возможны сложности в их вычислениях

Типичный сценарий функциональной верификации

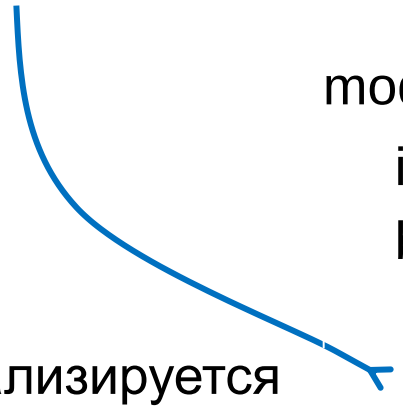


Методы построения тестов

- Случайные и псевдо-случайные методы
- Направленная псевдо-случайная симуляция
- Детерминированные методы
  - Методы, основанные на решении задачи ВЫП
  - Символьная симуляция
  - Методы автоматического построения тестов

Построение тестов при помощи задачи ВЫП

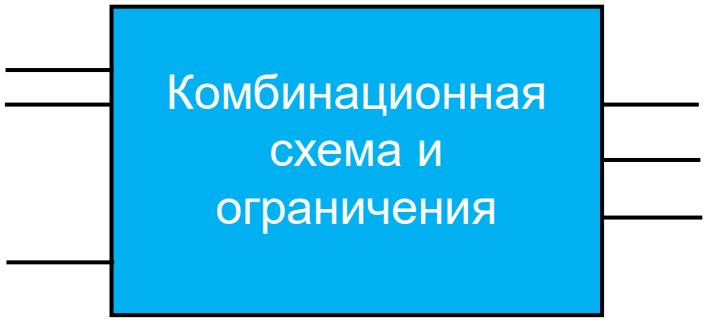
- Пусть задана RTL спецификация комбинационной схемы
- Симуляция (псевдо-случайные или направленные вектора)
- Достигнуто нужное покрытие кода?



```
module
    input ports, output ports, internal signals
    begin                                  . . . . .
        If (condition) then
                assign signal = function( );
        end if;
                . . . . .
    end module
```

Если нет, то анализируется булева функция или комбинационная схема, реализующая соответствующее описание

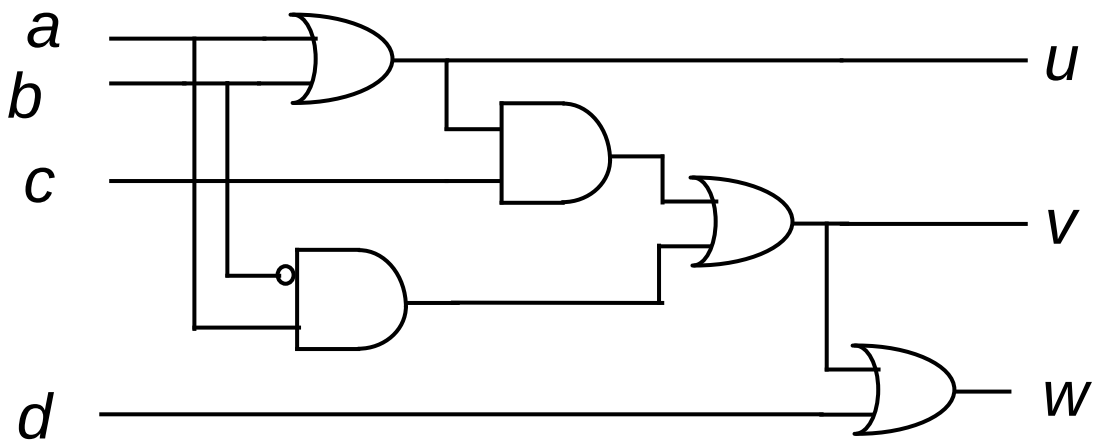
Решение задачи ВЫП при помощи BDD



- Добавить ограничения (изменяя схему)
- Построить BDD для каждого выхода (с учетом ограничений)
- Построить BDD для конъюнкции выходов
  - Если BDD – тождественный 0 (задача невыполнима)
  - Иначе: находим все выполнимые наборы - тесты.

Простой пример

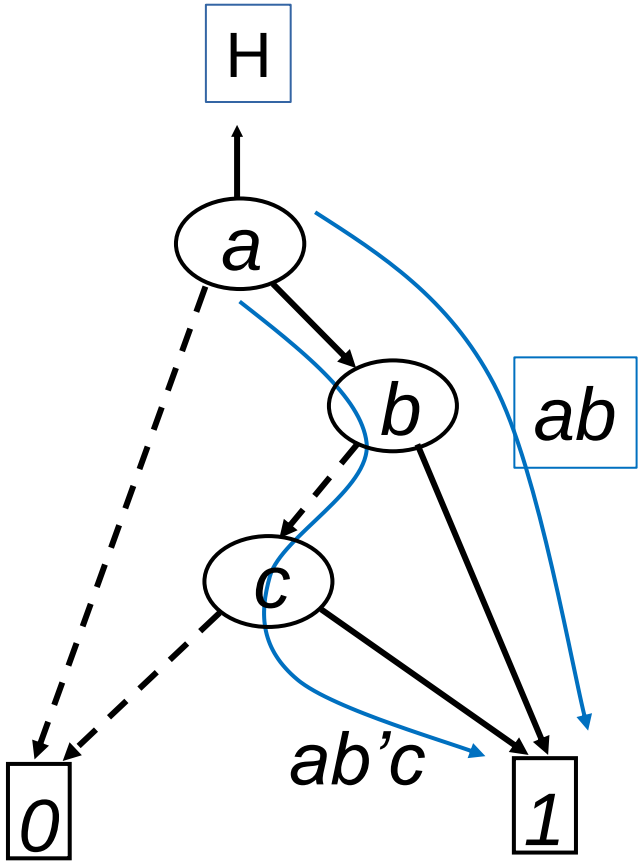
- *Дано*: требуемые значения на выходах схемы
- *Вычислить*: допустимые наборы на входах схемы



- Дано: *u=1, v=1, w=1*
- Значения на входах: *a,b,c,d* = ?

