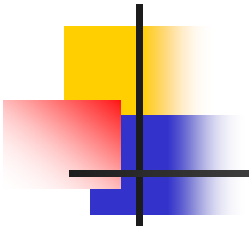
Верификация параллельных программных и аппаратных систем

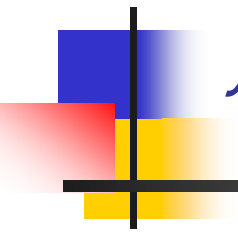
Курс лекций

Карпов Юрий Глебович
профессор, д.т.н., зав.кафедрой
“Распределенные вычисления и компьютерные сети”
Санкт-Петербургского политехнического университета
karpov@dcn.infos.ru



План курса

1. Введение
2. Метод Флойда-Хоара доказательства корректности программ
3. Исчисление взаимодействующих систем (CCS) Р.Милнера
4. Темпоральные логики
5. Алгоритм model checking для проверки формул CTL
6. Автоматный подход к проверке выполнения формул LTL
7. Структура Крипке как модель реагирующих систем
8. Темпоральные свойства систем
9. Система верификации Spin и язык Promela. Примеры верификации
10. Применения метода верификации model checking
11. BDD и их применение
12. Символьная проверка моделей
13. Количественный анализ дискретных систем при их верификации
14. Верификация систем реального времени (I)
15. Верификация систем реального времени (II)
16. Консультации по курсовой работе



Лекция 6

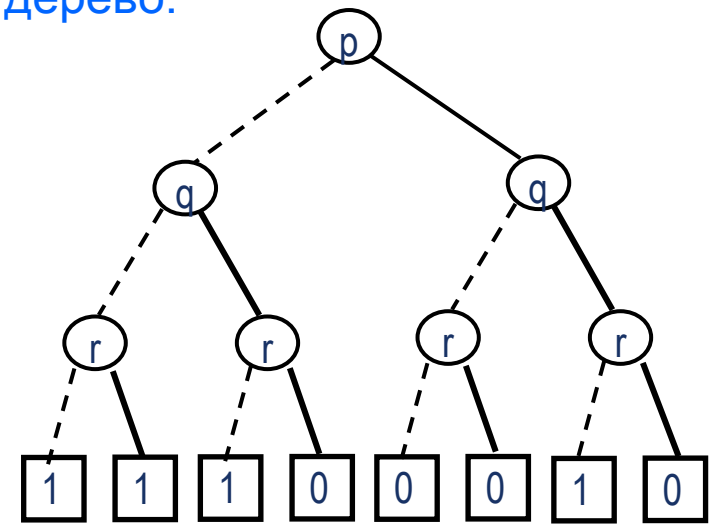
Model checking для формул LTL

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

3. Логическая формула $f(p,q,r) = \neg p \vee q \oplus rq(p \vee r)$

4. СДНФ: $f(p,q,r) = \neg p \neg q r \vee \neg p \neg q \neg r \vee qp \neg r \vee q \neg p \neg r$

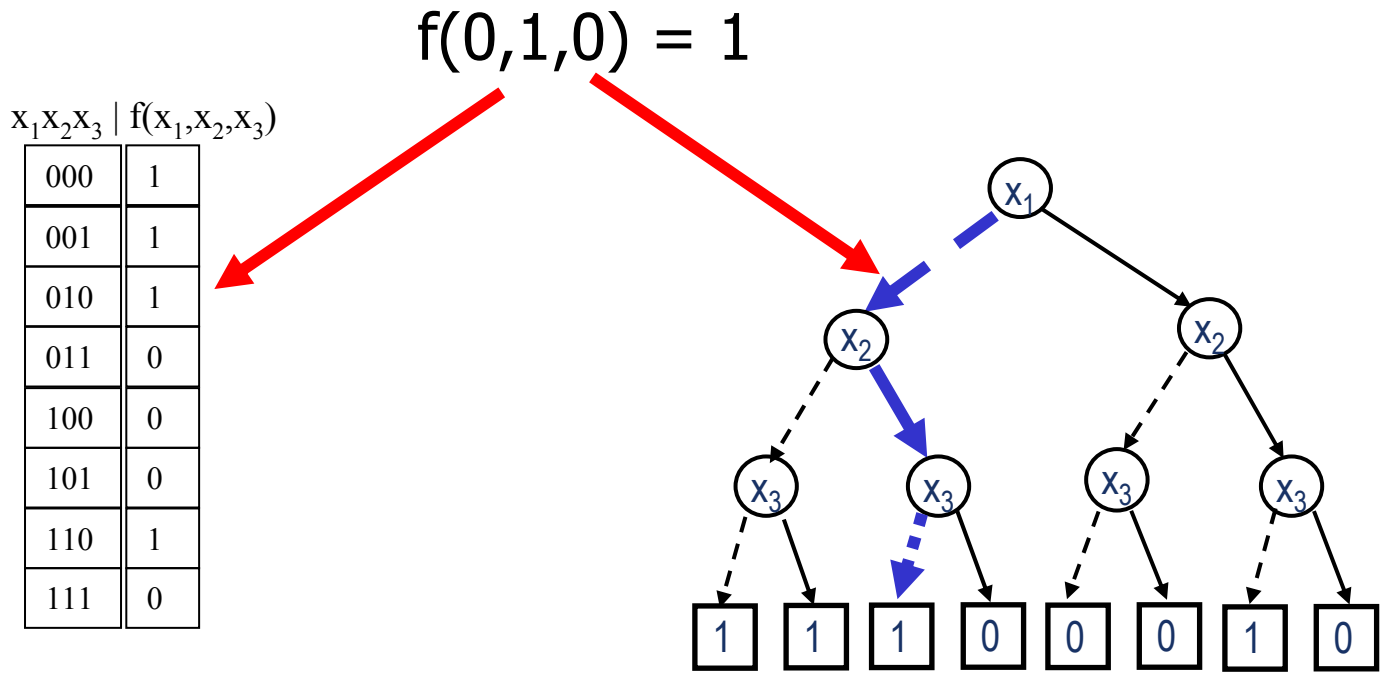
5. ДНФ: $f(p,q,r) = \neg p \neg q \vee q \neg r$

6. КНФ: $f(p,q,r) = (\neg p \vee q)(\neg q \vee \neg r)$

7. Полином Жегалкина: $f(p,q,r) = 1 \oplus p \oplus pq \oplus qr$

Но все они громоздки!!

Как вычислять значение двоичной функции по бинарному решающему дереву



- Значение функции вычисляется простым движением по ветвям от корня к листу

Проблемы, связанные с БФ

Для заданной булевой функции f, представленной в некоторой форме:

- Найти значение f на конкретной интерпретации
- Найти все интерпретации, на которых f = 1
- Проверить, является ли функция f
  - выполнимой (есть наборы, на которых f равна 1)
  - общезначимой (на всех наборах f равна 1)
  - невыполнимой (на всех наборах f равна 0)
- Проверить, совпадают ли две булевых функции

Работать с БФ удобно и легко, если число переменных 2-5!

Как задачи с БФ решаются при большом числе переменных??

Весьма трудно! Вернее, НИКАК

Все проблемы для БФ не решаются при большом числе переменных

Немасштабируемость решения задач с БФ

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

Это - отражение главного положения теории сложности вычислений:

задача является практически неразрешимой, если время, требуемое для решения конкретных экземпляров этой задачи, растет экспоненциально с увеличением размеров этих экземпляров

Пример.

ТИ функции от 20 переменных имеет $2^{20} = 10^6$ строк, нужно оперировать с мегабайтами информации

ТИ функции от 50 переменных имеет $2^{50} = 10^{15}$ строк, тысячи терабайт

BDD – новая форма представления булевых функций

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

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

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

Представление в BDD используемых на практике БФ растет полиномиально, или даже линейно

Наиболее экономно использование BDD там, где есть симметрия (например, аппаратные системы)

▪R.E.Bryant. Graph-based algorithms for boolean functions manipulation. *IEEE Transactios on Computers, 8 (C35), 1986*

BDD – Бинарные решающие диаграммы (Binary Decision Diagrams)

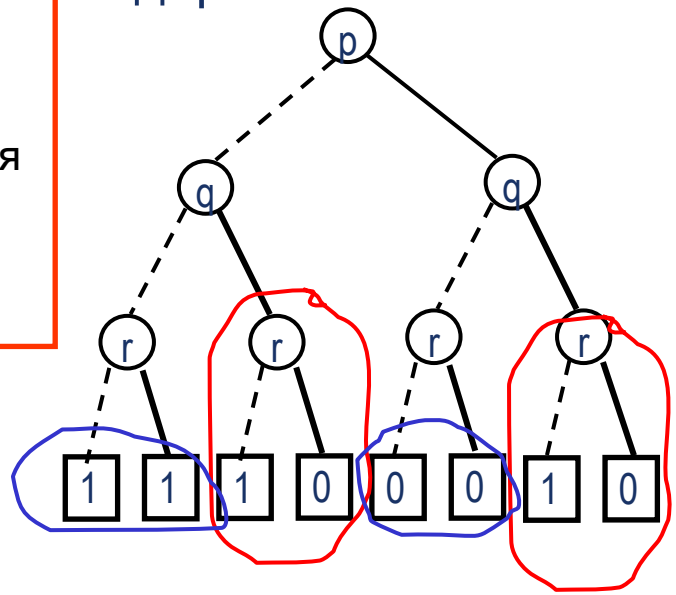
BDD-это семантическое дерево без избыточностей

