Математические модели и методы логического синтеза СБИС

Осень 2022

Лекция 6

План лекции

- Конъюнктивно-инверсные графы (англ. And-inverter graph, AIG)
- Приведенные AIG, структурное хеширование AIG
- Оптимизация AIG, AIG rewriting

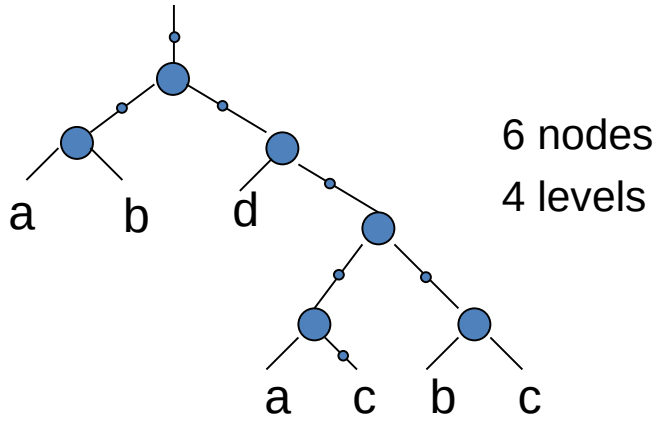
And-Invert Graph (AIG)

- AIG – граф специального вида, который представляет СФЭ в базисе Поста (конъюнкция, отрицание)
- Любая ФАЛ может быть представлена при помощи AIGs, при этом:
  - Для многих ФАЛ, встречающихся на практике AIG имеют меньший размер, нежели соответствующие BDD
  - AIG является структурным представлением рассматриваемой ФАЛ
  - Размер AIG хорошо соотноситься с размером результирующей схемы
- AIG не является каноничным представлением
  - Для одной ФАЛ может быть несколько структурно отличающихся представлений в виде AIG
  - Операция приведения позволяет сделать AIG «частично» каноническими (англ. Functionally Reduced And-Inver Graph, FRAIG)

Определение AIG

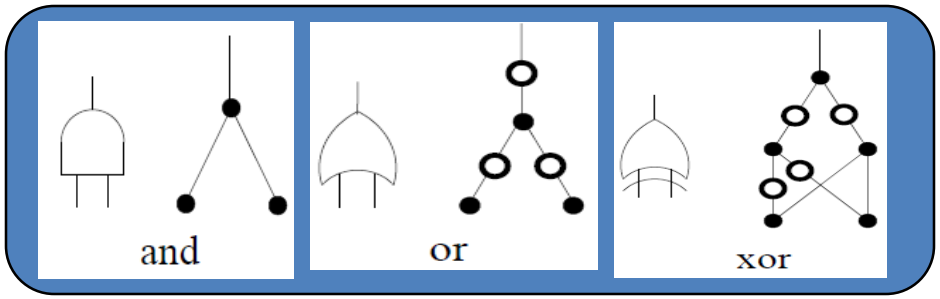
ab \ cd	00	01	11	10
00	0	0	1	0
01	0	0	1	1
11	0	1	1	0
10	0	0	1	0

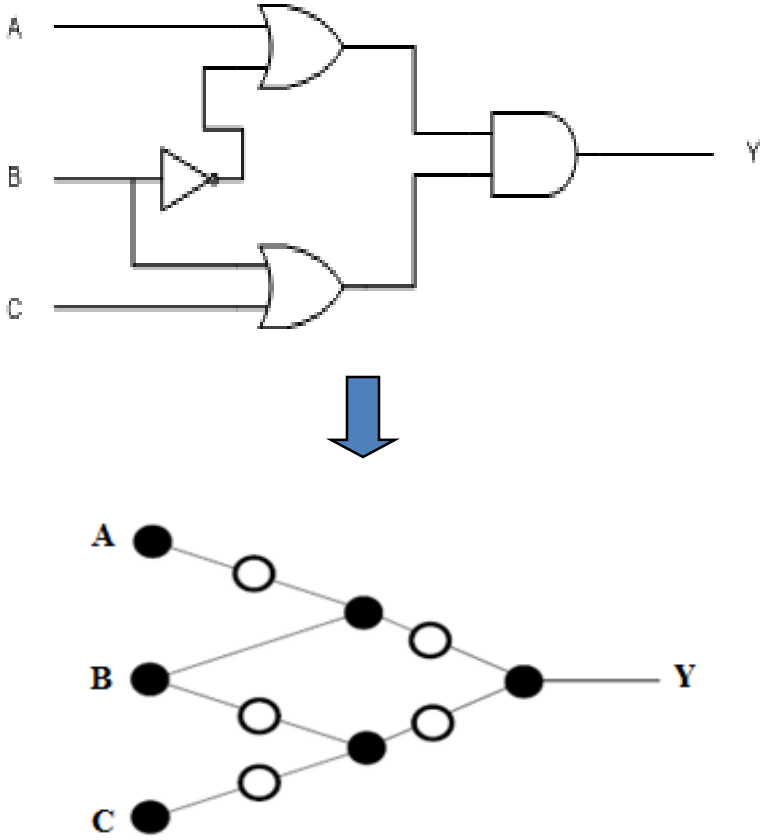
$$F(a,b,c,d) = ab \vee d(a\bar{c} \vee bc)$$



