БДД

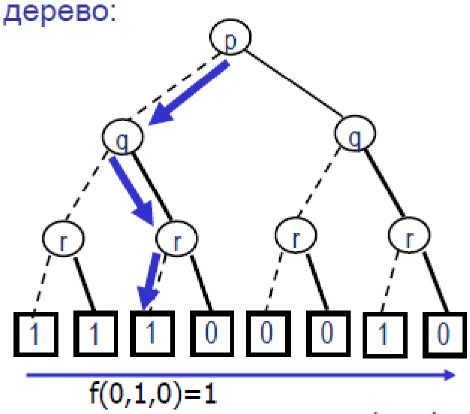
Представления булевых функций

1. Таблица истинности:

pqr	f
000	1
001	1
010	1
011	0
100	0
101	0
110	1
111	0

=

2.Семантическое дерево:



- Произвольная формула, ДНФ и КНФ не являются каноническими представлениями
- ТИ, семантическое дерево, СДНФ и СКНФ, полином Жегалкина являются каноническими представлениями

Но все они громоздки!!

Сложность $O(2^n)$

3. Логическая формула $f(p,q,r) = \neg p \vee q \oplus rq(p \vee r)$
4. СДНФ: $f(p,q,r) = \neg p\, \neg q\, r \vee \neg p\, \neg q\, \neg r \vee q\, p\, \neg r \vee q\, \neg p\, \neg r$
5. ДНФ: $f(p,q,r) = \neg p\, \neg q \vee q\, \neg r$
6. КНФ: $f(p,q,r) = (\neg p \vee q)\, (\neg q \vee \neg r)$,
7. СКНФ: $f = (\neg p \vee q \vee r)\, (p \vee q \vee \neg r)\, (\neg q \vee \neg r \vee p)\, (\neg q \vee \neg r \vee \neg p)$
8. Полином Жегалкина: $f(p,q,r) = 1 \oplus p \oplus pq \oplus qr$

Немасштабируемость решения задач с БФ

Масштабируемость = scalability

БФ, представленные стандартными способами, не позволяют эффективно решать проблемы вследствие экспоненциального роста представления БФ

Это - отражение главного положения теории сложности вычислений:

ЕСЛИ время, требуемое для решения конкретных экземпляров задачи, растет экспоненциально с увеличением размеров этих экземпляров,

ТО эта задача является практически неразрешимой,

ЧТО ДЕЛАТЬ?

Пример.
ТИ функции от 20 переменных имеет 2^{20} ~ МИЛЛИОН строк, нужно оперировать с мегабайтами информации

50 двоичных переменных: ТИ с числом строк $2^{50} = 10^{17}$ строк. Если перебирать по миллиону строк в секунду в такой ТИ, для поиска нужно больше 3 миллиардов лет

BDD – новая форма представления булевых функций

В 1986 г. Randell Briant предложил новую очень эффективную форму представления БФ, которая называется Binary Decision Diagram (BDD)

BDD – это представление функции в виде направленного графа - решающей диаграммы, в которой нет избыточностей

R.E.Bryant. Graph-based algorithms for boolean functions manipulation. *IEEE Transactios on Computers, 8 (C35), 1986* – *самая цитируемая работа в Computer Science*

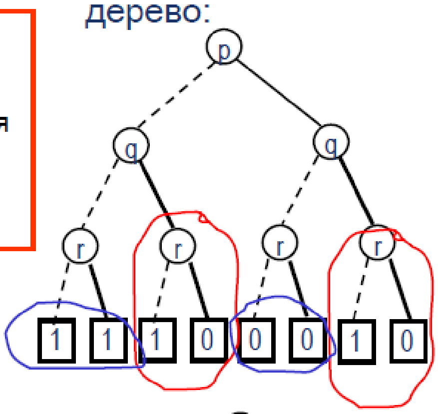
BDD имеют множество хороших свойств, с их помощью в настоящее время решаются многие задачи

Представление в BDD используемых на практике двоичных функций растет полиномиально, или даже линейно от числа двоичных переменных, хотя существуют функции, сложность представления которых экспоненциальна

Наиболее экономное представление в BDD там, где есть симметрия (например, схемы и аппаратные системы)

BDD – Бинарные решающие диаграммы (Binary Decision Diagrams)

Семантическое дерево:

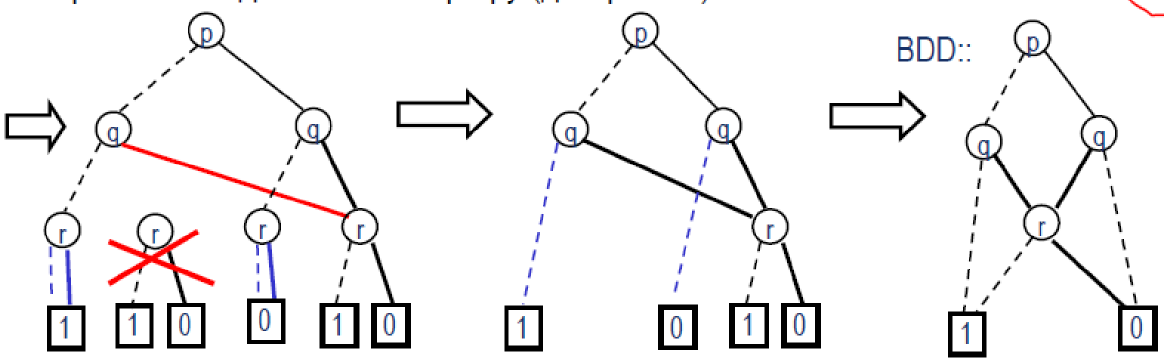


BDD-это семантическое дерево без избыточностей

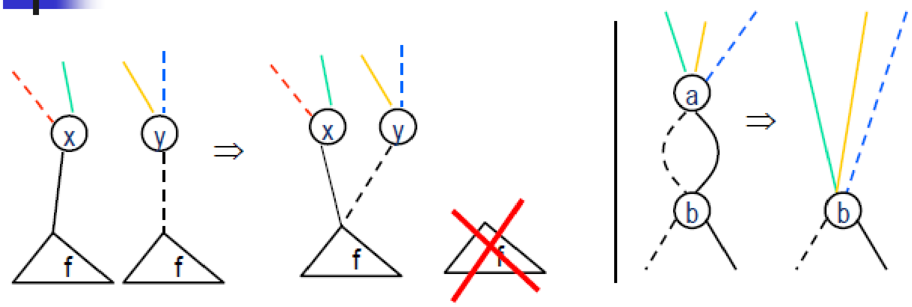
BDD – ациклический орграф, в котором отсутствуют повторения в структуре, с одной корневой вершиной, двумя листьями, помеченными 0 и 1, и промежуточными вершинами. Корневые и промежуточные вершины помечены переменными, из них выходят ровно два ребра

Binary – потому что двоичные переменные

Decision – потому что решения принимаются при направленном движении по графу (диаграмме)



Алгоритм Reduce: преобразование семантического дерева в BDD



C1: Если в графе имеются одинаковые подструктуры, то оставляем только одну из них

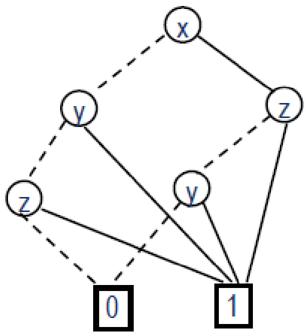
C2: Если обе выходные дуги вершины v ведут в одну вершину, то вершина v выбрасывается

> Повторное применение C1, C2 в любом порядке к семантическому дереву функции f или к любому полученному из него графу, приводит к минимальному представлению f, которое называется BDD

Вопрос: почему алгоритм Reduce завершается??

BDD, OBDD и ROBDD

Не любая BDD является каноническим представлением, нужно еще ограничение на порядок переменных



BDD, но не OBDD

Ordered BDD - это BDD, в которой переменные встречаются в фиксированном порядке не более одного раза

Reduced Ordered BDD - это редуцированная OBDD. Именно ROBDD обладает всеми хорошими свойствами

> Чаще всего ROBDD называют просто BDD

Порядок: x < y < z   по одной ветви
x < z < y   по другой

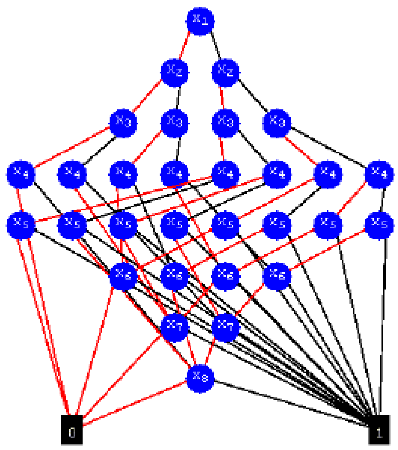
R.E.Bryant. Symbolic function manipulation with ordered binary-decision diagrams. *ACM Computing Surveys, 24(3), 1992*