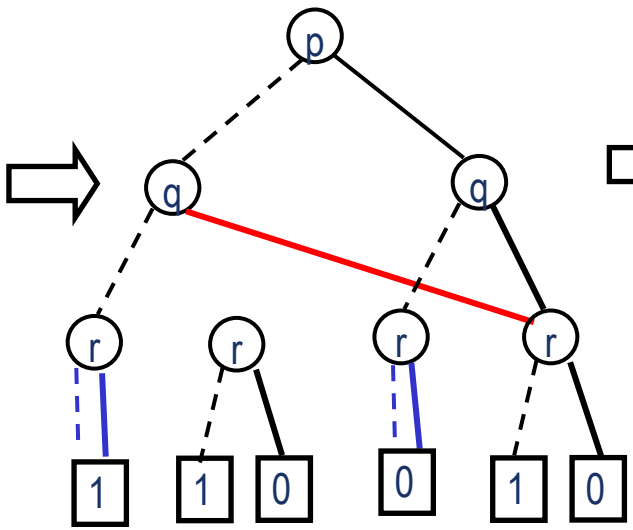
BDD – ациклический орграф, в котором отсутствуют повторения в структуре, с одной корневой вершиной, двумя листьями, помеченными 0 и 1, и промежуточными вершинами. Корневые и промежуточные вершины помечены переменными, из них выходят ровно два ребра

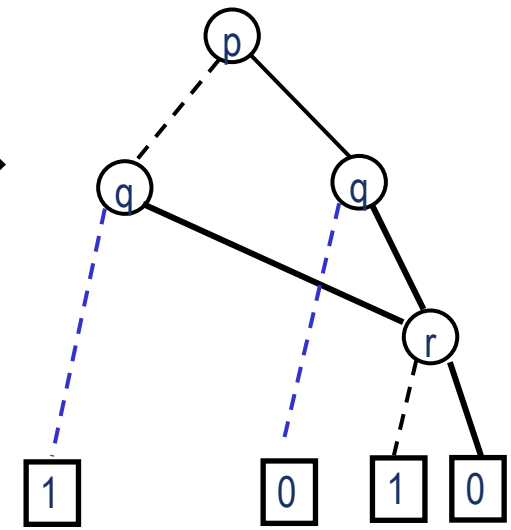
Binary – потому что двоичные переменные

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

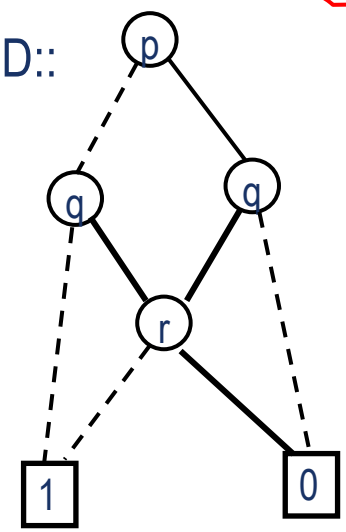
Семантическое дерево:







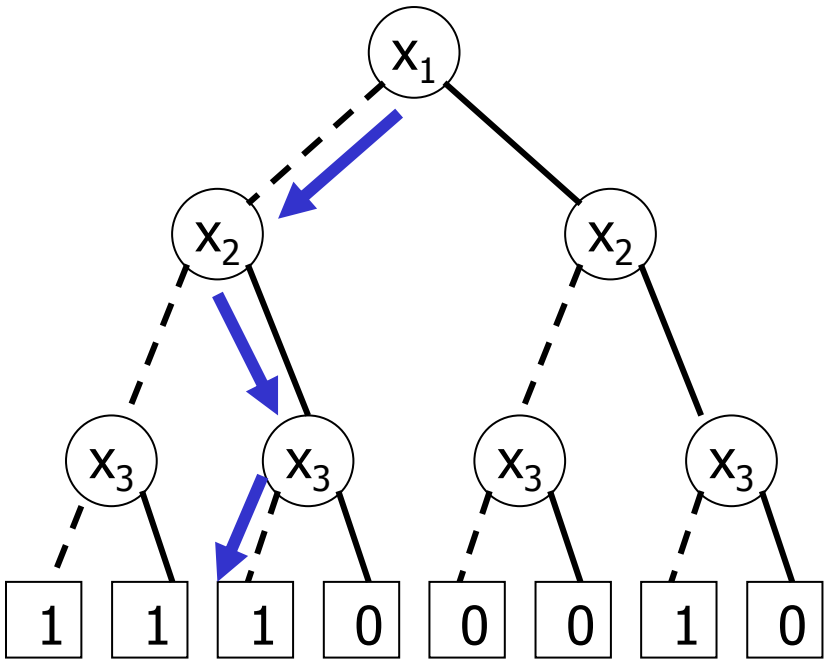
BDD::

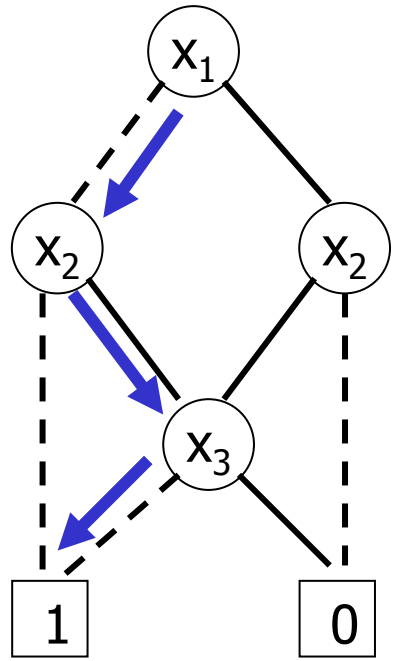


Как вычислять значение двоичной функции по бинарному решающему дереву

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

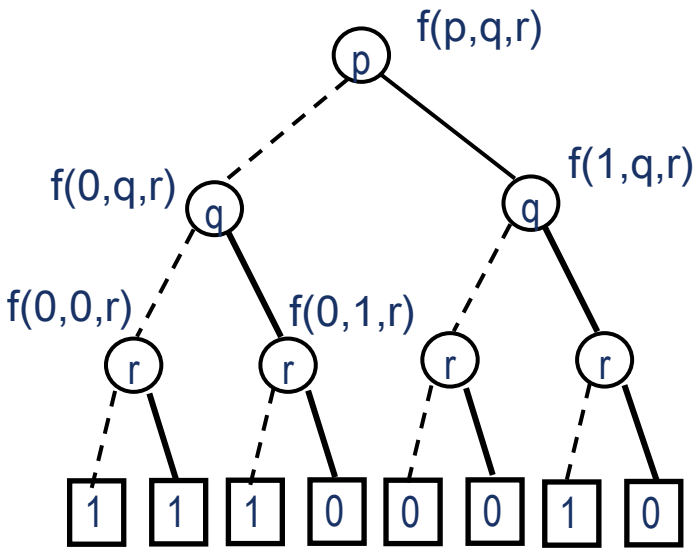
Что такое BDD (формально)

Разложение Шеннона:

$f = \neg x\, f_{x=0} \vee x\, f_{x=1}$

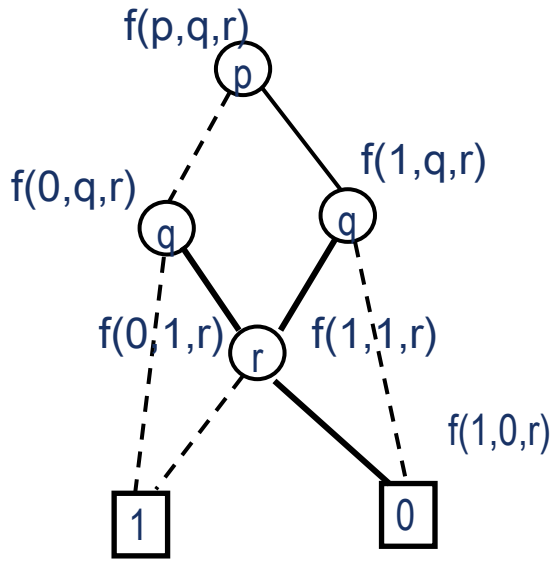
f(p,q,r) = *If* p *then* f(1,q,r) *else* f(0,q,r)

Семантическое дерево::



f(0,0,0) f(0,0,1) ... f(1,1,1)

BDD::

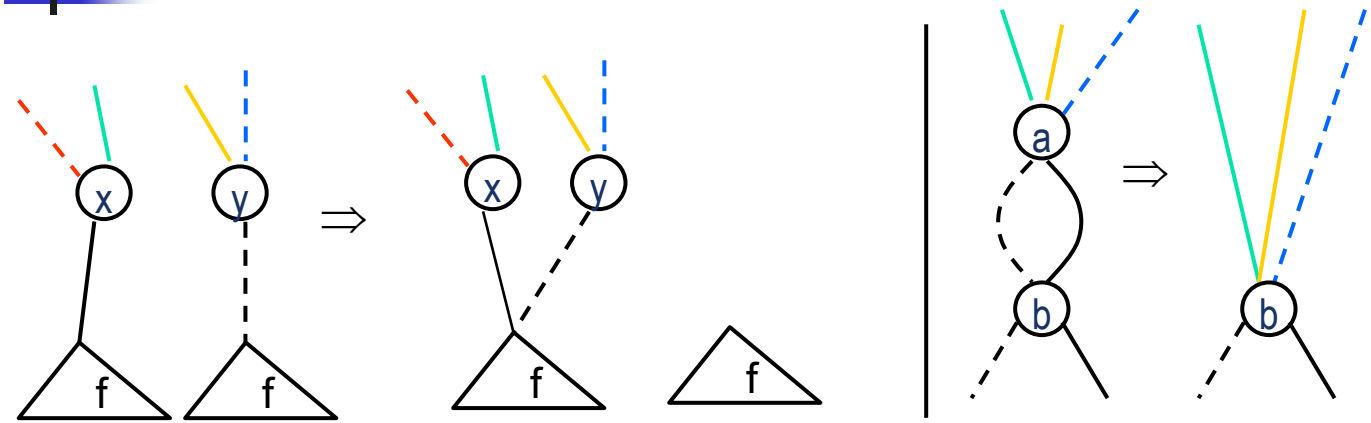


f(0,0,r)= f(0,1,0)= f(1,1,0)

f(1,0,r)= f(1,1,1)= f(0,1,1)