Построение теста

- Анализ ВЫП
  - H – результирующая BDD
    - Задает множество допустимых решений
  - Для проверки H = 1 (0) достаточно построить путь от вершины H до полюса 1 (0)
  - Путь однозначно определяет тестовый вектор



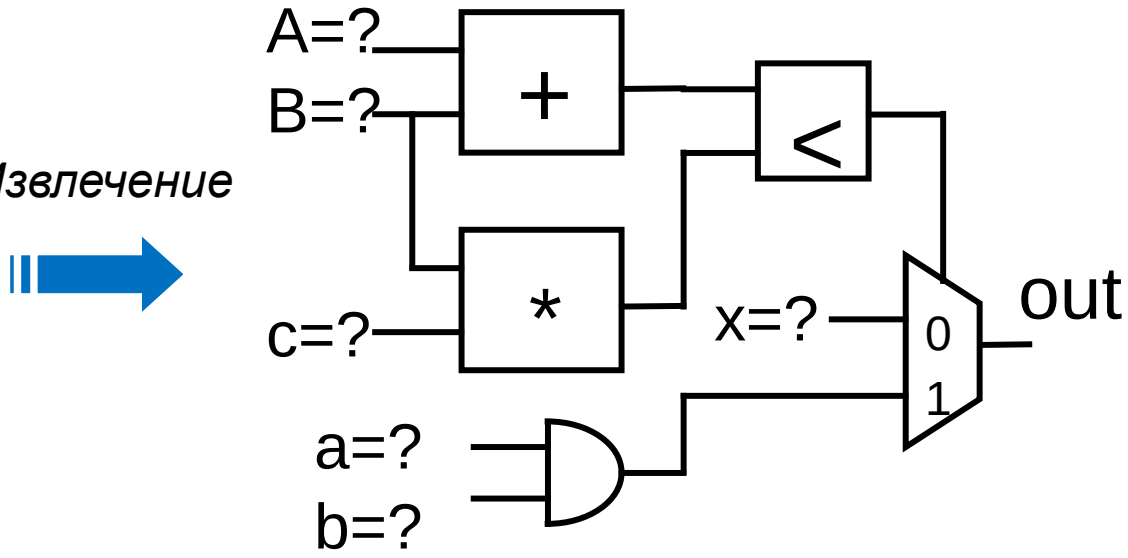
{1,1,-}, {1,0,1}

Функциональное тестирование при помощи символьной симуляции

- Схема рассматривается как программа
- Входы схемы – символьные переменные
- Для каждого пути в графе схемы (траектории выполнения программы) строится специальная задача ВЫП, которая позволяет найти значения символьных переменных на которых «срабатывает» данный путь

```
Module  DUT
   …
   always @ (clk) begin
     if (A+B < B*C)
       out = x;
     else
       out = a & b
   end
```

Извлечение



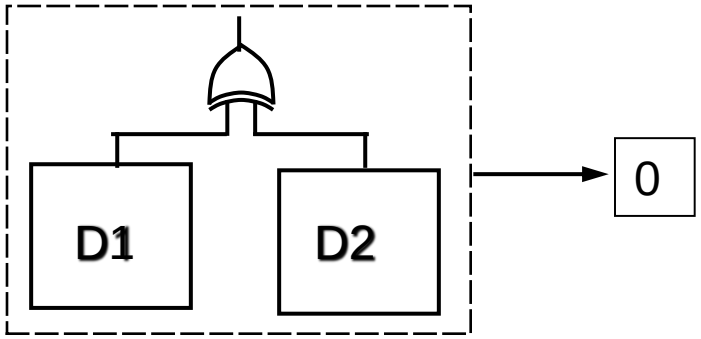
Формальная верификация

- Дедуктивные методы
  - Использование аксиом и правил для доказательства корректности системы
  - Требовательны к вычислительным ресурсам и нет гарантии, что процесс сойдется
- Проверка моделей
  - Методы автоматизированной проверки корректности распределенных систем (интегральные схемы, коммуникационные протоколы и т.д.)
- Проверка эквивалентности
  - Проверка, что две реализации устройства эквивалентны
  - Задача разрешима только для комбинационных схем

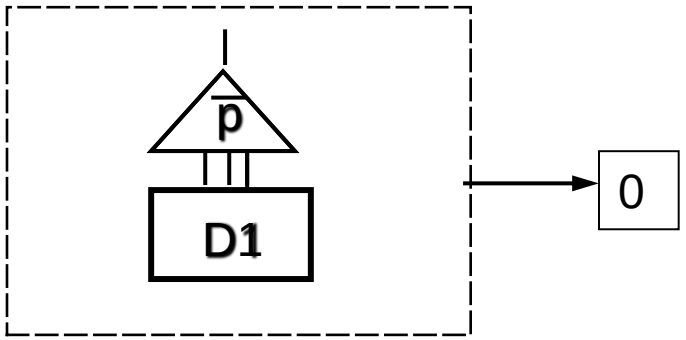
Формальная верификация

- Проверка эквивалентности
  - Для двух реализаций устройства строится «митра»
- Проверка свойств
  - Для заданной реализации и схемы, реализующий предикат выбранного свойства строится «митра»
- Для построенной «митры» решается задача ВЫП
  - Если устройство реализовано при помощи AIG, то можно использовать пакет ABC