Построение AIG

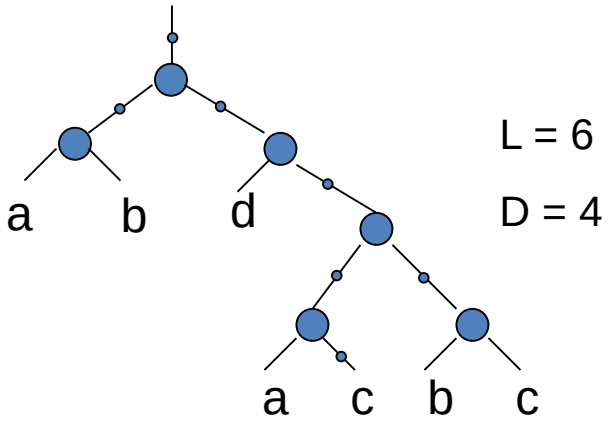
- AIGs могут быть построены непосредственно из КЛС или аналогичного представления и их размер может оптимизирован при помощи операции приведения





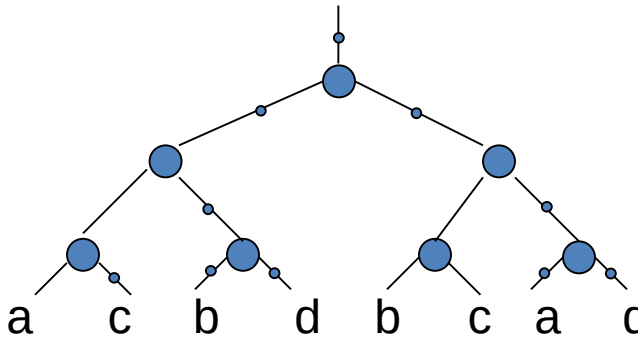
AIG не являются каноничными

- AIG не каноничны
  - Для одной ФАЛ может быть несколько структурно отличающихся представлений в виде AIG
  - BDD – каноничны для заданного порядка следования переменных при разложении
  - AIG можно сделать «частично» каноническими (“canonical enough”, A. Mishchenko)



L = 6

D = 4



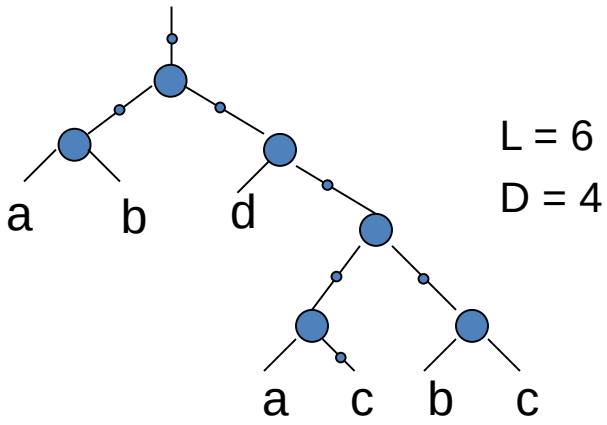
L = 7

D = 3

Пример AIG

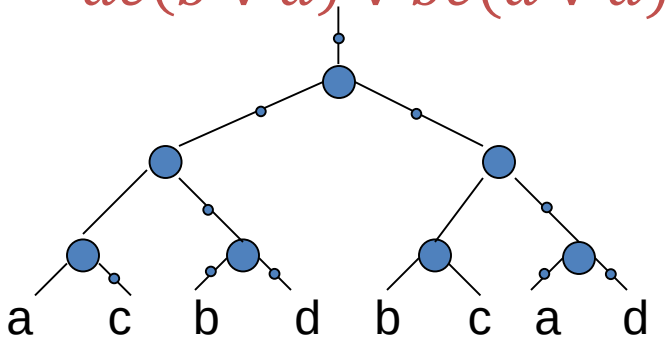
cd \ ab	00	01	11	10
00	0	0	1	0
01	0	0	1	1
11	0	1	1	0
10	0	0	1	0

$$F(a,b,c,d) = ab \vee d(a\overline{c} \vee bc)$$



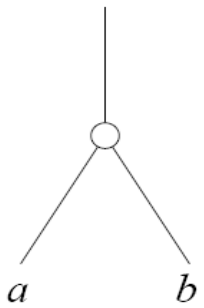
cd \ ab	00	01	11	10
00	0	0	1	0
01	0	0	1	1
11	0	1	1	0
10	0	0	1	0

$$F(a,b,c,d) = a\overline{c}\overline{(\overline{b} \cdot \overline{d})} \vee cb\overline{(\overline{a} \cdot \overline{d})}$$
$$= a\overline{c}(b \vee d) \vee bc(a \vee d)$$

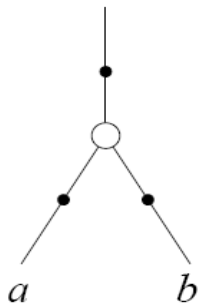


Построение AIG

- AIG может быть построена по любой логической схеме (теорема перехода)
  - Отрицания $\neg a$ $\neg a$
  - Конъюнкция $a \wedge b$ (ab) $a \wedge b$
  - Дизъюнкция $a \vee b$ ($a+b$) $\neg(\neg a \wedge \neg b)$
  - Импликация $a \Rightarrow b$ $\neg(a \wedge \neg b)$
  - Эквивалентность $a \sim b$ $\neg(a \wedge \neg b) \wedge \neg(\neg a \wedge b)$
  - *Сумма по модулю 2* $a \oplus b$ $\neg(\neg(a \wedge \neg b) \wedge \neg(\neg a \wedge b))$