BDD - это минимальное представление f в базисе { ite(p,q.r), 0, 1}

Преобразование семантического дерева в BDD



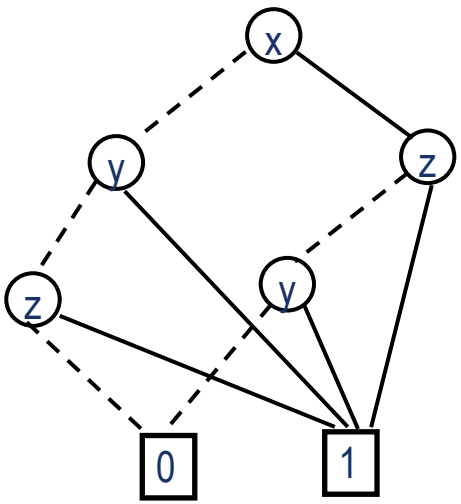
C1: Если в графе имеются одинаковые подструктуры, то остается только одна из них

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

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 не повторяются и встречаются в фиксированном порядке

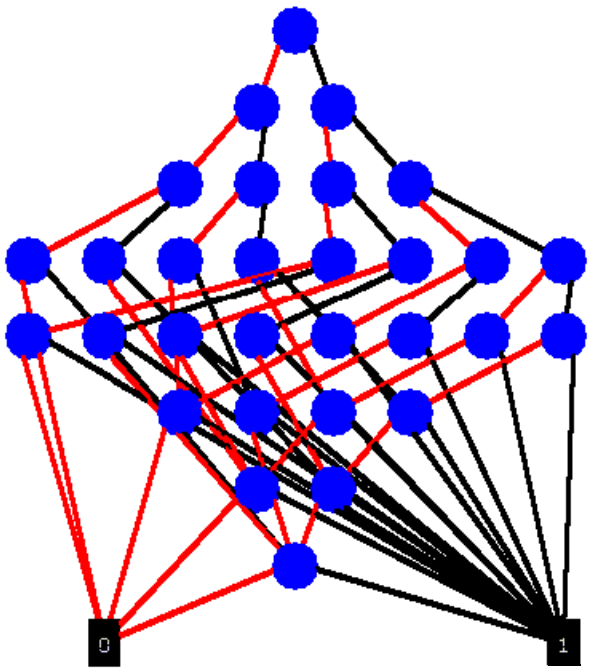
Reduced Ordered BDD - это редуцированная OBDD. Именно ROBDD обладает всеми хорошими свойствами

Чаще всего ROBDD называют просто BDD

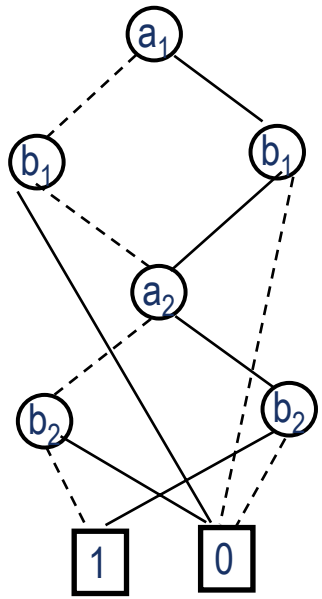
R.E.Bryant. Symbolic function manipulation with ordered binary-decision diagrams. *ACM Computing Surveys, 24(3), 1992*

Примеры ROBDD

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



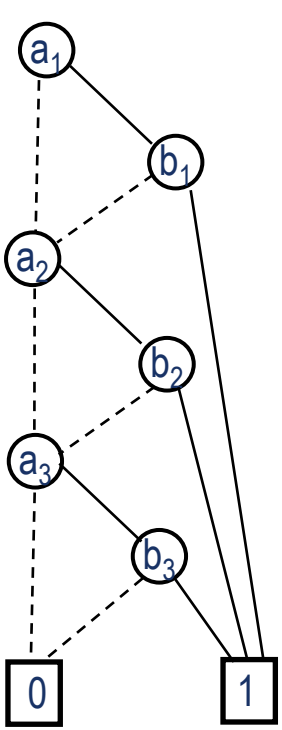
$f = (a_1 \equiv b_1)(a_2 \equiv b_2)$

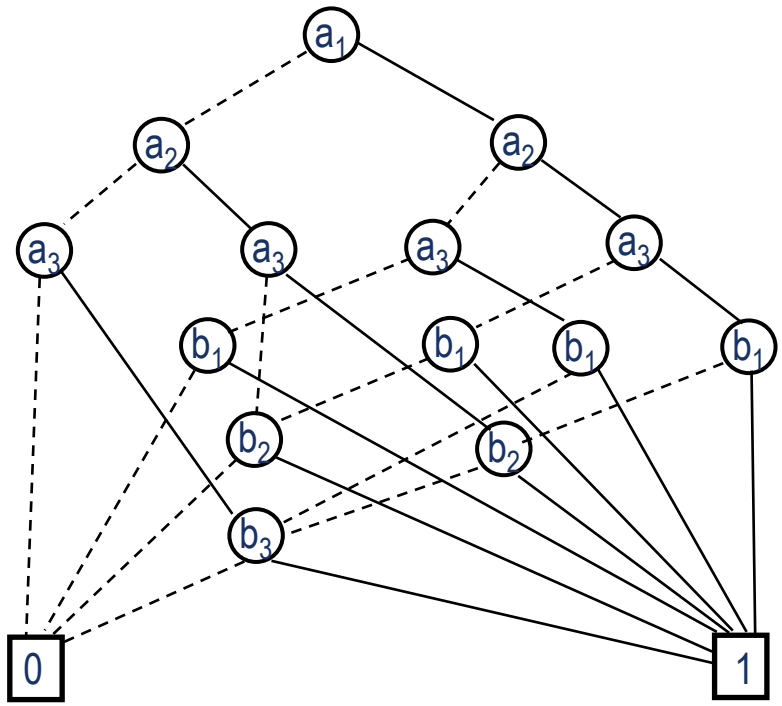


a1<b1<a2<b2

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

Задача проверки оптимальности порядка является NP полной

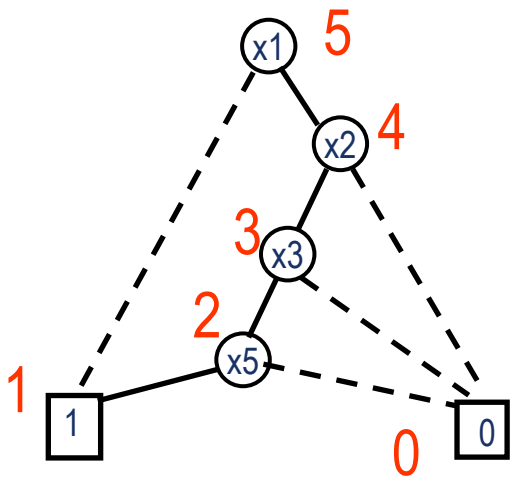
Существуют эвристики для нахождения субоптимального порядка

Какова сложность BDD при оптимальной упорядоченности?

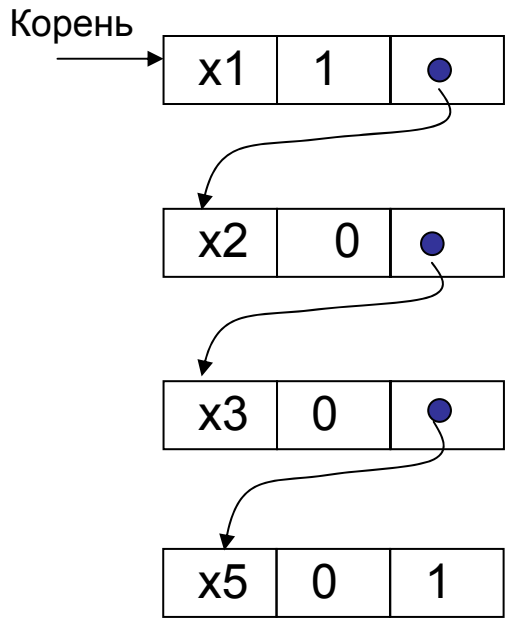
Для подавляющего числа встречающихся на практике функций сложность BDD линейна

Представление BDD

$f = \neg x_1 \vee x_2 x_3 x_5$



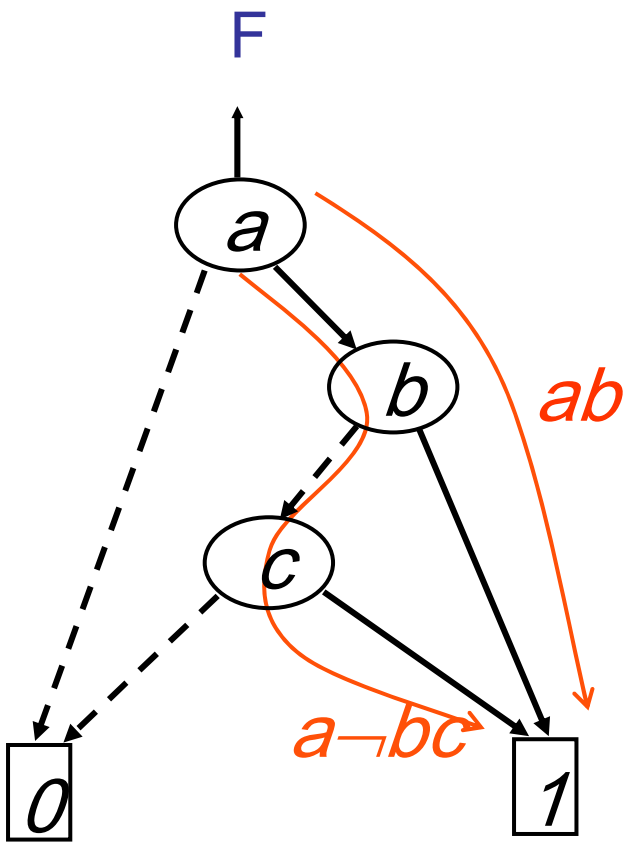
n	var	low	high
0			
1			
2	5	0	1
3	3	0	2
4	2	0	3
5	1	1	4



- BDD представляется таблицей, в которой указываются номер вершины, номер переменной, и куда направлены два выхода, 0 и 1
- Вместо таблицы можно использовать связный список
- Сложность представления БФ в BDD пропорциональна числу вершин