Проверка эквивалентности



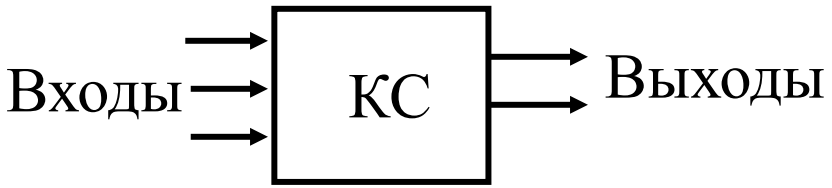
Проверка свойств



Задача проверки эквивалентности

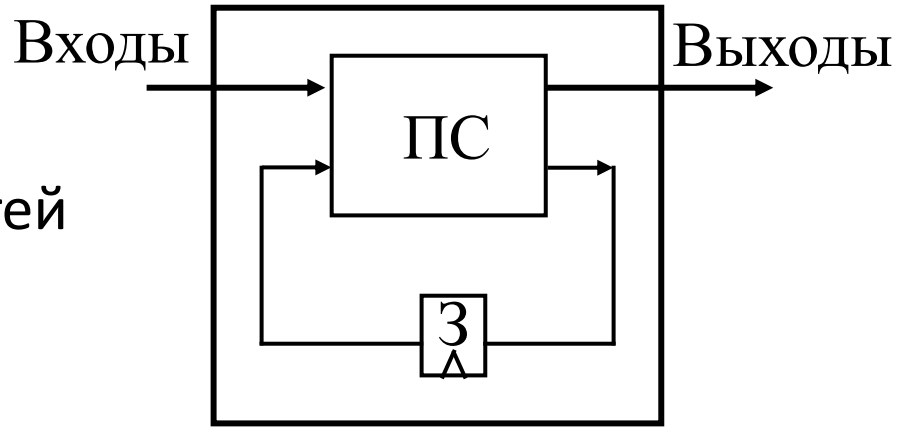
- Две схемы *функционально* эквивалентны, если их выходы выдают одинаковые значения