$a \wedge b$

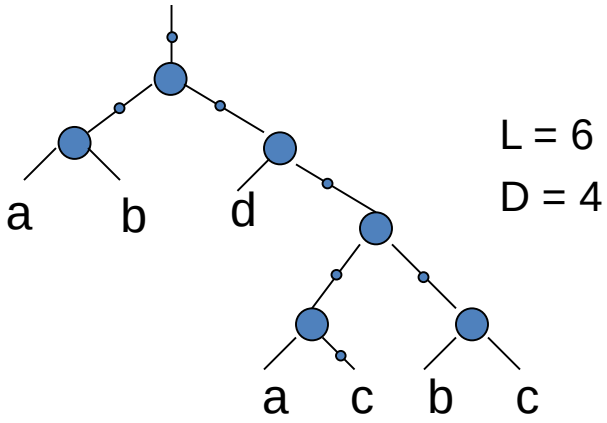


$a \vee b$

Функционалы качества AIG

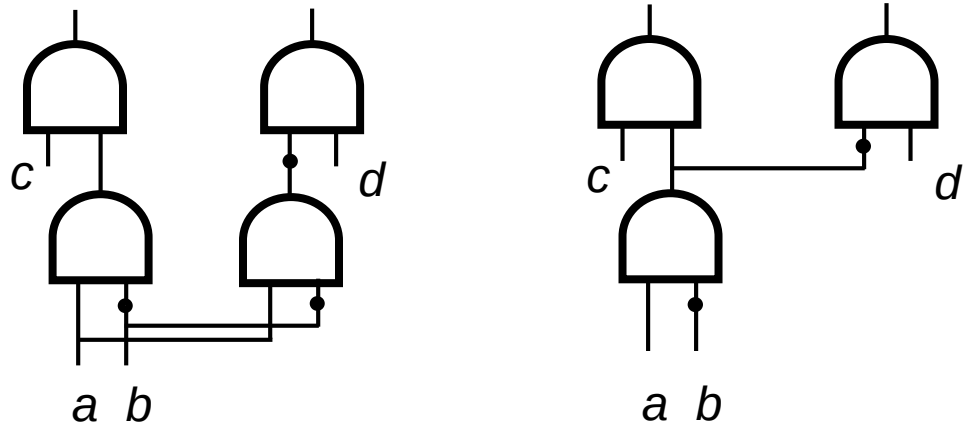
- Размер AIG (сложность)
  - Число вершин, соответствующее числу элементов конъюнкции

- Глубина AIG
  - Длина максимального пути от входов до выхода AIG = максимальное число элементов конъюнкции на пути от некоторого входа до выхода AIG
  - Элементы отрицания игнорируются при подсчете глубины



Структурное хеширование (Strashing)

- Построени AIG без структурного хэширования
  - Вершины добавляют по одной в схему без каких либо дополнительных проверок на наличие эквивалентных вершин в схеме
- Одноуровневое структурное хэширование
  - При добавлении новых вершин осуществляется проверка в специальной таблице существования в схеме вершин с идентичной входной сигнатурой (имена и полярность входов совпадают)
  - Если такая вершина найдена в таблице, тогда возвращается указатель на нее, иначе создается новая вершина