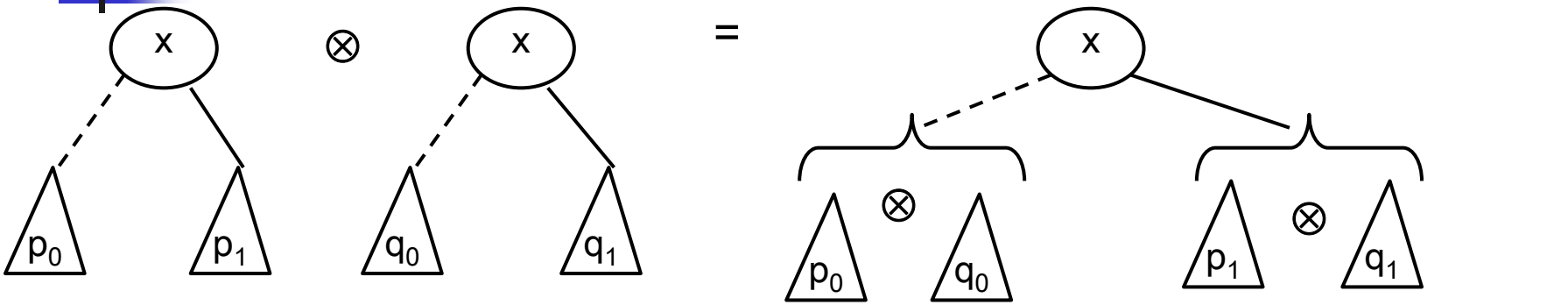
Вычисление значений функции по BDD

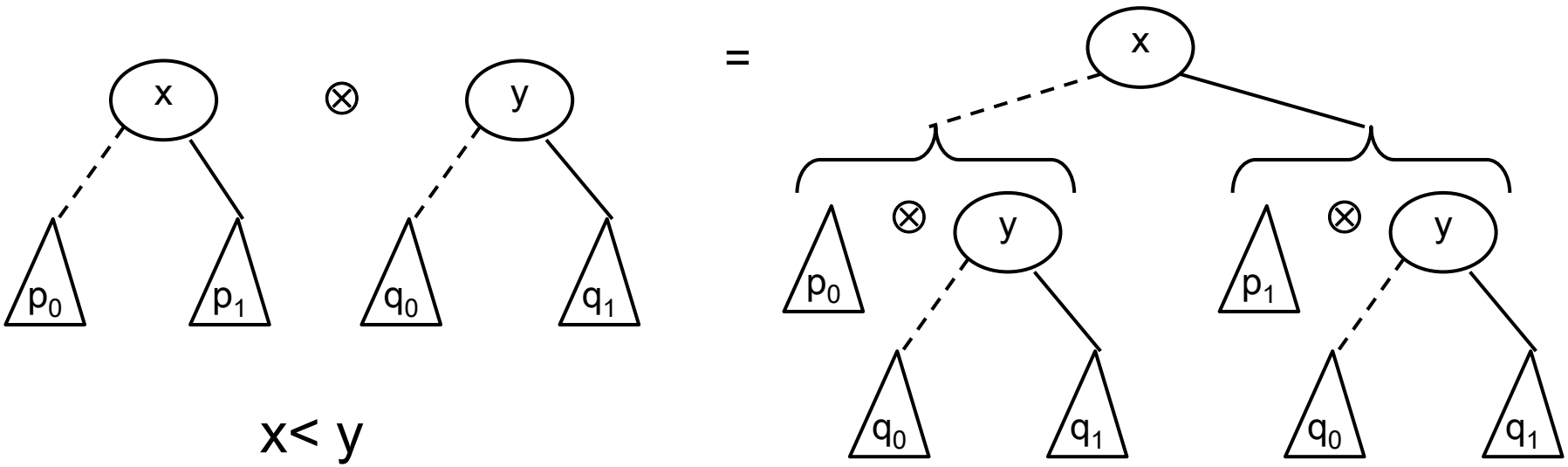
$F(a,b,c) = ab \vee a\neg bc$



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

Булевы операции над BDD: алгоритм Бриана



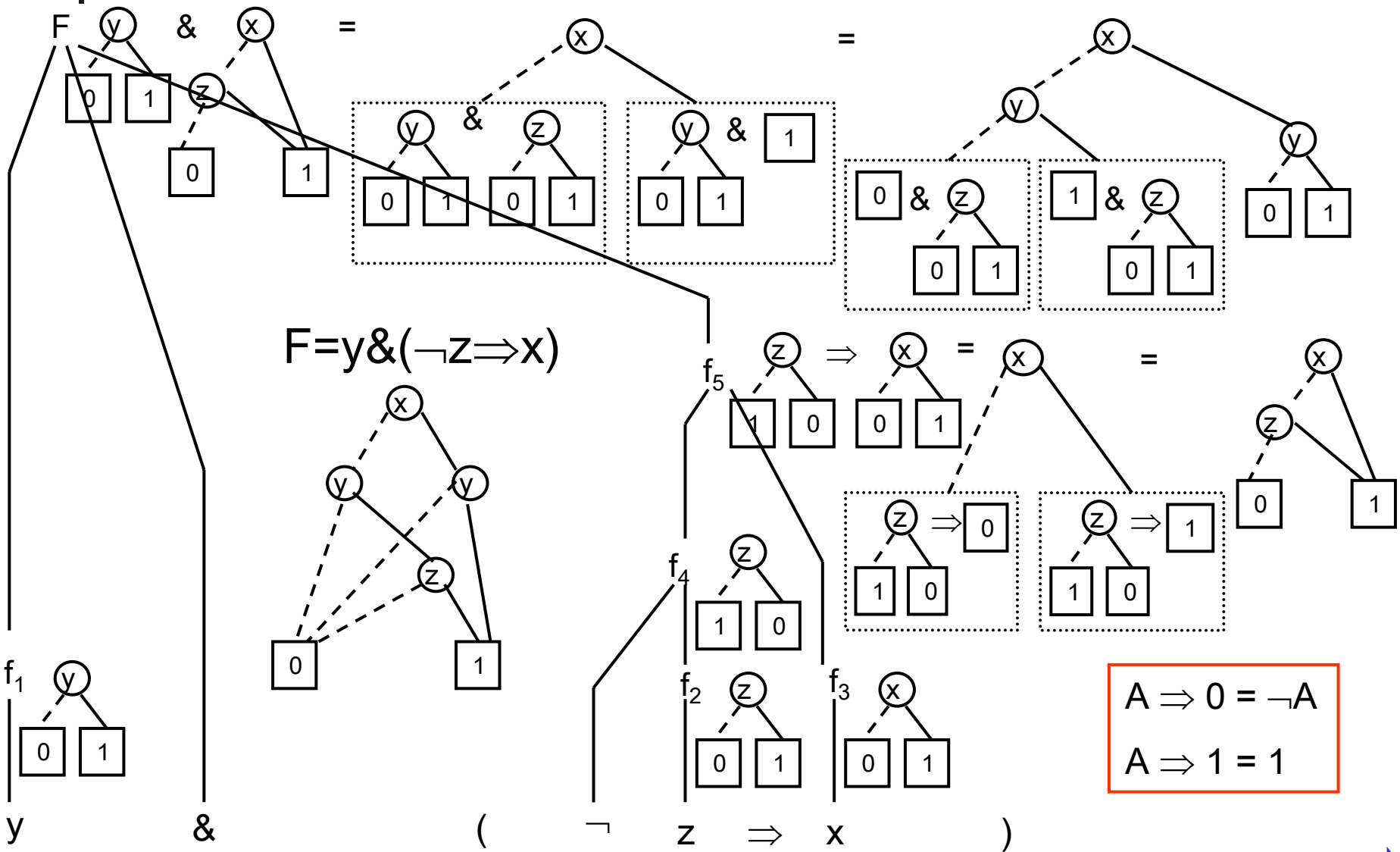


Реализация булевых операции над BDD имеет линейную сложность

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 минимальное представление* (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 $3\times 2^n-1$
при порядке $a_1 b_1 a_2 b_2 \ldots a_n b_n$ имеет сложность $3\times n+2$

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

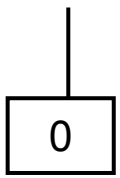
4. BDD для большинства функций имеет линейную сложность.
Некоторые классы функций имеют экспоненциальную сложность BDD *при любом* порядке переменных (пример: функция, выдающая средний бит результата произведения A × B n-разрядных переменных A и B)

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|)

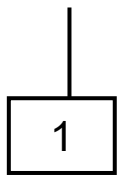
Свойства BDD (2)

- 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 построение представления функции в BDD в худшем случае занимает экспоненциальное время

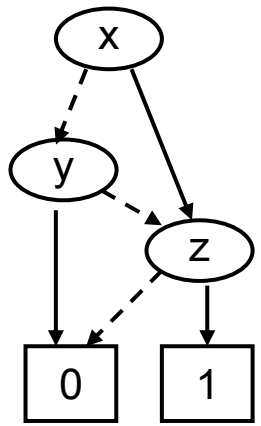
Вырожденные BDD:



Невыполнима



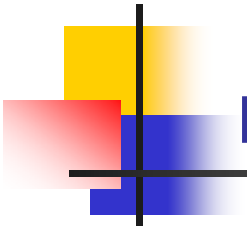
Общезначима



Выполнима

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 and a World-Level package by Stefan Höreth
http://www.rs.e-technik.tu-darmstadt.de/~sth/
Vahidi's JDD , a java library that supports common BDD and ZBDD
operations http://javaddlib.sourceforge.net/jdd/