- Комбинационные схемы
  - Для всех наборов значений входных переменных

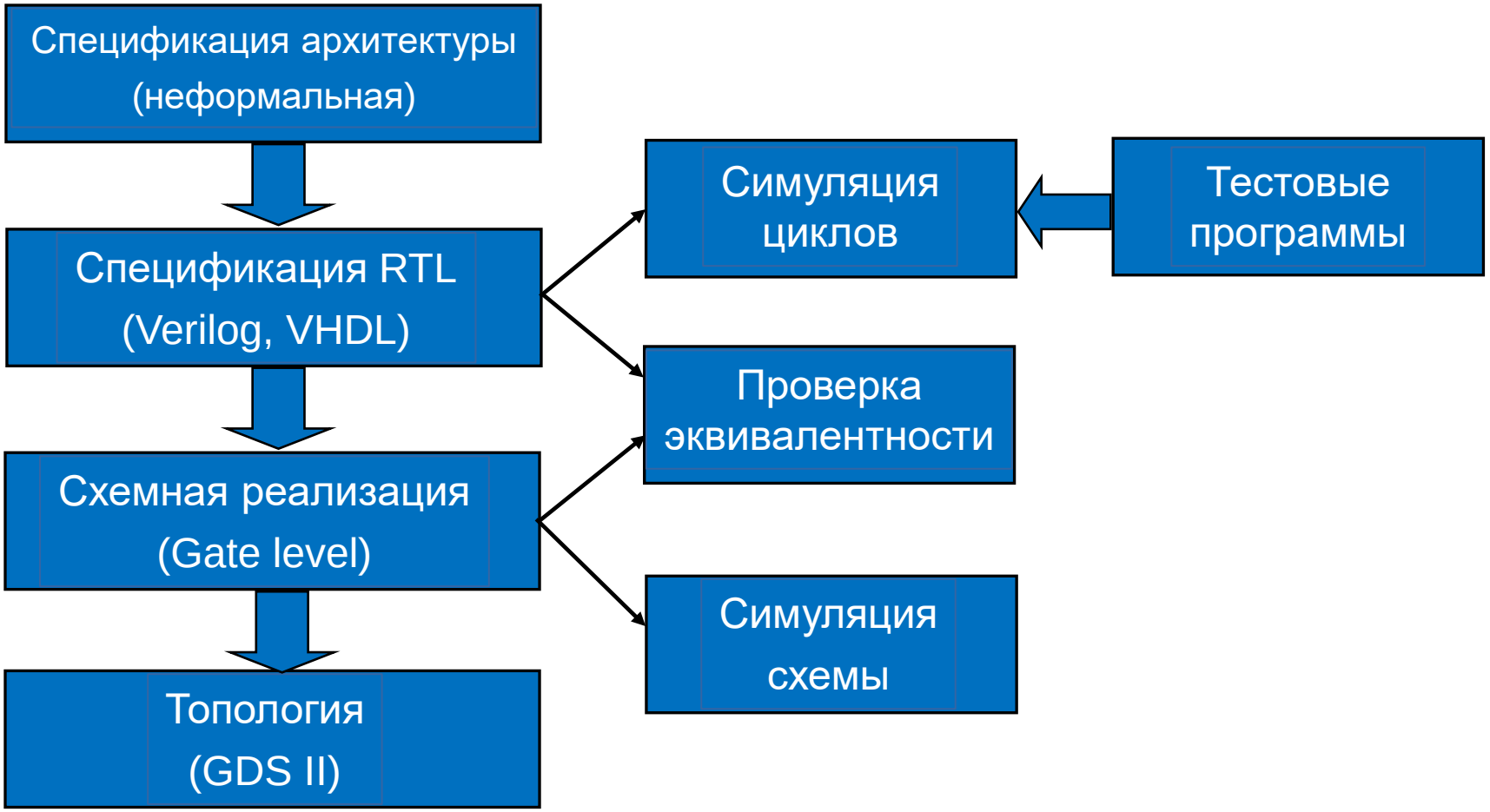


- Последовательные схемы

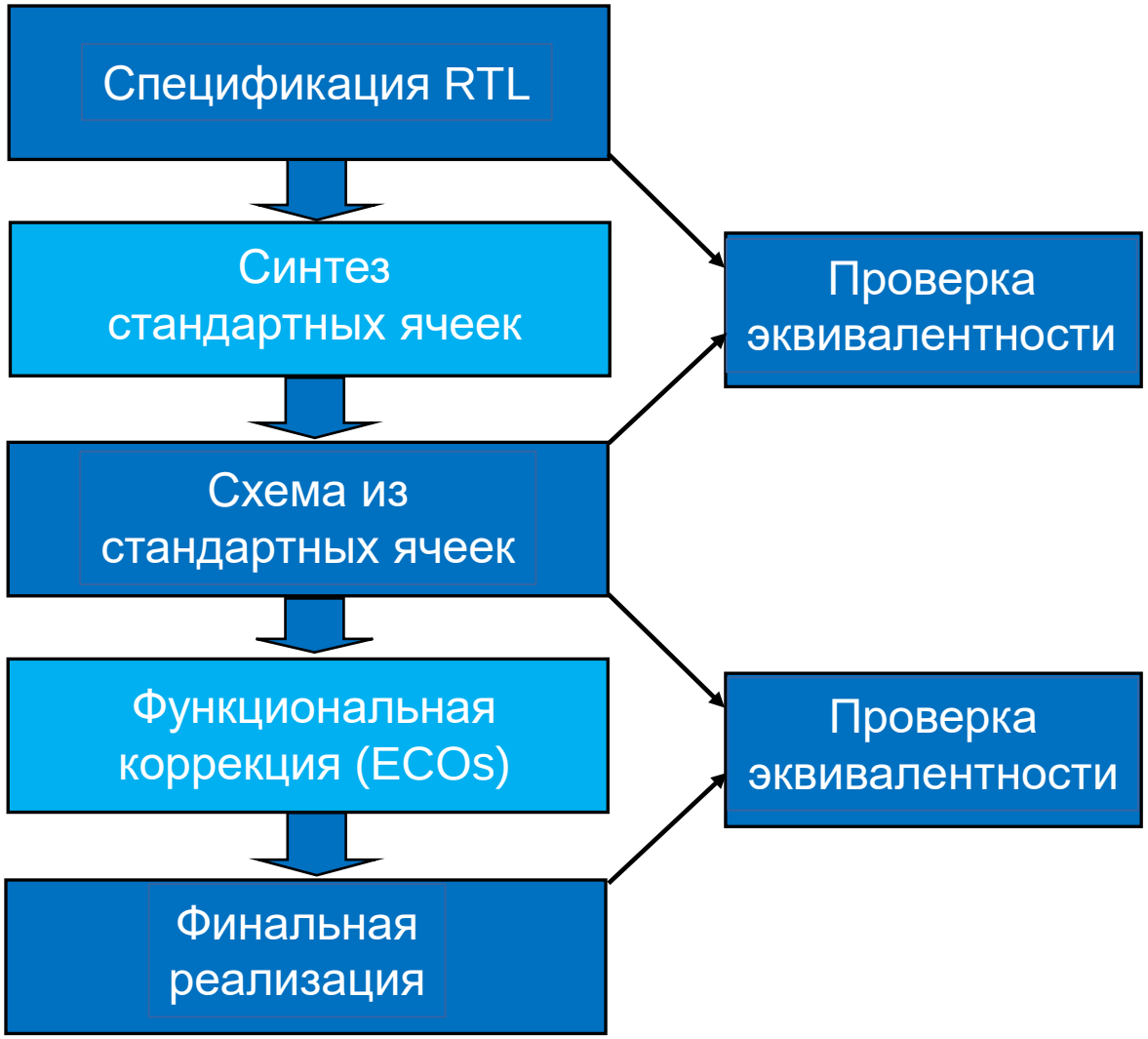
  – Для всех последовательностей наборов значений входных переменных