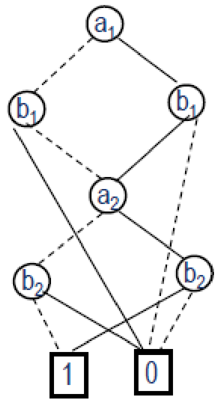
Примеры ROBDD

Функция от 8 переменных. ТИ – 2^8 = 256 строк

$f=x_1x_3x_5\neg x_7 \vee x_2 \neg x_4x_5 \vee x_2 \neg x_3 \neg x_6x_7 \vee x_5x_6x_7 \neg x_8$

$f = (a_1 \equiv b_1)(a_2 \equiv b_2)$

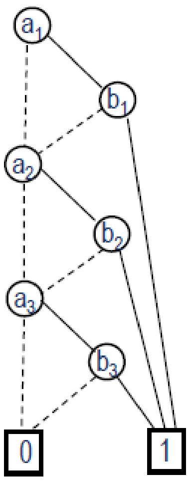




$a1<b1<a2<b2$

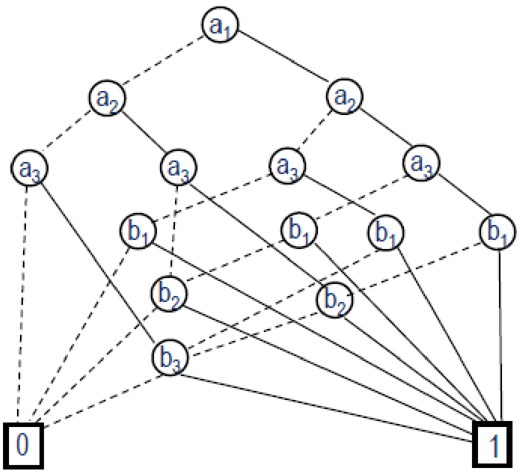
Сложность BDD зависит от порядка переменных

$f = a_1b_1 \vee a_2b_2 \vee a_3 b_3$



$a_1<b_1<a_2<b_2<a_3<b_3$

Рост линеен



$a_1<a_2<a_3<b_1<b_2<b_3$

Рост экспоненциален

Какова сложность BDD при оптимальной упорядоченности?

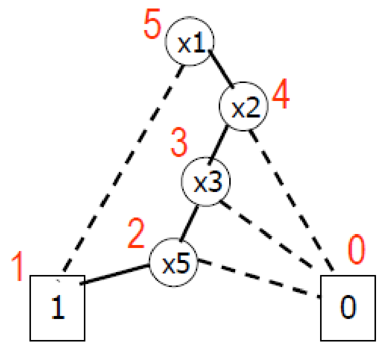
Задача проверки оптимальности порядка BDD является NP трудной

Для почти всех встречающихся на практике булевых функций сложность BDD линейна, либо существуют простые эвристики для установления “хорошего” упорядочения. Существуют алгоритмы динамического переупорядочивания

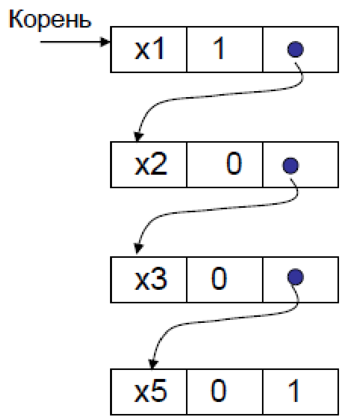
R.Rudell. Dynamic variable ordering for ordered binary decision diagrams. // Int. Conf. on Computer-Aided Design, IEEE, 1993