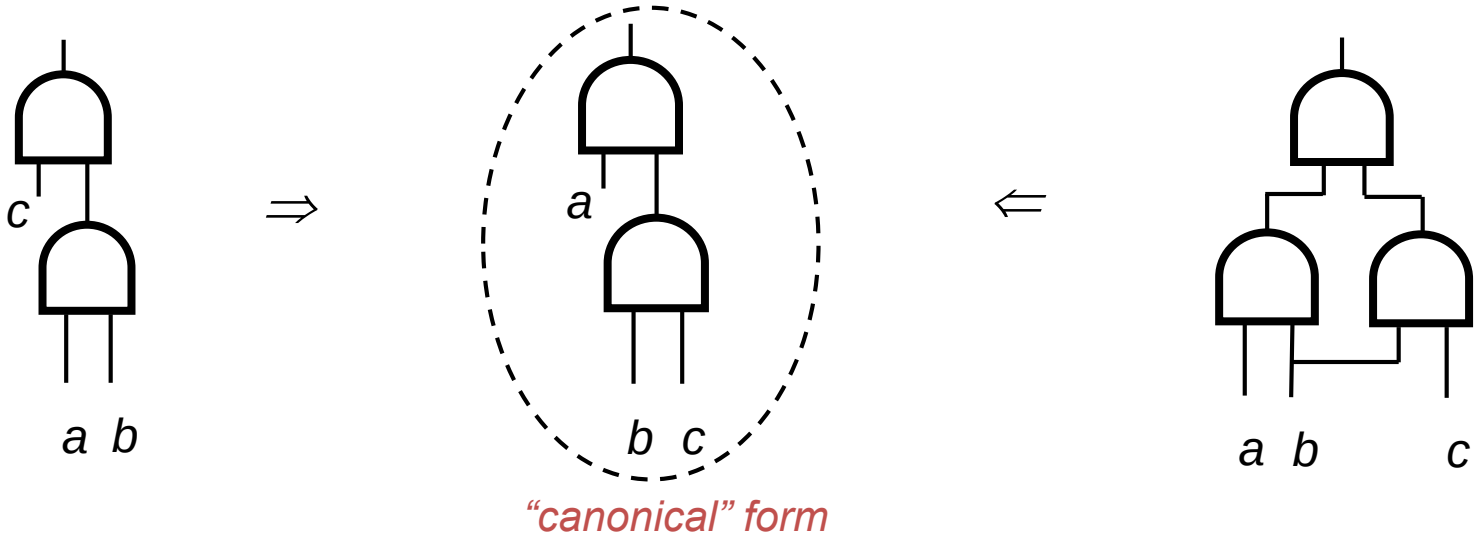


Алгоритм построения AIG

```
Aig_Node * OperationAnd( Aig_Man * p, Aig_Node * n1, Aig_Nodee * n2 )
{
    Aig_Node * res, * cand, * temp;   Aig_NodeArray * class;
    /*** trivial cases ***/
    if ( n1 == n2 )                                          return n1;
    if ( n1 == NOT(n2) )                                     return 0;
    if ( n1 == const )                                       return 0  or  n2;
    if ( n2 == const )                                       return 0  or  n1;
    if ( n1 < n2 )  { /*** swap the arguments ***/
          temp = n1; n1 = n2; n2 = temp;
    }
    /*** one level structural hashing ***/
    res = HashTableLookup( p->pTableStructure, n1, n2 );
    if ( res )                                               return res;
    res = CreateNode( p, n1, n2 );
    HashTableAdd( p->pTableStructure, res );
    if ( p->FlagUseOneLevelHashing )                         return res;
```

Двухуровневое структурное хеширование

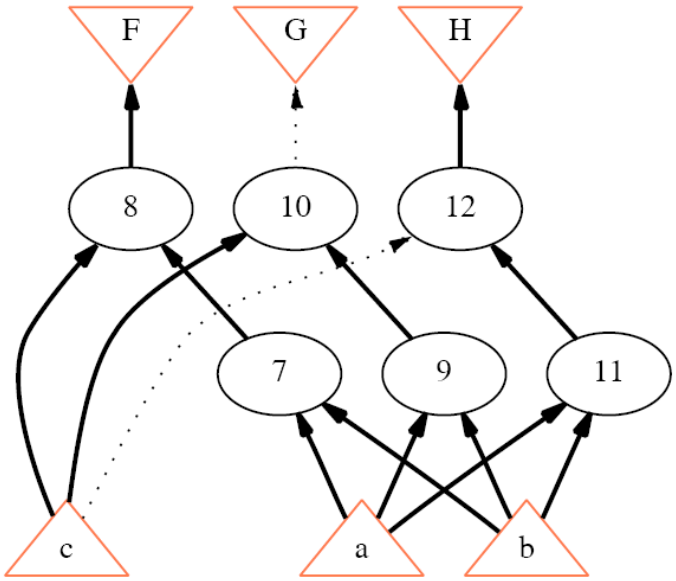
- При добавлении новой вершины:
  - Рассматриваются два уровня вершин-предков добавляемой вершины
  - Среди всех AIG глубины два реализующих заданную булеву функцию выбирается некоторая схема, которая объявляется каноническим представителем
  - Таким образом обеспечивается частичная каноничность всей схемы



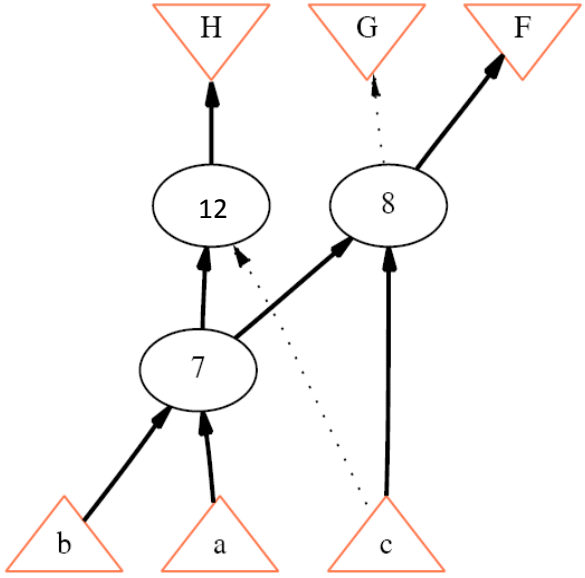
Пример структурного хэширования

F = abc *G = (abc)'* *H = abc'*

Начальное AIG

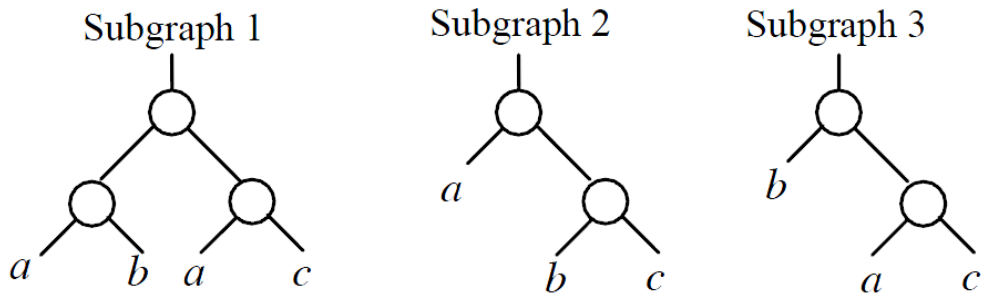


AIG после структурного хэширования



Функциональное приведение AIG

- AIG не являются каноническими – структурно различные AIG могут быть функционально эквивалентными, а также могут содержать функционально эквивалентные внутренние вершины с различными входными сигнатурами



Different AIG structures for function $F = abc$.