Задачи проверки эквивалентности при проектировании микропроцессоров



Задачи проверки эквивалентности при проектировании ASIC



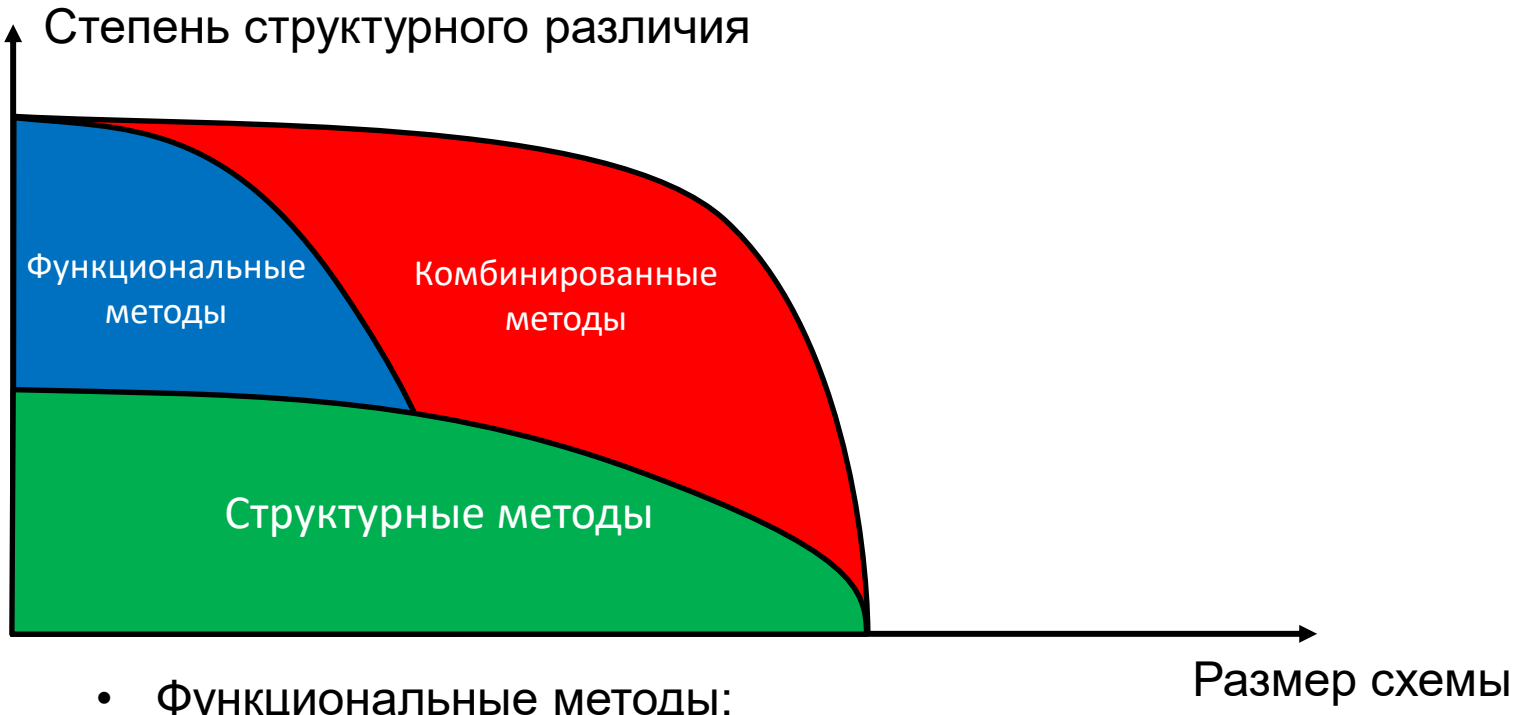
Проверка эквивалентности комбинационных схем

- Функциональный подход
  - Найти для выходов схем канонические представления.
  - Схемы эквивалентны, если совпадают представления соответствующих выходов.
  - Эффективные представления: BDD и их модификации.
- Структурный подход
  - Найти структурно похожие подсхемы.
  - Доказать эквивалентность на границах подсхем.
  - Найти зависимости между значениями входов и выходов подсхем (импликации).

Функциональный подход

- Для небольших схем можно построить разделяемую упорядоченную BDD и использовать ее для проверки эквивалентности
- Для больших схем не всегда возможно построить BDD
  - Разбить схему на подсхемы
  - Для каждой подсхемы построить BDD
  - Проверить эквивалентность на границах подсхем (метод разрезов)

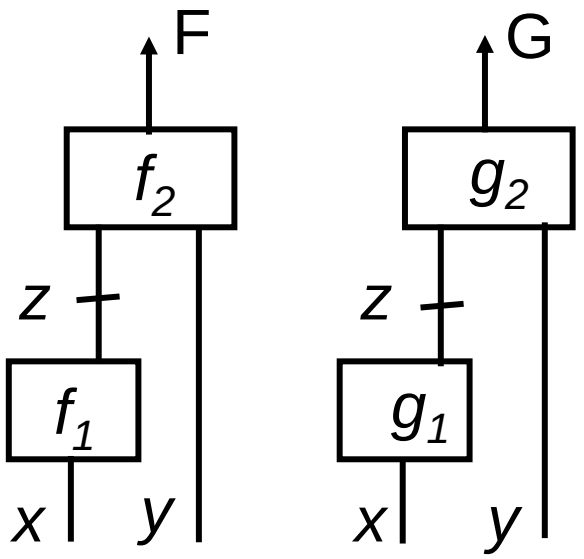
Методы проверки эквивалентности



- Функциональные методы:
  - Полный перебор (симуляция)
  - Решающие диаграммы
- Структурные методы:
  - Хэширование графов
  - Выполнимость и другие формальные методы