Представление BDD

$f = \neg x_1 \vee x_2x_3x_5$



n	var	low	high
0			
1			
2	x5	0	1
3	x3	0	2
4	x2	0	3
5	x1	1	4



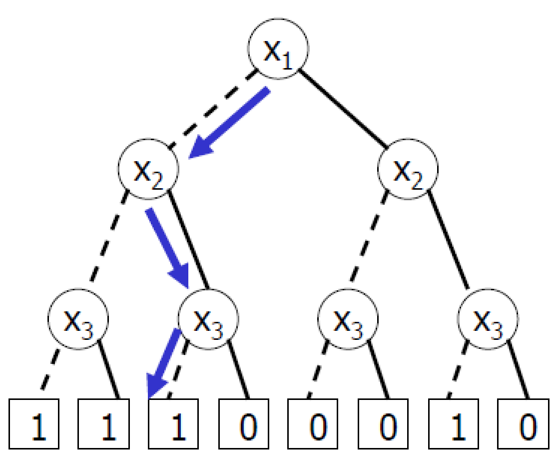
- BDD представляется таблицей, в которой указываются номер вершины, номер переменной, и куда направлены два выхода, 0 и 1
- Вместо таблицы можно использовать связный список или формулу для вершины v: $[x \to p0, p1]$, где x – переменная, помечающая вершину v, а p0 и p1 – указатели на подграфы, в которые из вершины v идут ребра, помеченные, соответственно, 0 и 1.
- Сложность представления БФ в BDD пропорциональна числу вершин

Как вычислять значение двоичной функции по бинарному решающему дереву

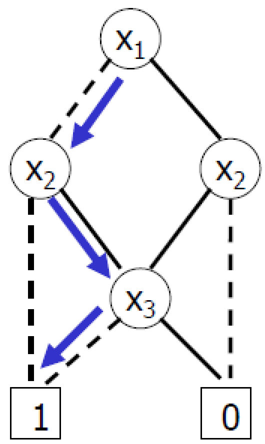
Вычисление f на наборе <0,1,0>:
f(0,1,0) = 1