- Операции над такими AIG могут быть неэффективными
- Обнаружение и слияние эквивалентных вершин называется операцией приведения (functional reduction).

“DAG-Aware AIG Rewriting A Fresh Look at Combinational Logic Synthesis” - Alan Mishchenko, Satrajit Chatterjee, Roland Jiang, Robert Brayton, DAC’06 Proceedings of the 43rd annual Design Automation Conference

Функциональное приведение AIG

AIGs изначально строятся при помощи структурного хэширования (*strashing*) а потом дополнительно модифицируются при помощи алгоритмов приведения

- Приведение на основе BDD (BDD Sweeping) [1]
  - Для всех внутренних вершин схемы строится BDD
  - Вершины схемы, которые отображаются в одну вершину BDD отождествляются
  - Размер BDD ограничивает применимость этого метода
- Приведение на основе ВЫП (SAT Sweeping) [2]
  - Эквивалентность вершин обнаруживается при помощи решения топологически упорядоченного набора задач ВЫП
  - Кандидаты определяются на основе симуляции схемы

[1] A. Kuehlmann, et.al., “Robust boolean reasoning for equivalence checking and functional property verification”, *IEEE Trans. CAD, Vol. 21(12), 2002*

[2] A. Kuehlmann, “Dynamic Transition Relation Simplification for Bounded Property Checking”. *Proc. ICCAD ‘04.*

Функциональное приведение AIG

- Этапы алгоритма:
  - При добавлении новой вершины производится структурное хэширование
  - При этом осуществляется проверка на наличие в схеме вершины с идентичной входной сигнатурой
    - Если такая вершина существует, то возвращается указатель на нее
    - Иначе, создается новая вершина
  - Произвести структурное приведение
    - При помощи симуляции проверить может ли новая вершина быть эквивалентной уже существующей
    - Если класс-кандидат найден произвести формальную проверку эквивалентности

- В результате получается приведенная AIG, которая является условно канонической
  - В схеме нет эквивалентных вершин
  - Структурное представление каждой функции не фиксировано
    - Структура AIG зависит от порядка добавления вершин
    - AIG может быть приведена к каноническому виду при помощи соответствующей библиотеки канонических AIG

Функциональное приведение AIG

```
Aig_Node * OperationAnd( Aig_Man * p, Aig_Node * n1, Aig_Nodee * n2 )
{
    Aig_Node * res, * cand, * temp;   Aig_NodeArray * class;
    /*** trivial cases ***/
    if ( n1 == n2 )                                          return n1;
    if ( n1 == NOT(n2) )                                     return 0;
    if ( n1 == const )                                       return 0  or  n2;
    if ( n2 == const )                                       return 0  or  n1;
    if ( n1 < n2 )  { /*** swap the arguments ***/
          temp = n1; n1 = n2; n2 = temp;
    }
    /*** one level structural hashing ***/
    res = HashTableLookup( p->pTableStructure, n1, n2 );
    if ( res )                                               return res;
    res = CreateNode( p, n1, n2 );
    HashTableAdd( p->pTableStructure, res );
    if ( p->FlagUseOneLevelHashing )                         return res;
   /*** functional reduction ***/
    class = HashTableLookup( p->pTableSimulation, n1, n2 );
    if ( class == NULL ) {
        class = CreateNewSimulationClass( res );
        HashTableAdd( p-> pTableSimulation, class );         return res;
    }
    for each node cand in class
        if ( CheckFunctionalEquivalence( cand, res ) ) {
            AddNodeToEquivalenceClass( class, res );         return cand;
        }
    AddNodeToSimulationClass( class, res );                  return res;
}
```

Оптимизация AIG (Rewriting)

Рассматриваются разрезы заданной ширины (например, 4)

- Разрезы схемы ищутся в топологическом порядке от входов к выходам схемы
- Для внутренней вершины *n* с двумя входами *a* и *b* множество разрезов *C*(*n*) рассчитывается на основе объединения множеств разрезов для *a* и *b.*
- Для каждого разреза выбранной вершины рассматривается все оптимальные реализации (оптимальные схемы, в библиотеке оптимальных схем) и выбирается реализация, которая ведет к максимальному уменьшению числа вершин.

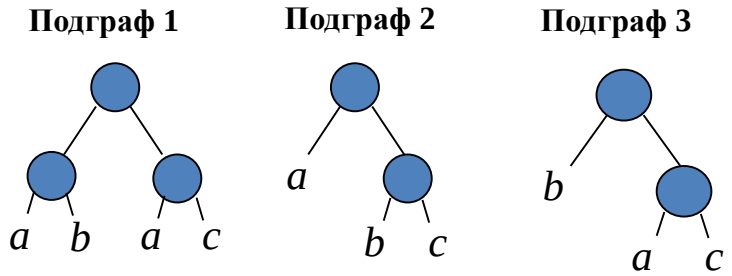
Оптимизация AIG с учетом задержки

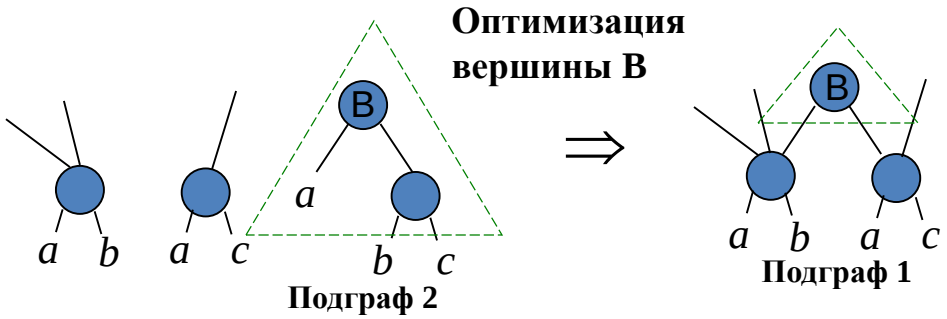
- Разложение AIG (AIG refactoring)
- Оптимизация глубины AIG (AIG balancing)