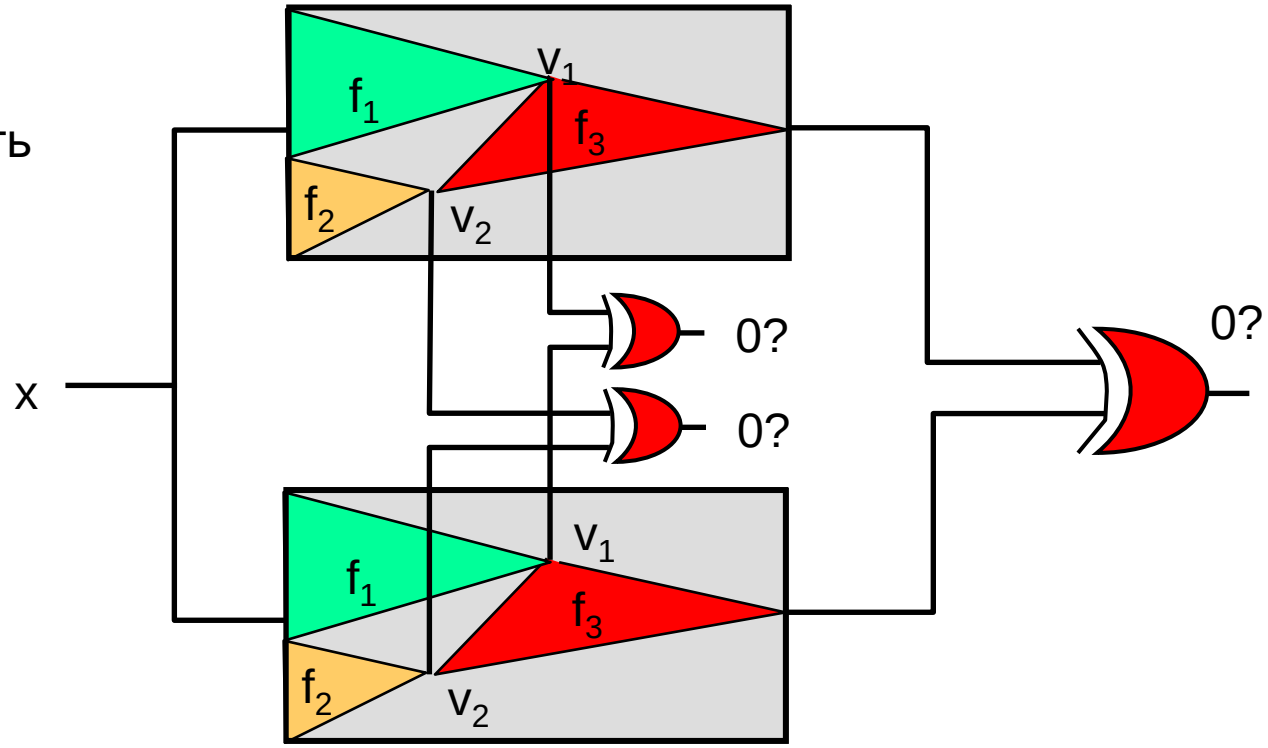
Функциональные методы – общая схема

- Каждая схема разбивается на подсхемы (блоки)
  - Каждый блок представляется в виде BDD (разделяемая BDD)
  - Определяются точки разрезов *(z)*
  - Проверяется эквивалентность блоков в точках разрезов, начиная от основных входов



Построение разрезов

Разрезы используются для того, чтобы разбить митру на части



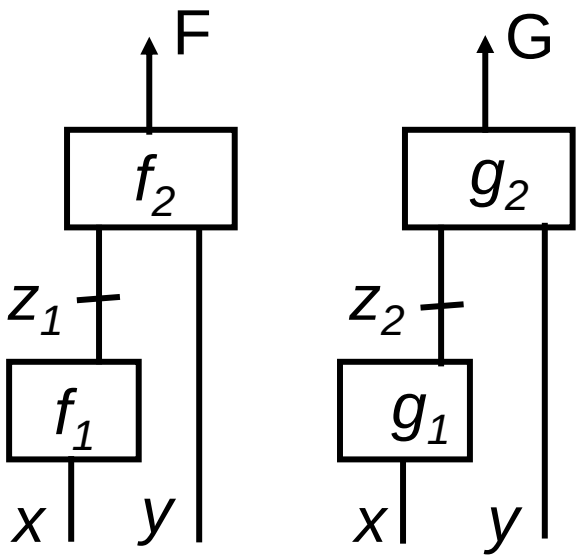
Нахождение разрезов:

- Для всех вершин находятся сигнатуры при помощи симуляции (случайной)
- Сигнатуры упорядочиваются и выбираются разрезы
- Итеративно производится верификация и уточнение разрезов
- Верификация выходов

Задача разрешения разрезов

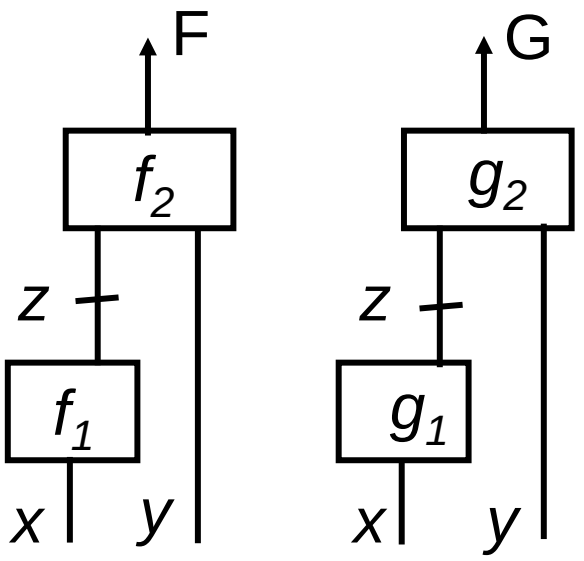
- Если все пары разрезов (z_1,z_2) эквивалентны
  - Исходные функции F и G эквивалентны
- Если промежуточные функции (f_2,g_2) неэквивалентны
  - Функции (F,G) могут быть эквивалентными
  - Ложно отрицательный результат