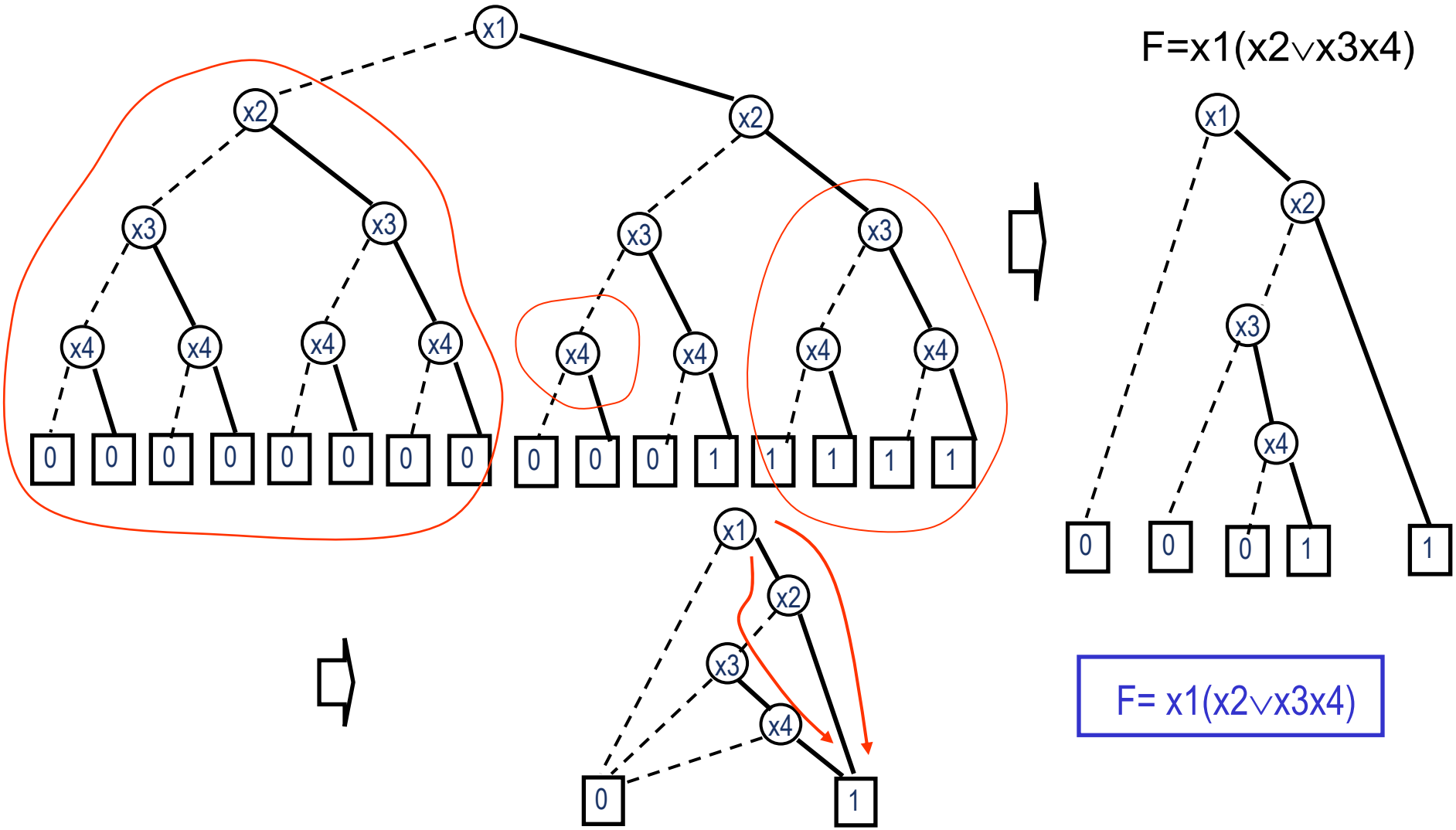


Применения BDD

- Применений оказалось огромное количество
- “Прямые” применения
  - Минимизация логических 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 реактивных (reactive) систем
    - С использованием BDD стало возможным увеличить сложность верифицируемых систем на многие порядки!
- Расширения BDD

Минимизация логических функций

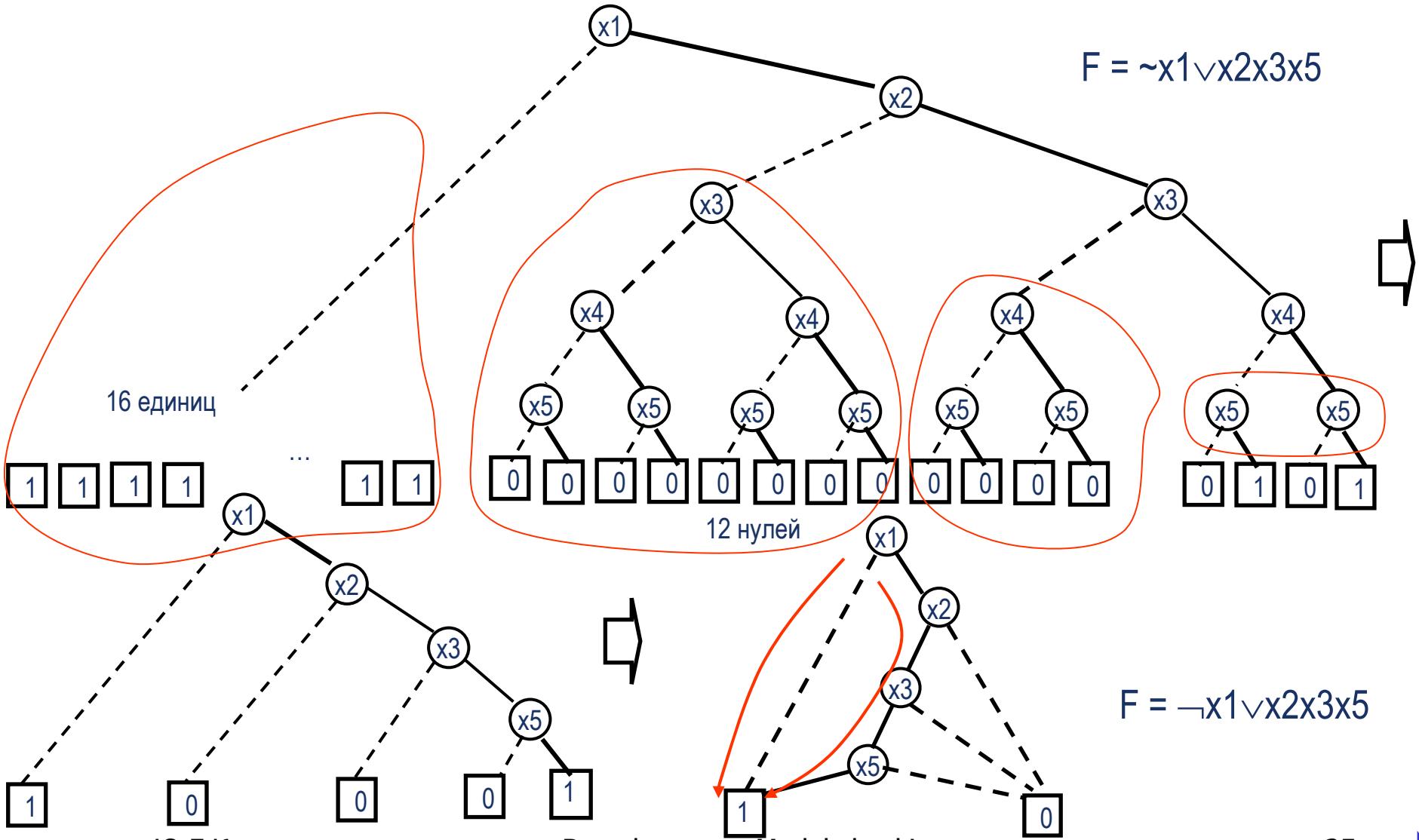
$F = x_1 \sim x_2 x_3 x_4 \vee x_1 x_2 \sim x_3 x_4 \vee x_1 x_2 \sim x_3 \sim x_4 \vee x_1 x_2 x_3 \sim x_4 \vee \sim(\sim x_1 x_2 \sim x_3 x_4)$



$F = x_1(x_2 \vee x_3 x_4)$

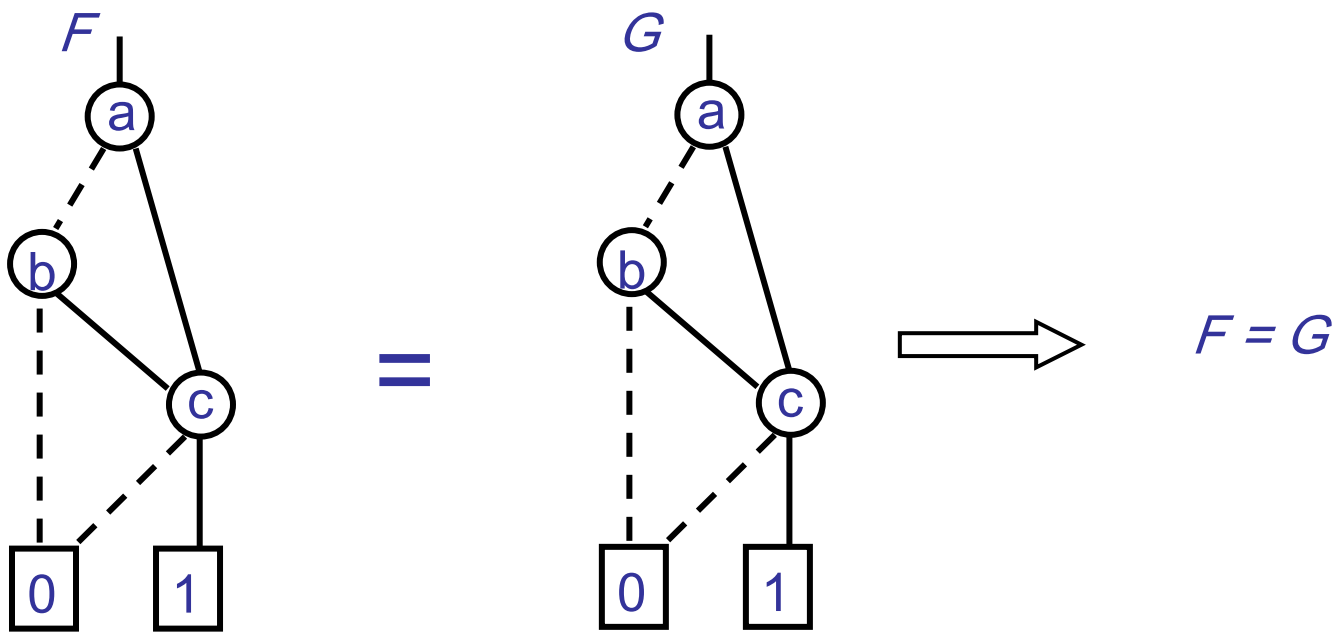
Минимизация логических функций (2)