$x_1x_2x_3$	$f(x_1,x_2,x_3)$
000	1
001	1
010	1
011	0
100	0
101	0
110	1
111	0

ТИ



Binary Decision Tree



Binary Decision Diagram

Значение функции вычисляется простым движением по ветвям от корня к листу

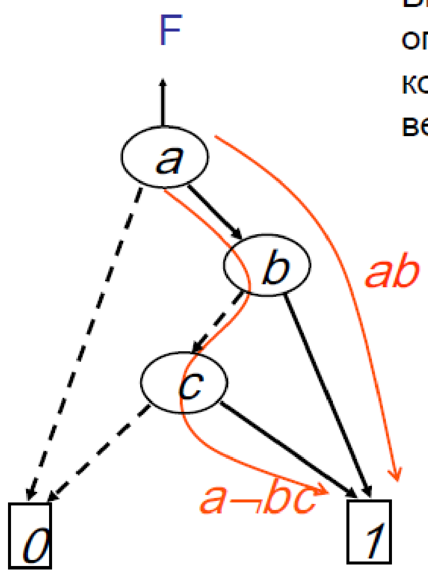
Вычисление значений функции по BDD

$F(a,b,c) = ab \vee a\neg bc$

a b c	f(a,b,c)
000	0
001	0
010	0
011	0
100	0
101	1
110	1
111	1

$f2 = a\neg bc$

$f1 = ab$

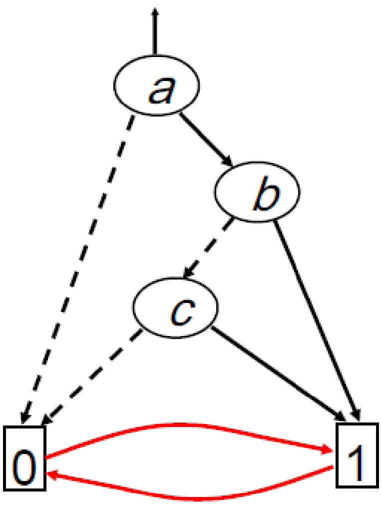


Вид функции в ДНФ определяют те пути, которые из корневой вершины идут в 1

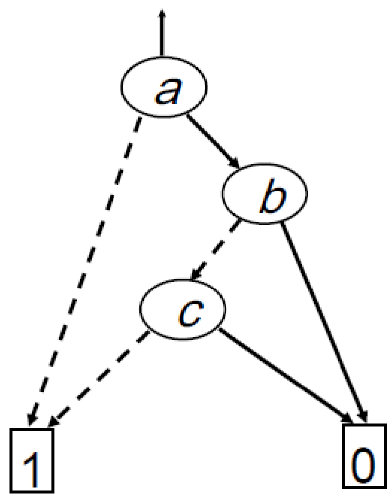
$F = ab \vee a\neg bc$

Булевы операции над BDD: отрицание

$F1 = ab \vee a\neg bc$

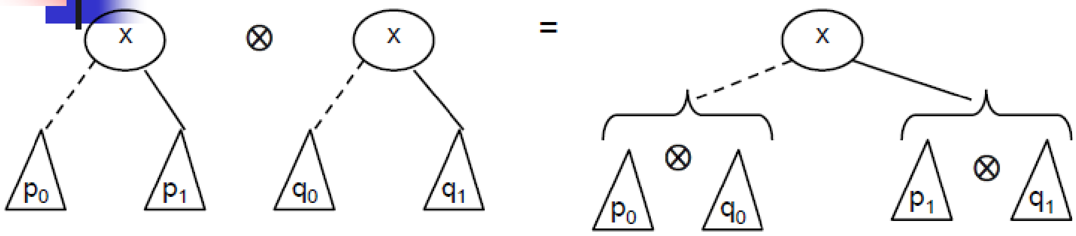


$F2 = \neg\,(ab \vee a\neg bc)$

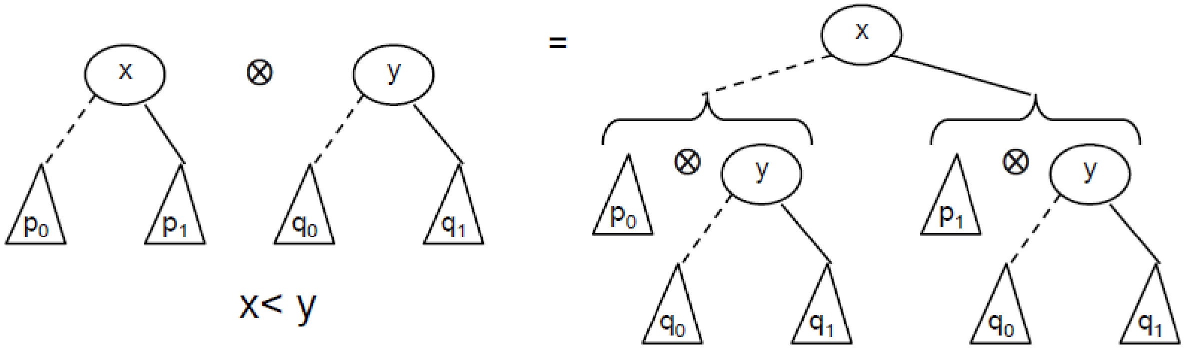


- BDD для отрицания функции строится изменением значения 0 на 1 и 1 на 0 в финальных вершинах

Булевы операции над BDD: алгоритм Apply



$$[x \to p_0, p_1] \otimes [x \to q_0, q_1] = [x \to p_0 \otimes q_0 ,\ p_1 \otimes q_1]$$