- Почему получается ложно отрицательный результат?
  - Функции представлены при помощи вспомогательных (промежуточных) переменных
  - Для доказательства или опровержения эквивалентности нужно функции выразить (в BDD) через входные переменные



Задача разрешения разрезов

- Пусть $f_1(x)=g_1(x)\ \forall x$
  - Если $f_2(z,y) \equiv g_2(z,y),\ \forall z,y,$ то $f_2(f_1(x),y) \equiv g_2(f_1(x),y) \Rightarrow F \equiv G$
  - Если $f_2(z,y) \neq g_2(z,y),\ \forall z,y \not\Rightarrow f_2(f_1(x),y) \neq g_2(f_1(x),y) \Rightarrow F \neq G$



Невозможно определить

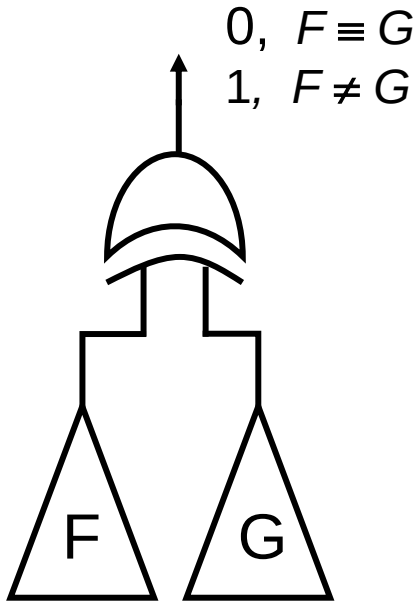
$F \equiv G$ или нет

- *Ложно отрицательный результат*
  - Две схемы эквивалентны, но алгоритм верификации считает их не эквивалентными

Методы разрешения разрезов

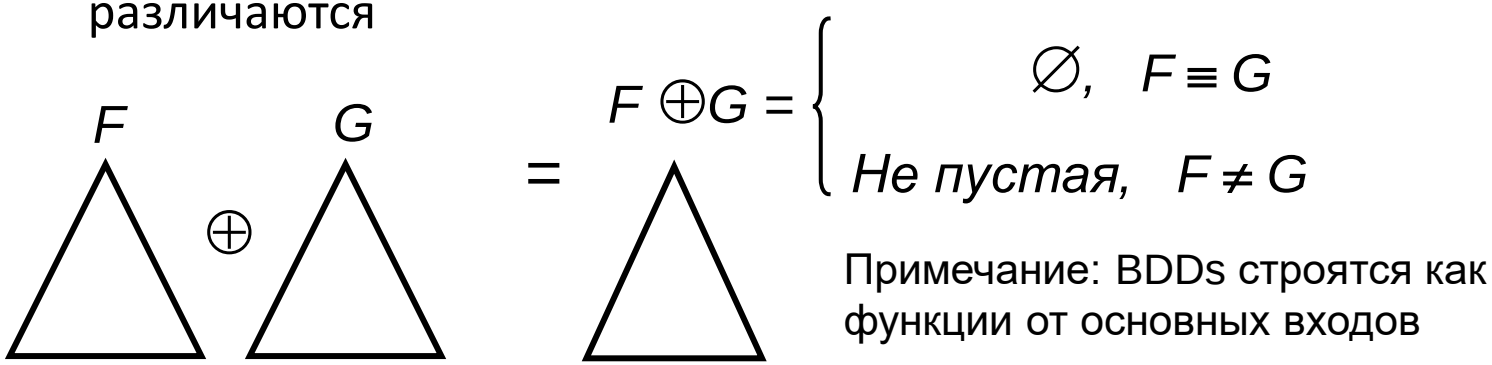
- Как проверять отрицательный результат?

- Метод 1: Создается «митра» для двух потенциально эквивалентных вершин
  - Производится автоматическое тестирование нулевой константной неисправности на выходе «митры»
  - Если тест найдет, то $F \neq G$
  - Метод эффективен для, если существует тестовый вектор
  - Метод неэффективен, когда тестового вектора не существует



Методы разрешения разрезов

- Метод 2: Создать BDD для $F \oplus G$
  - Проводится анализ BDD
    - Если BDD для $F \oplus G = \varnothing$, то схемы эквивалентны (ложно отрицательный результат)
    - Если BDD для $F \oplus G \neq \varnothing$, то есть набор на котором схемы различаются

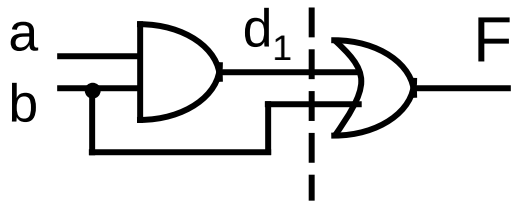


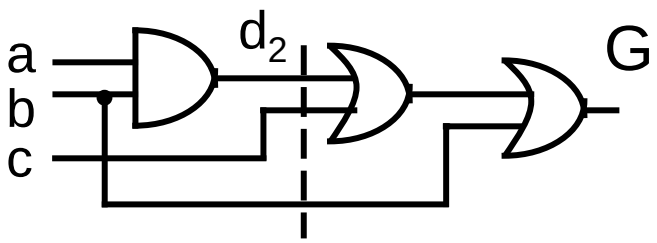
Примечание: BDDs строятся как функции от основных входов