F =¬x1¬x2¬x3¬x4¬x5∨¬x1¬x2¬x3¬x4¬x5∨¬x1¬x2 ∨ x1x2x3¬x4x5 ∨ x1x2x3x4x5



Проверка эквивалентности булевых функций

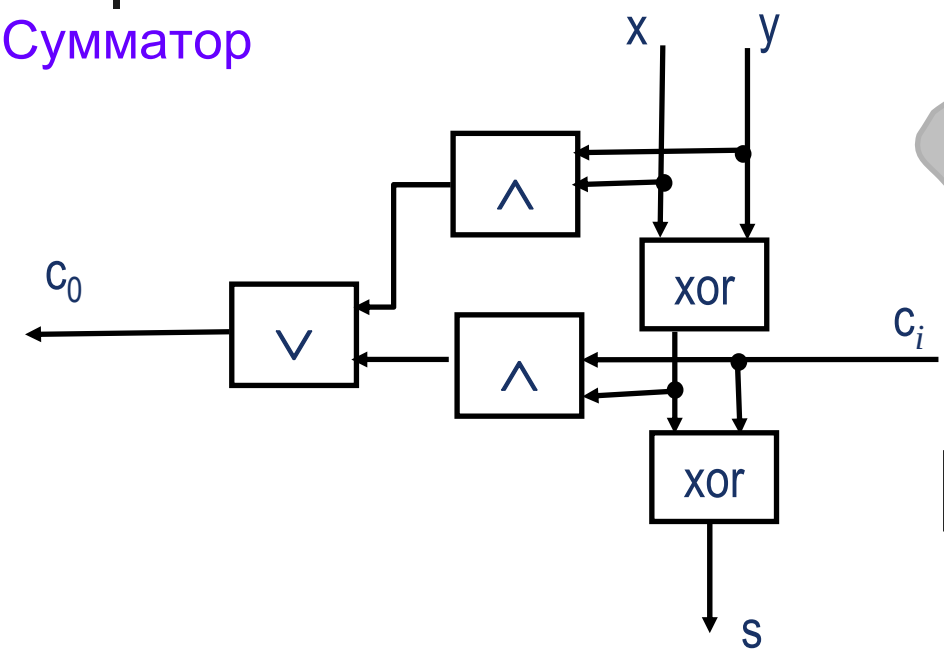
$F = a\neg bc + abc + ab\neg c$ $G = ac + bc$ $F = G$ (?)

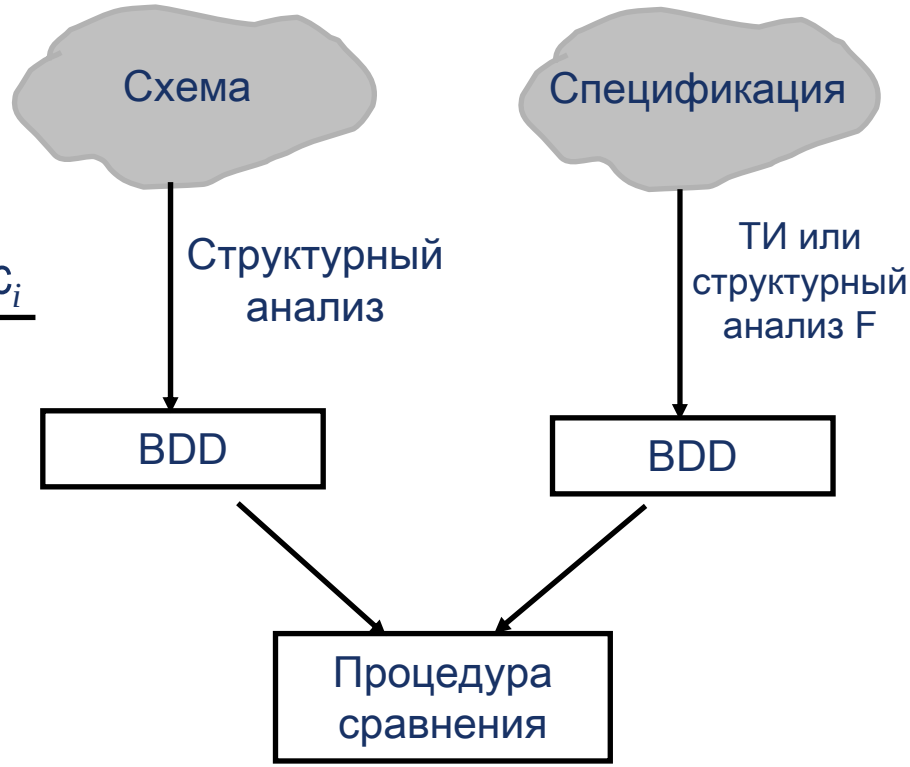


Поскольку BDD – каноническое представление, проверка эквивалентности двух функций, представленных в BDD, сводится к проверке совпадения направленных графов

Применение BDD: удовлетворяет ли логическая схема ее спецификации?

Сумматор





Структурный анализ схемы:

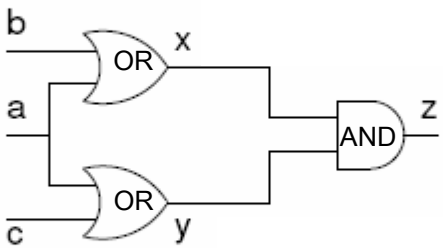
$s=(x \oplus y) \oplus c_i$

$c_0=x \wedge y \vee (x \oplus y) \wedge c_i$

Спецификация: ТИ или формула

Аналогично проверяется эквивалентность двух схем

Как построить BDD по структуре схемы

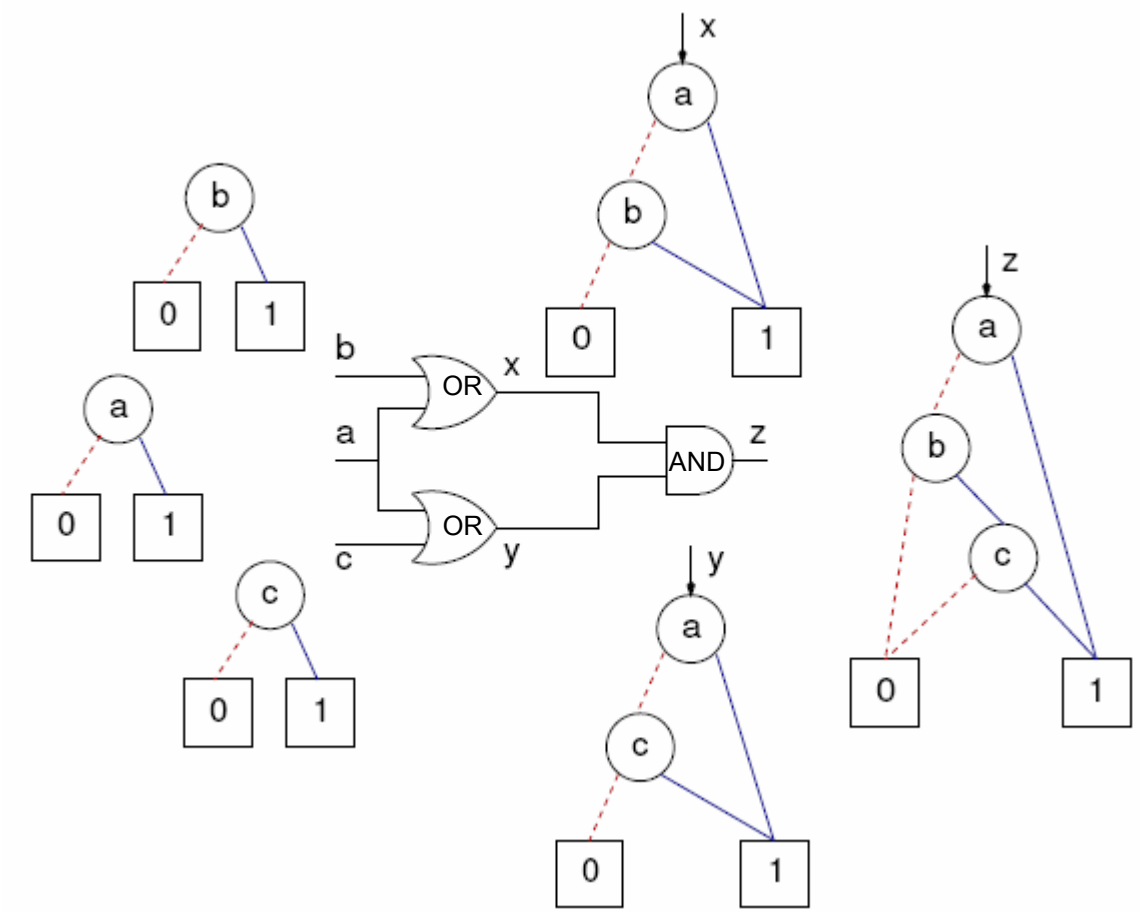


1. Сначала строятся BDD исходных переменных

2. Затем по ярусам строятся BDD всех промежуточных подфункций

3. Результирующая BDD представляет ЛФ, реализуемую схемой

Как построить BDD по структуре схемы

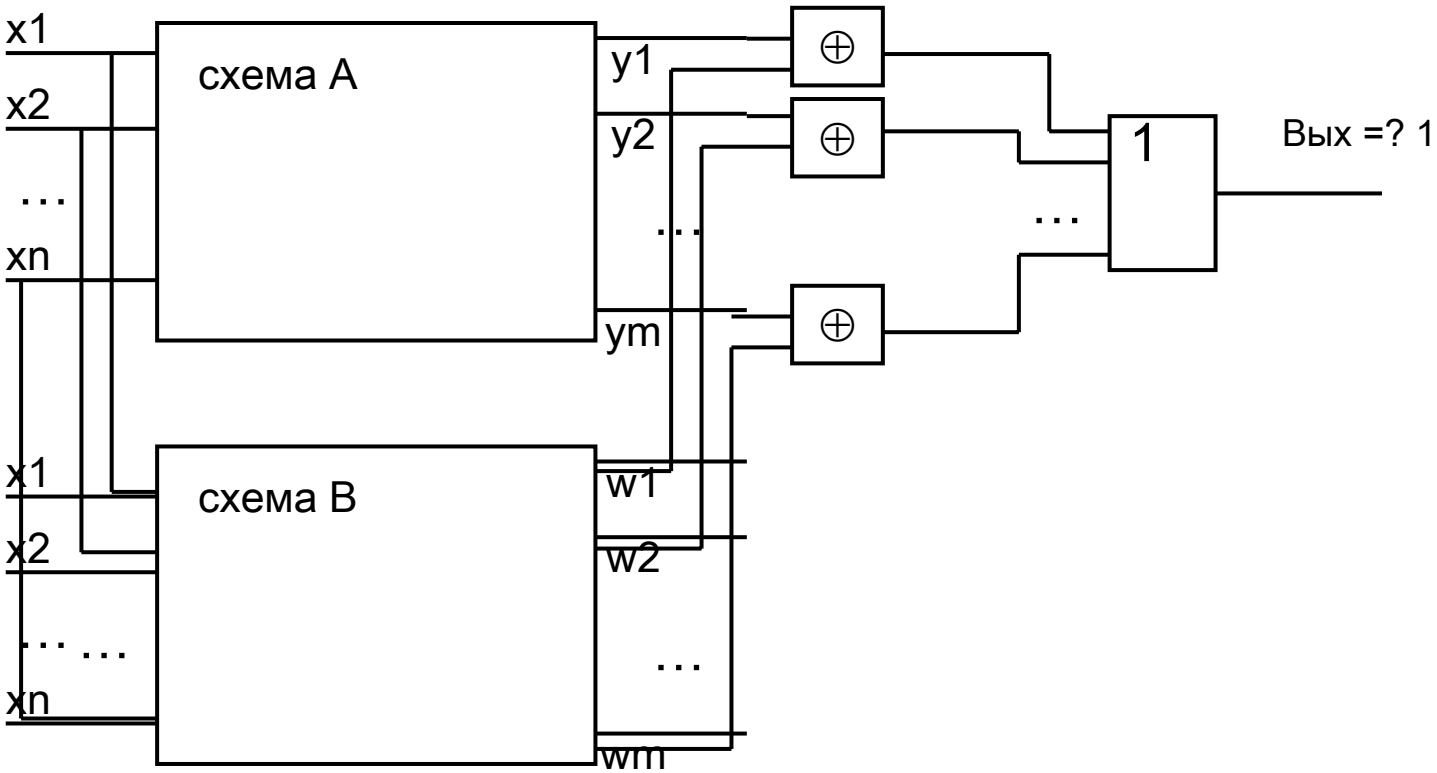


1. Сначала строятся BDD исходных переменных

2. Затем по ярусам строятся BDD всех промежуточных подфункций

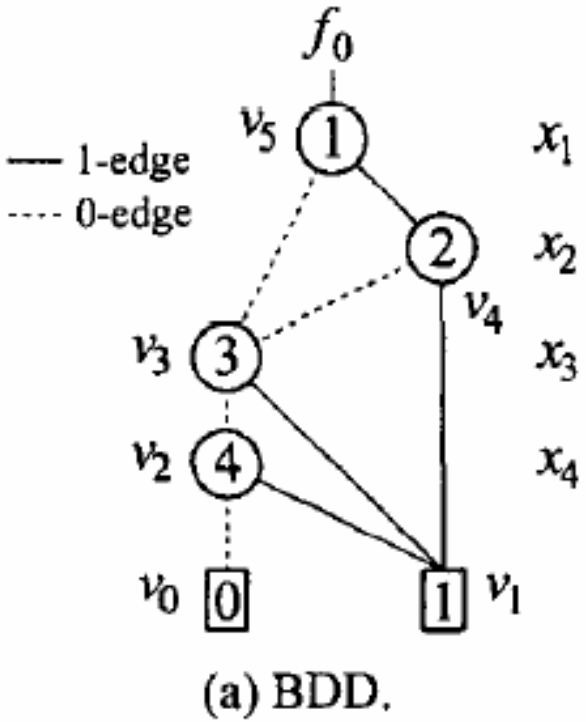
3. Результирующая BDD представляет ЛФ, реализуемую схемой

Эквивалентность двух схем – общий случай



- Строим БФ F для композиции и проверяем, существуют ли интерпретации, на которых F равна 1? Но это проблема SAT!
- С BDD эта проблема ОБЫЧНО решается просто!

Автоматическая генерация программы для вычисления значений функции по BDD



(a) BDD.

v_5: if(x_1 == 0) goto v_3;
else goto v_4;
v_4: if(x_2 == 0) goto v_3;
else goto v_1;
v_3: if(x_3 == 0) goto v_2;
else goto v_1;
v_2: if(x_4 == 0) goto v_0;
else goto v_1;
v_1: return(1);
v_0: return(0);

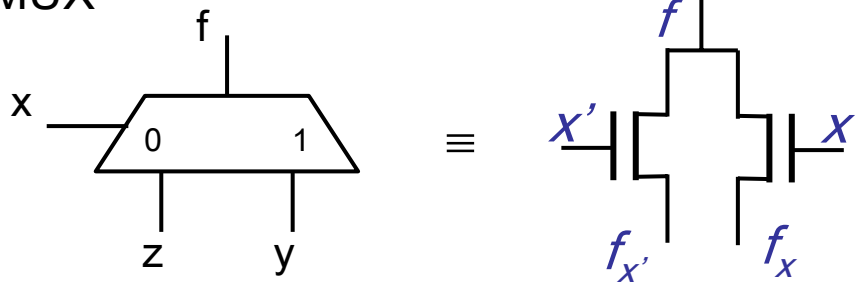
(b) Branching program generated from BDD.

Вычисление значений БФ по BDD – просто движение по упорядоченному графу. Найдем ответ за N шагов

Реализация схемы по BDD

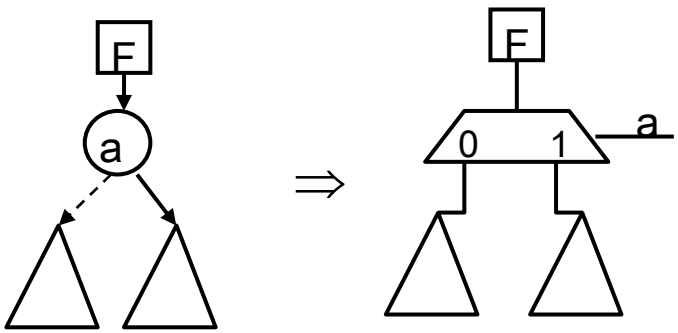
Используется мультиплексор (MUX) – схема, с тремя входами и одним выходом

Существует технология PTL (Pass Transistor Logic) построения ЛС на основе MUX

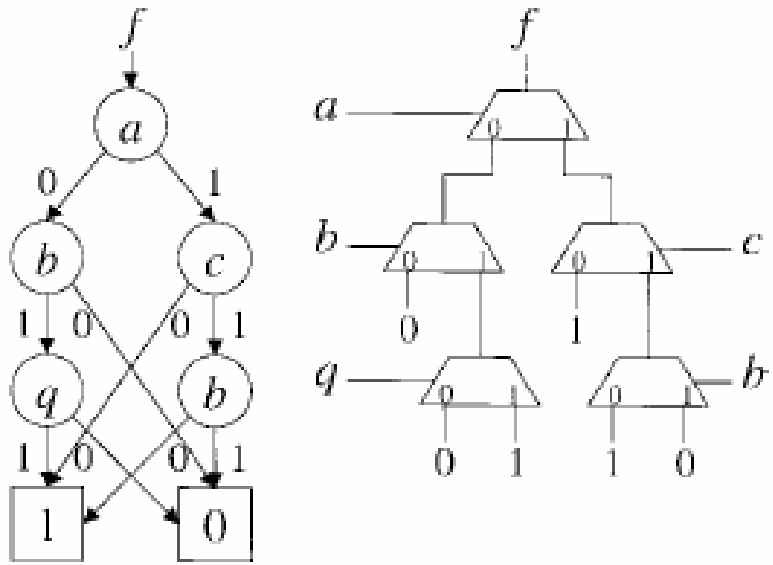


ite(x,y,z)=*if* x *then* y *else* z

$f = x\, f_x \vee x'\, f_{x'}$

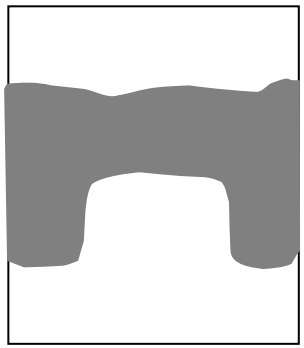


PTL синтез, основанный на "прямом" BDD mapping

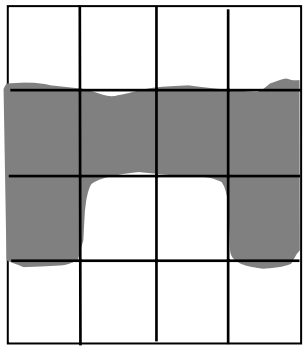


Интенсивно развивается направление BDD-Based Synthesis с PTL и комбинации PTL с CMOS логикой

Компрессия изображений



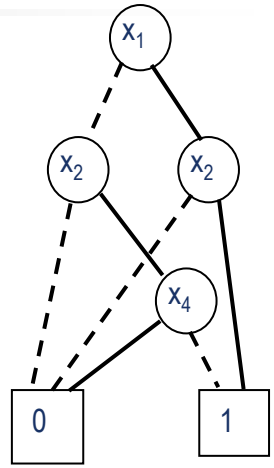
а)



б)

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

в)



г)

$f = x1x2 \vee x2\neg x4$

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 $\lceil \log (8*10^6) \rceil = 23$)

Использование BDD для поиска в БД

Код 0000 0001 0010 0011 ...