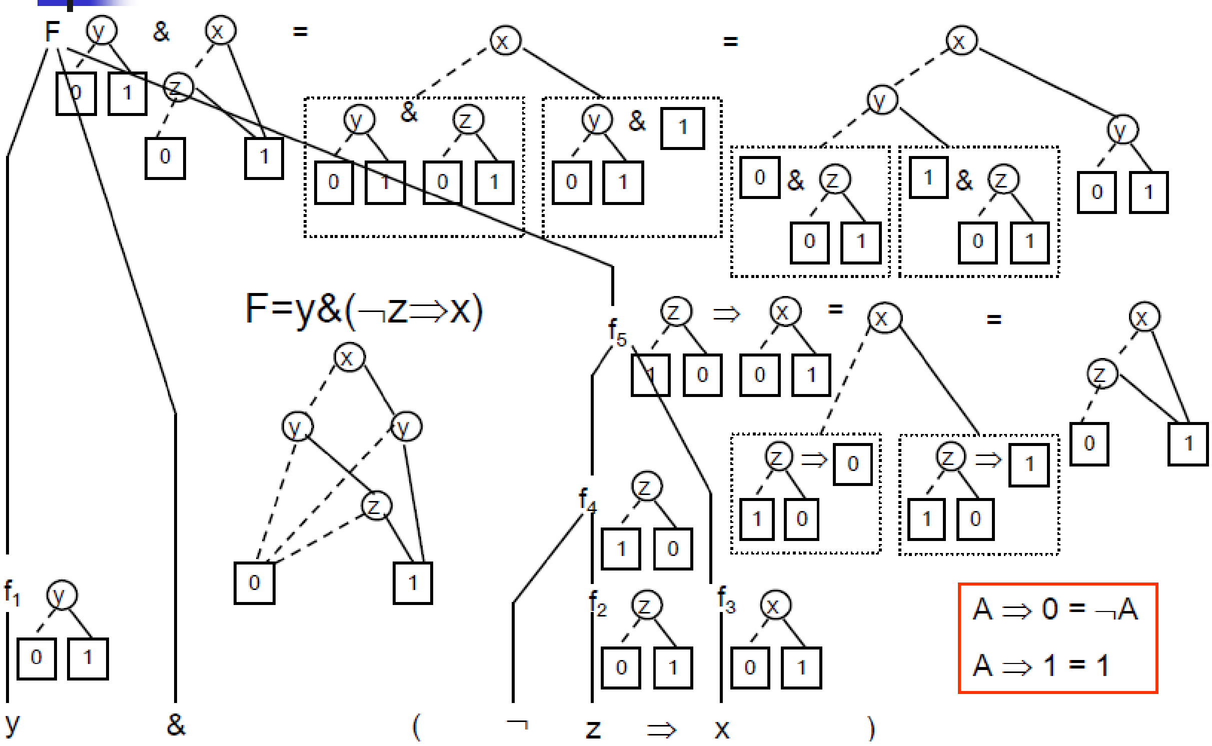
$$[x \to p_0, p_1] \otimes [y \to q_0, q_1] = [x \to p_0 \otimes [y \to q_0, q_1],\ p_1 \otimes [y \to q_0, q_1]\ ]$$

Реализация булевых операции над BDD имеет линейную сложность

Как построить BDD по булевой функции

$F=y\&(\neg z \Rightarrow x)$

$x<y<z$



Свойства BDD

1. BDD – каноническое представление – ***любая** логическая функция имеет единственное представление, минимальное **в базисе {0, 1, ite} при данном порядке переменных*** (этим не обладают КНФ и ДНФ!!!)

2. Сложность зависит от порядка переменных. Разный порядок переменных дает разные представления

Пример: функция $(a_1 \oplus b_1) \& \ldots \& (a_n \oplus b_n)$

при порядке $a1 \ldots a_n\ b_1 \ldots b_n$ имеет сложность $3 \times 2^n - 1$ - экспоненциальная

при порядке $a_1 b_1 a_2 b_2 \ldots a_n b_n$ имеет сложность $3 \times n + 2$ - линейная

3. Проблема нахождения оптимального порядка NP – трудна. Но есть эвристики

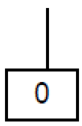
4. BDD для почти всех функций имеет линейную сложность.
Некоторые классы функций имеют экспоненциальную сложность BDD *при любом* порядке переменных (пример: функция, выдающая средний бит результата произведения A × B n-разрядных переменных A и B)

5. Булевы операции над БФ, представленными в BDD, просты.
Теорема. Сложность выполнения булевых операций над двумя функциями f и g, представленными в BDD, полиномиальна: O(|f| x |g|)

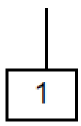
Свойства BDD (2)

- Выполнимость (Теорема Кука: "*Проблема выполнимости булевой формулы NP-полна*")
  - Для BDD проверка выполнимости, невыполнимости и общезначимости тривиальны – не является ли BDD вырожденной (значения 0 или 1)
  - Но проблема построения представления функции в BDD является NP-полной

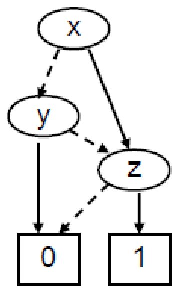
Вырожденные BDD:



Невыполнима



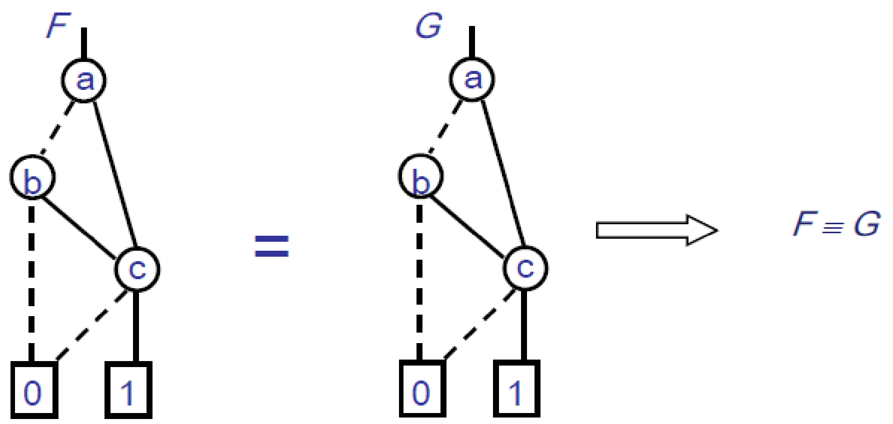
Общезначима



Выполнима

Проверка равнозначности булевых функций

$F = a\neg bc \vee abc \vee ab\neg c$ $G = ac \vee bc$ $\boxed{F \equiv G\ (?)}$



Поскольку BDD – каноническое представление, проверка равнозначности двух функций, представленных в BDD, сводится к проверке совпадения направленных графов

Библиотеки

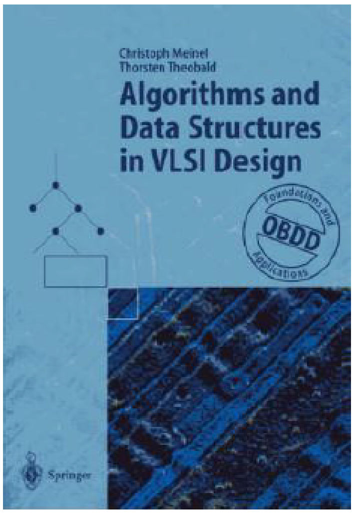
ABCD: The ABCD package by Armin Biere
http://fmv.jku.at/abcd/
BuDDy: A BDD Package by Jørn Lind-Nielsen.
http://sourceforge.net/projects/buddy/
CMU BDD, BDD package, Carnegie Mellon University, Pittsburgh
http://www.cs.cmu.edu/~modelcheck/bdd.html
CUDD: BDD package, University of Colorado, Boulder
http://vlsi.colorado.edu/~fabio/CUDD/
JavaBDD, a Java port of BuDDy that also interfaces to CUDD, CAL, and JDD
http://javabdd.sourceforge.net/
The Berkeley CAL package which does breadth-first manipulation
http://embedded.eecs.berkeley.edu/Research/cal_bdd/
TUD BDD: A BDD Package by Stefan Höreth
http://www.rs.e-technik.tu-darmstadt.de/~sth/
Vahidi's JDD, a java library that supports common BDD and ZDD
http://javaddlib.sourceforge.net/jdd/

Применения BDD

- Применений оказалось огромное количество – везде, где, в принципе, возможно применение двоичных функций, использование BDD как формы представления двоичных функций качественно повышает эффективность решений

- "Прямые" применения
  - Минимизация двоичных функций
  - Операции над БФ от большого числа переменных
  - Синтез логических схем по их представлению в форме BDD
  - Верификация и проверка эквивалентности логических схем
- Борьба с "проклятием размерности"
  - Упрощение любых алгоритмов, манипулирующих конечными структурами данных большого объема
    - Компрессия изображений, Поиск в БД, Алгоритмы на графах
      ...
  - Верификация реагирующих (reactive) систем
    - С использованием BDD стало возможным увеличить сложность верифицируемых систем на многие порядки!

BDD и синтез СБИС

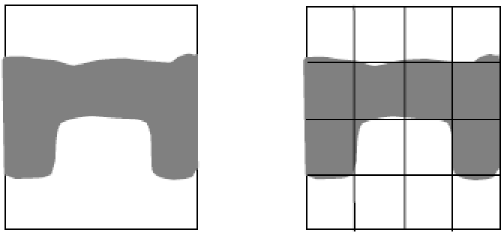


- Подзаголовок книги:
  OBDD – Foundations and Applications

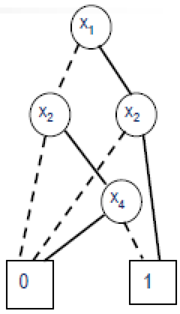
Из введения:
Для сверхбольших схем (СБИС) проблема представления и преобразования их функционального поведения становится почти неразрешимой. Проблема была разрешена в результате установления плодотворной связи с одним из базовых результатов теоретической вычислительной науки в области построения структур данных и эффективных алгоритмов их обработки. Этот результат - использование OBDD - привел к драматическому улучшению производительности и прорыву во многих CAD проектах во всем мире