Оптимизация AIG (Rewriting)

- Оптимиация AIG уменьшает количество вершин, при этом не увеличивая глубину схемы
- Предвычисление AIG подграфов
  - Рассмотрим *f = abc*

Оптимизация AIG подграфов





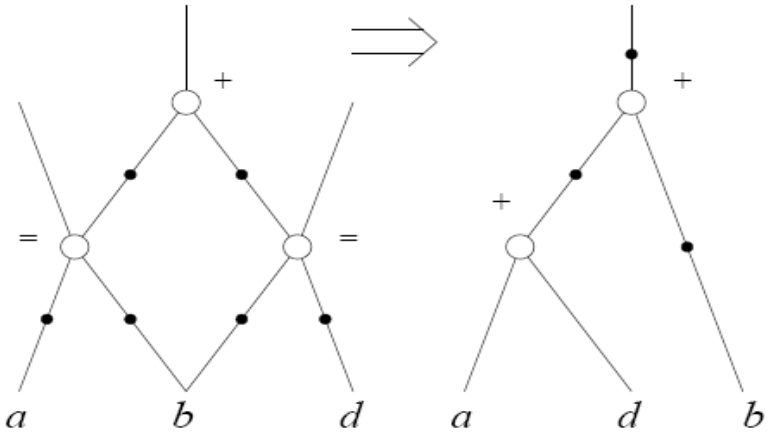
В обоих случая сложность уменьшается

Алгоритм оптимизации AIG

```
Rewriting( network AIG, hash table PrecomputedStructures, bool UseZeroCost )
{
    for each node N in the AIG in the topological order {
        for each 4-input cut C of node N computed using cut enumeration {
            F = Boolean function of N in terms of the leaves of C
            PossibleStructures = HashTableLookup( PrecomputedStructures, F );
            // find the best logic structure for rewriting
            BestS = NULL; BestGain = -1;
            for each structure S in PossibleStructures {
                NodesSaved = DereferenceNode( AIG, N );
                NodesAdded = ReferenceNode( AIG, S );
                Gain = NodesSaved – NodesAdded;
                Dereference( AIG, S );  Reference( AIG, N );
                if ( Gain > 0 || (Gain = 0 && UseZeroCost) )
                    if (  BestS = NULL ||  BestGain < Gain )
                        BestS = S; BestGain = Gain;
            }
            if ( BestS == NULL ) continue;
            // use the best logic structure to update the netlist
            NodesSaved = DereferenceNode( AIG, N );
            NodesAdded = ReferenceNode( AIG, S );
            assert( BestGain = NodesSaved – NodesAdded );
        }
    }
}
```

Локальная оптимизация AIG

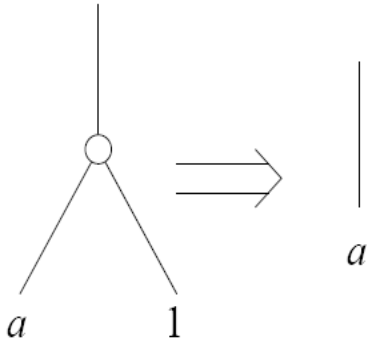
- AIG optimization is based on AIG rewriting, from one form to a simpler form