  - Если BDD не пустая, то легко построить тестовый вектор
  - Данный метод является эффективным для ложно отрицательных результатов (проверка BDD на пустоту – производится за O(1))

Структурные методы

- Пусть заданы две *схемы*, каждая имеет свою структуру
  - Определение “похожих” внутренних верши, разрезов
  - Извлечение информации о эквивалентности элементов (например, подсхем) рассматриваемых схем
- Проблема ложно отрицательных результатов
  - F ≡ G, но структурно различны (например, зависят от разных наборов переменных)
  - Верификационный алгоритм не может не имеет достаточной информации, чтобы объявить F и G эквивалентными

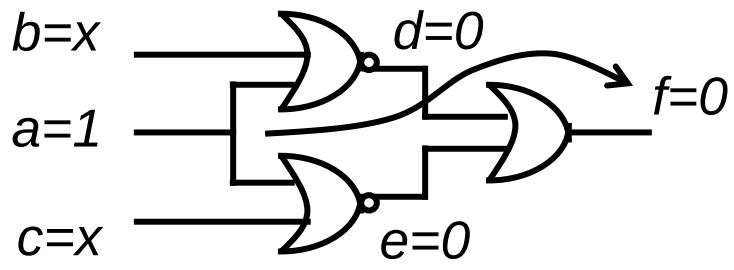


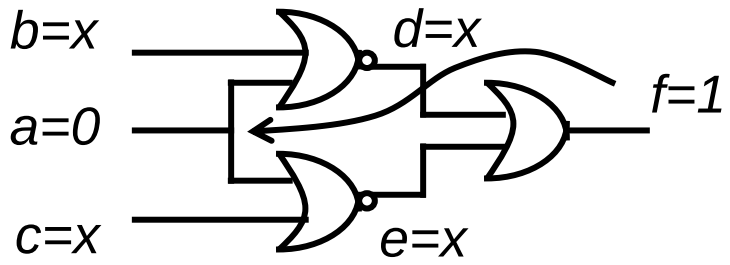


- Решение: использовать методов на базе BDD или автоматической генерации тестов-based
- Использование методов построения импликант (логический вывод ограничений).

Импликанты

- Различные методы извлечения и использования информации о внутренних связях между вершинами схем, которые позволяют ускорить или упростить процесс верификации
- Импликанты – прямые и обратные (косвенные)



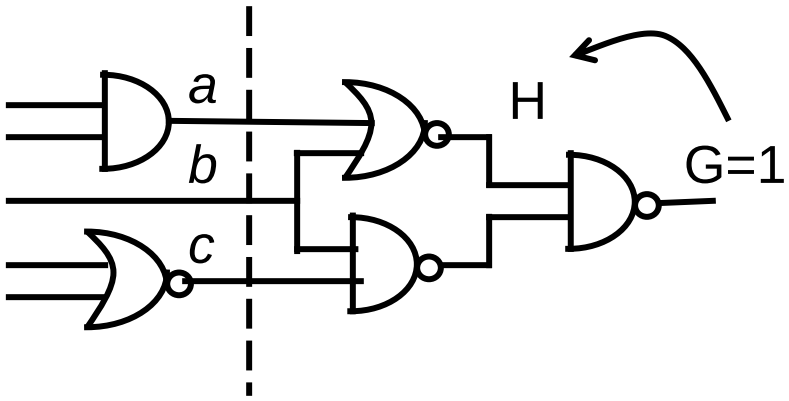


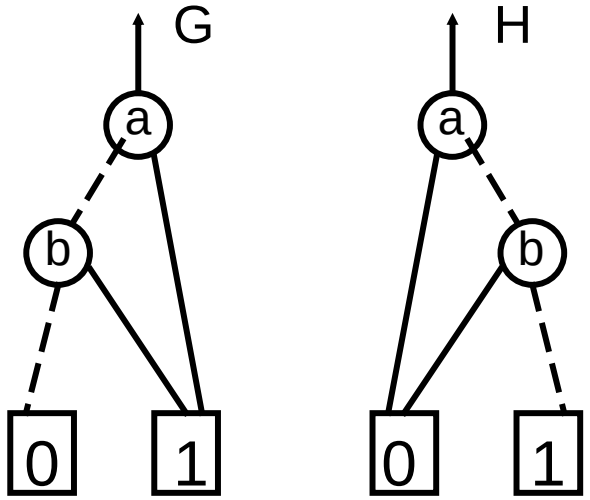
Прямая: $a=1 \Rightarrow f=0$

Обратная: $f=1 \Rightarrow a=0$

Построение обратных импликант

- Логический вывод (обучение)
  - Процесс поиска/построения обратных импликант
  - Рекурсивное обучение
    - Рекурсивно анализируется каждый вывод
  - Функциональное обучение
    - Используется BDDs для нахождения обратных импликант





$G=1 \Rightarrow H=0$

Примеры функционального обучения

- Существует целый ряд способов функционального обучения, позволяющих проверить конкретную импликанту G=1 ⇒ H=0
  - Построить BDD для G • H’
    - Если функция выполнима, то импликанта корректна и по BDD можно построить тестовый вектор
    - Иначе импликанта неверна.
  - В силу того, что G=1 ⇒ H=0 ≡ (G’+H’)=1. Можно построить BDD для (G’+H’)
    - Импликанта корректна, если (G’+H’)=1