BDD являются основной моделью для представления и обработки информации в современной технологии разработки СБИС

Компрессия изображений



x_1x_2 \ x_3x_4	10	11	01	00
10	0	0	0	0
11	1	1	1	1
01	1	0	0	1
00	0	0	0	0



а) б) в) г)

f = x1x2 ∨ x2¬x4

а) двумерное черно-белое изображение

б) разбиение изображения на множество пикселей

в) соответствующая двоичная функция $x_1x_2 \vee x_2\neg x_4$

г) BDD, представляющая изображение

Для представления мегабайтного изображения нужна BDD функции от 23 переменных (поскольку $\lceil \log (8*10^6) \rceil = 23$)

Во многих случаях такое представление экономнее стандартных, алгоритмов сжатия, например JPEG

Характеристические булевы функции

Обозначим χ_A характеристическую булеву функцию множества A. Она равна 1 на наборах, кодирующих элементы из A, т.е. на { 000, 001, 010, 110}

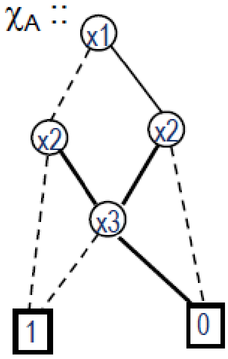
	f
000	1
001	1
010	1
011	0
100	0
101	0
110	1
111	0

Пусть x_1,x_2,x_3 – разряды кодировки.

Тогда:

$\chi_A = \neg x_1\neg x_2\neg x_3 \vee \neg x_1\neg x_2 x_3 \vee \neg x_1 x_2\neg x_3 \vee x_1 x_2\neg x_3$

задает A = {000, 001, 010, 110}



Итак, подмножества конечного множества можно задавать булевой формулой, представляющей характеристическую функцию, и, следовательно, ее можно представить в форме BDD

Будем писать A = {000, 001, 010, 110} (x_1, x_2, x_3), чтобы показать переменные кодировки и их порядок

BDD в обработке данных

- Конечные структуры данных и алгоритмы работы с ними используются в подавляющем большинстве применений CS, алгоритмов, дискретной аппаратуры. BDD широко используется для эффективного представления конечных структур данных и эффективных алгоритмов, работающих с ними. Три причины использования BDD:
  - компактность представления данных
  - эффективность алгоритмов манипулирования этими данными
  - каноничность представления двоичных функций с помощью BDD
- Можно провести аналогию с использованием позиционной числовой записи вместо унарного кода (в унарной системе счисления количество N представляется N единицами). В обычных алгоритмах проводится перебор элементов конечных множеств по одному. В алгоритмах, основанных на BDD, операции выполняются над МНОЖЕСТВАМИ данных, представленных характеристическими функциями
- Это, фактически, революция во всех областях, связанных с обработкой дискретной информации. В частности:
  - BDD являются основной моделью для представления и обработки информации в области разработки VLSI
  - использование BDD в ИИ дало новую жизнь многим направлениям в ИИ (например, оказалось очень успешным в задачах планирования)
- Разработаны системы эффективного манипулирования отношениями для решения разнообразных задач на основе BDD: *CrocoPat: A Tool for Simple and Efficient Relational Computation,* http://mtc.epfl.ch/~beyer/CrocoPat/ (2005)

Недостатки представления БФ в форме BDD

- Два недостатка:
  - сложность представления в BDD зависит от порядка переменных. Например, при реализации сложения при одном порядке переменных BDD имеют линейную сложность, при другом – экспоненциальную
  - для некоторых функций (например, двоичное умножение) все функции имеют экспоненциальную сложность. Однако, большинство используемых на практике функций представляются эффективно (линейно)

Representation of boolean functions	compact?	test for satisf'ty	validity	boolean operations $\cdot$	$+$	$\bar{\ }$
Prop. formulas	often	hard	hard	easy	easy	easy
Formulas in DNF	sometimes	easy	hard	hard	easy	hard
Formulas in CNF	sometimes	hard	easy	easy	hard	hard
Ordered truth tables	never	hard	hard	hard	hard	hard
Reduced OBDDs	often	easy	easy	medium	medium	easy