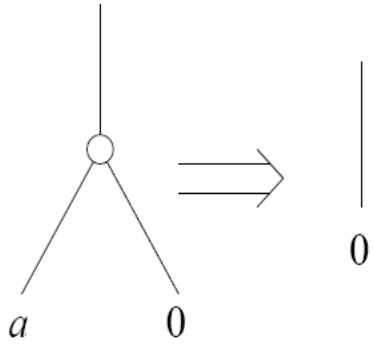
distributivity law

(a+b)(b+d) = ad+b

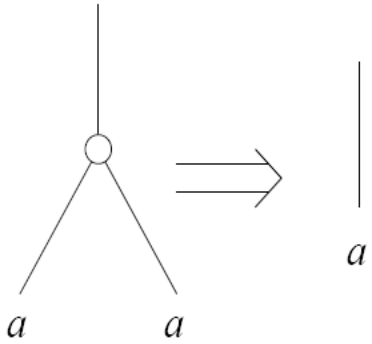
Примеры оптимизации AIG



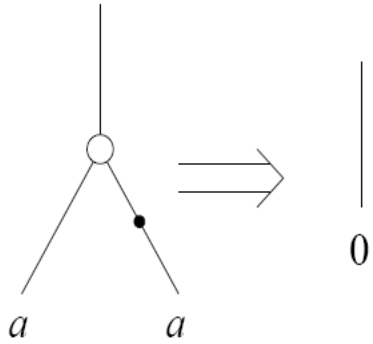
*a * 1 = 1*



*a * 0 = 0*

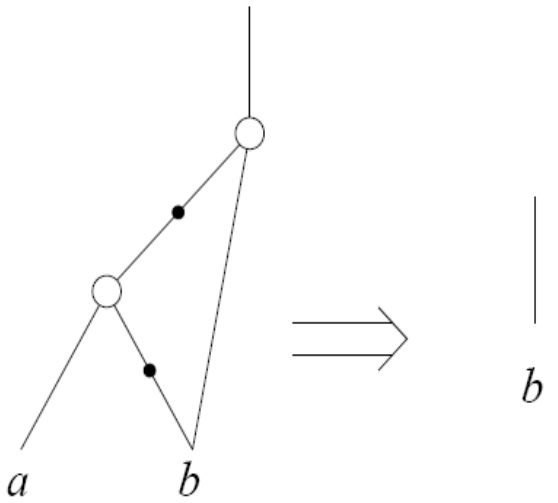


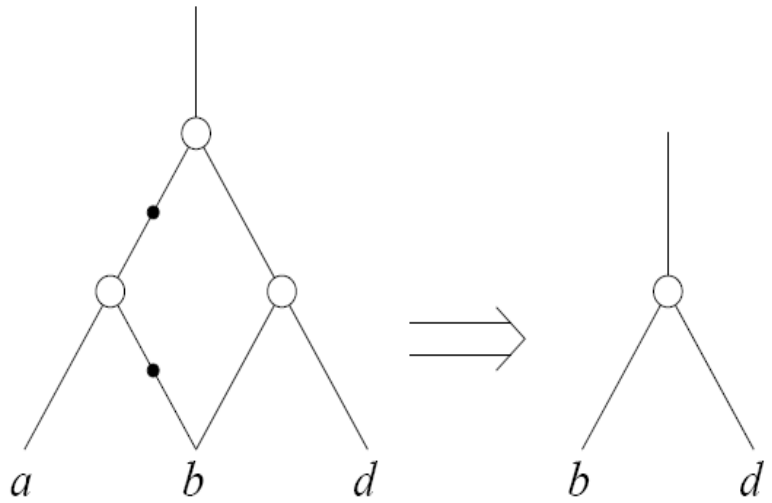
*a * a = a*



*a * ¬a = 0*

Примеры оптимизации AIG



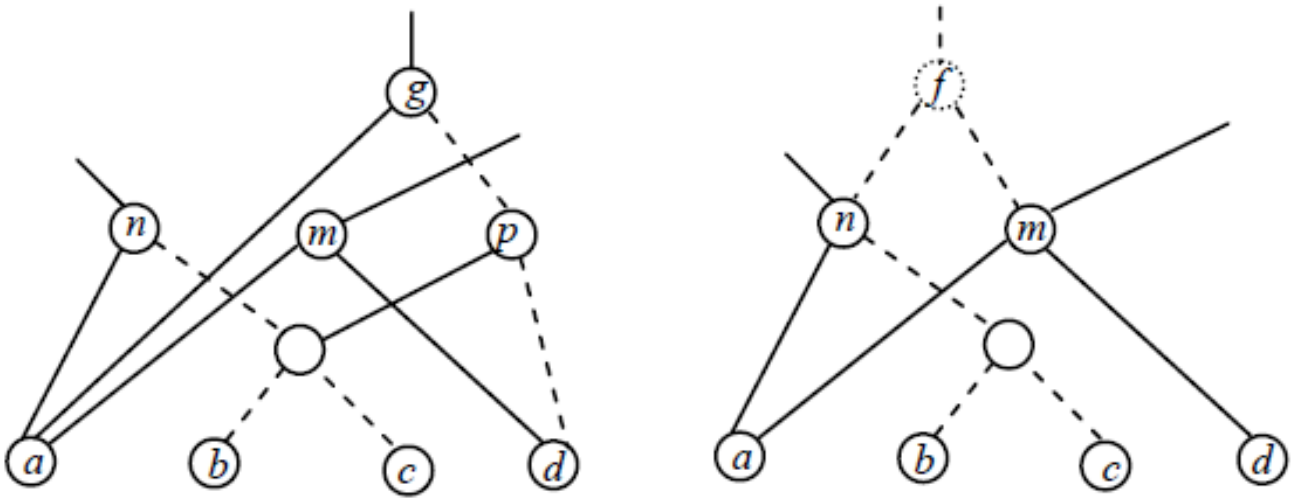


(¬a+b)b = b

((¬a+b)b) d = bd

Разложение(Resubstitution)

- Задача выразить функцию, реализуемую в вершины, при помощи других функций, реализуемых в других вершинах («деление»).
- Разложение 0-уровня: замкнутая подсхема (замкнутый конус) заменяется новой вершиной (подсхема реализует конъюнкцию с точностью до отрицания входов и выходов).
- Разложение 1-уровня: вершина заменяется новой вершиной ребра которой идут в дву существующие вершины. Пример:
  - Можно заменить вершину $g = a(b+c+d)$ новой вершиной $f' = n + m = a(b+c) + (a\ d) = a(b+c+d)$, учитывая, что в схеме существуют вершины $n = a\ (b+c)$ и $m = a\ d$.

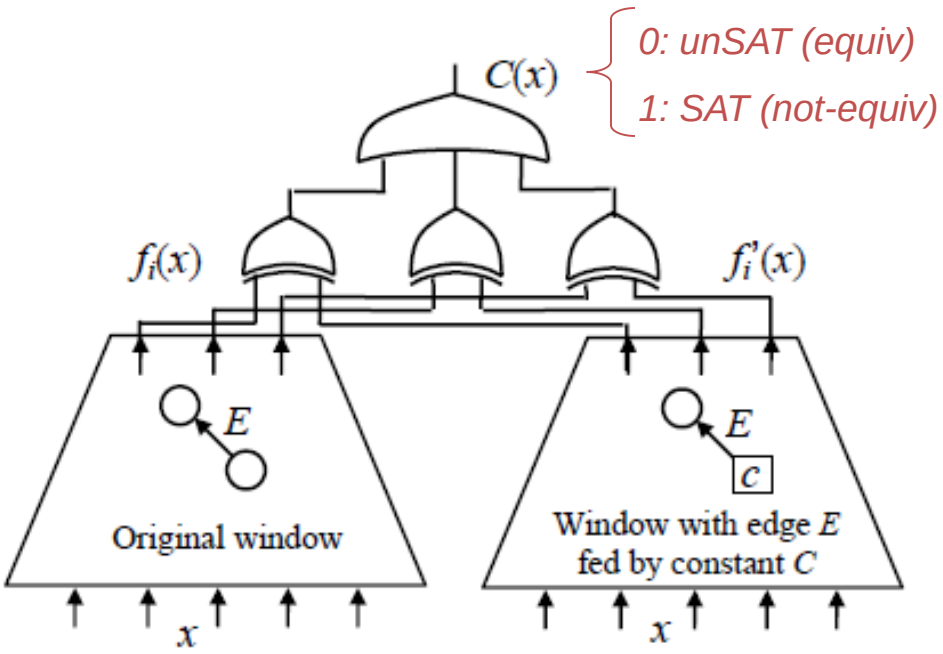


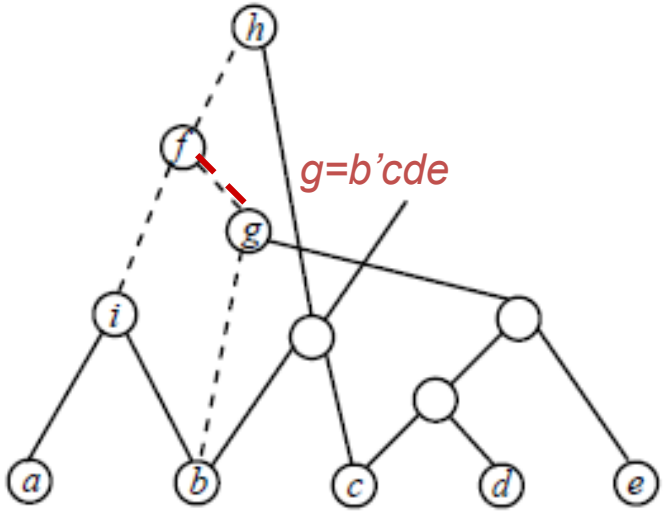
Сложность AIG уменьшается на 1 (можно исключить вершину p)

Исключение избыточных ребер

- Избыточным считается ребро, исключение которого не меняет функционирования схемы
- Симуляция схемы на случайных наборах позволяет определить неэквивалентные вершины и неизбыточные ребра
- ВЫП используется для доказательства или опровержения избыточности ребра
- Ребро $g \rightarrow f$ является избыточным (его можно удалить - $g=0$)

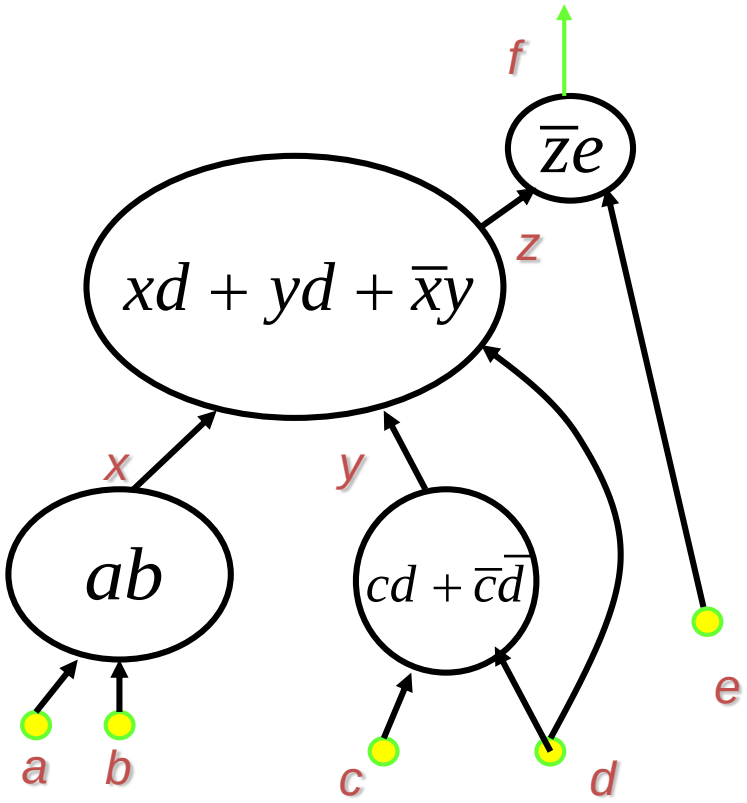
  $h = f'bc = (ab + b'cde)bc = abc$





Сравнение КЛС и AIG

КЛС



AIG