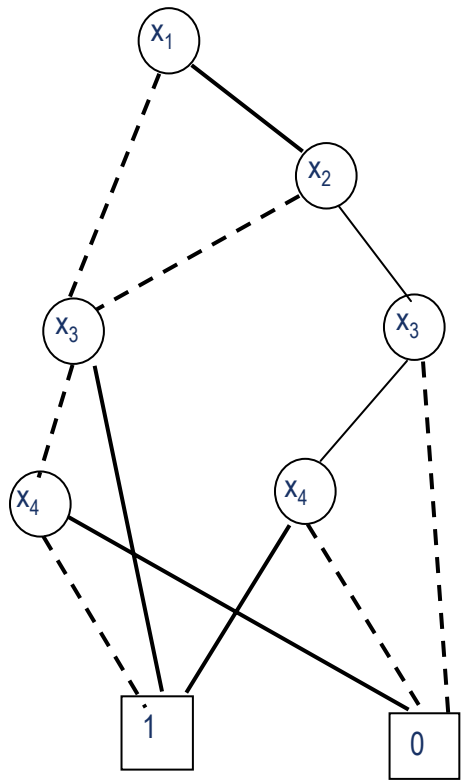
Документов 1 000 000 000; переменных 30

документы 0 1 2 3 4 5 6 7 8 9 10 ...

intensive	1 0 1 1 1 0 1 0 1 1 1 0 1 1 0 0 1
inversion	1 1 0 0 0 0 1 0 0 0 1 1 0 0 1
invert	1 0 1 0 1 1 0 0 0 1 1 1 0 1 1 0
inverter	0 0 0 1 1 0 0 1 1 0 0 0 1 0 1 0
invest	0 0 1 1 1 0 0 0 1 1 0 0 1 1 1 1
invest	1 1 1 1 1 0 0 0 1 1 1 0 0 1 1 1
investion	0 0 0 0 1 0 0 0 1 1 0 1 1 0 0 0
investment	1 0 0 1 1 0 0 0 1 1 1 1 1 1 1 0

$x_1x_2x_3x_4$	f
0000	1
0001	0
0010	1
0011	1
0100	1
0101	0
0110	1
0111	1
1000	1
1001	0
1010	1
1011	1
1100	0
1101	0
1110	0
1111	1

$f=\neg x_1x_3 \vee \neg x_1\neg x_4 \vee x_3x_4 \vee \neg x_2\neg x_4$

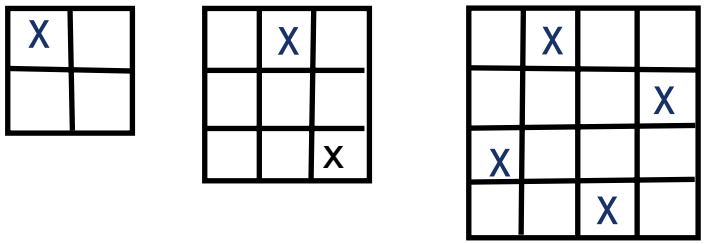


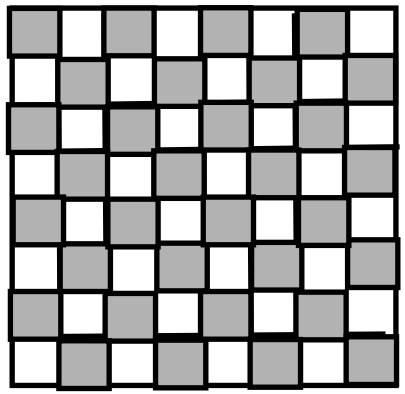
для слова intensive

- Над инвертированными списками выполняются булевы операции И, ИЛИ, НЕ
- Строим булеву функцию и ее BDD представление для каждого термина

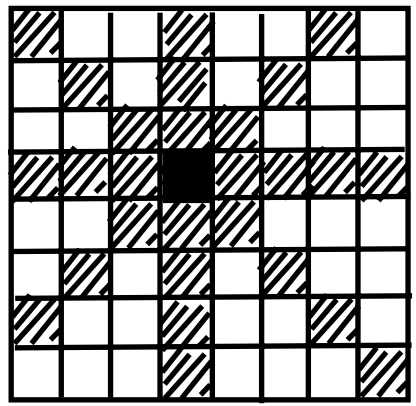
Логические задачи, решаемые с помощью БФ

Задача о расстановке ферзей





?



Если ферзь в клетке (*i,j*), *ТО:* ($x_{ij} \Rightarrow$):

на *i*-й горизонтали нет ферзей: $\bigwedge_{1 \le k \le N,\ k \ne j} \neg x_{ik}$

И на j-й вертикали нет ферзей: $\bigwedge_{1 \le k \le N,\ k \ne i} \neg x_{kj}$

И на с-з диагонали нет ферзей: $\bigwedge_{1 \le k \le N,\ 1 \le j+k-j \le N\ k \ne i} \neg x_{k,j+k-i}$

И на с-в диагонали нет ферзей: $\bigwedge_{1 \le k \le N,\ 1 \le j+I-k \le N\ k \ne i} \neg x_{k,j+i-k}$

И на каждой горизонтали есть ферзь: $\bigwedge_{1 \le i \le N} x_{i1} \vee x_{i2} \vee x_{i3} \ldots \vee x_{iN}$

Конъюнкция всех этих формул даст такую БФ F, что любая интерпретация, на которой F=1, будет решением

Для решения задачи нужна BDD для ф-ции от 64 переменных

Программирование в ограничениях

Constrained programming (Constraint Satisfaction Problem, CSP)

Основано на описании модели задачи, а не алгоритма ее решения

Модель специфицируется в виде отношений – ограничений, которые отражают связи, существующие между параметрами задачи

Постановка задачи

На переменные $Z_1,..., Z_n$ ($Z_i \in D_i$) наложены ограничения $C_k(Z_1, ..., Z_n)$ - уравнения, неравенства, логические выражения и т.п

Найти наборы значений $<a_1, ..., a_n>$ ($a_i \in D_i$), удовлетворяющие ограничениям

Использование BDD:

Значения в каждой области D_i кодируются двоичными наборами

Каждое ограничение C_k преобразуется в логическую функцию f_k , которая представляется в форме BDD

Множество ограничений представляется конъюнкцией этих функций $F = \&_i f_i$

Решение определяется такими наборами кодировок, на которых F =1

Пример: логическая головоломка

Задача (С.Расел, П.Норвиг: *Искусственный интеллект. Современный подход*)

В пяти домах разного цвета живут лица разных национальностей, которые пьют разные напитки, держат разных животных, курят разные сигареты

D^1: Цвет = {*красный, синий, желтый, зеленый, белый*} = { 000, …, 100 } (x_i^1, y_i^1, z_i^1)
D^2: Нац = {*англ, испанец, японец, норвежец, украинец*} = { 000, …, 100 } (x_i^2, y_i^2, z_i^2)
D^3: Напиток = {*сок, вино, чай, молоко, вода*} = { 000, …, 100 } (x_i^3, y_i^3, z_i^3)
D^4: Животное ={*улитка, лиса, кошка, лошадь, собака*} = { 000, …, 100 } (x_i^4, y_i^4, z_i^4)
D^5: Сигареты = {“Парламент”, “Кулс”…, “Честерфилд”} = { 000, …, 100 } (x_i^5, y_i^5, z_i^5)

Ограничения
C_1: Англичанин живет в доме желтого цвета
C_2: Человек, держащий лису, пьет сок
C_3: …

f_1 = (Нац = англ ⇔ Цвет = желтый)
f_2 = (Животное = лиса ⇔ Напиток = сок)

Q_1, ..., Q_5 – объекты (дома)

X_i^j – j-й параметр i-го объекта (25 переменных)

x_i^j, y_i^j, z_i^j – двоичные разряды кода X_i^j

$f_1 = \&_i (\neg x_i^2\ \neg y_i^2\ \neg z_i^2 \Leftrightarrow \neg x_i^1\ \ y_i^1\ \neg z_i^1)$
$f_2 = \&_i (\neg x_i^4\ \neg y_i^4\ \ z_i^4 \Leftrightarrow \neg x_i^3\ \neg y_i^3\ \neg z_i^3)$

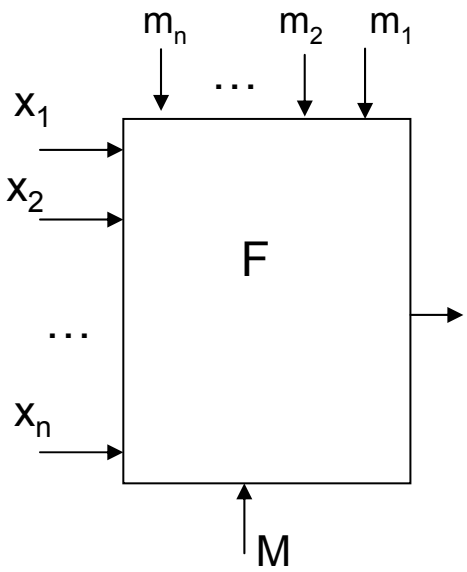
Возможные наборы параметров задаются теми наборами, на которых $F=\&_i f_i = 1$

BDD для функции F от 75 двоичных переменных решает задачу

Проблема SUBSET-SUM и BDD

- Проблема SUBSET-SUM.
  Дано S={ m_1, m_2, ..., m_n} и число M. Найти A $\subseteq$ S: $\Sigma_{mi \in A}$ = M
- Например, S={4, 5, 12, 97, 124, 13, 34 }, 7 чисел. M=140.
  Определить те подмножества S, сумма которых 140 (это 4, 12, 124)

- Известно, что SUBSET-SUM - NP-полная проблема. Сведем проблему к построению БФ в форме BDD вместо перебора

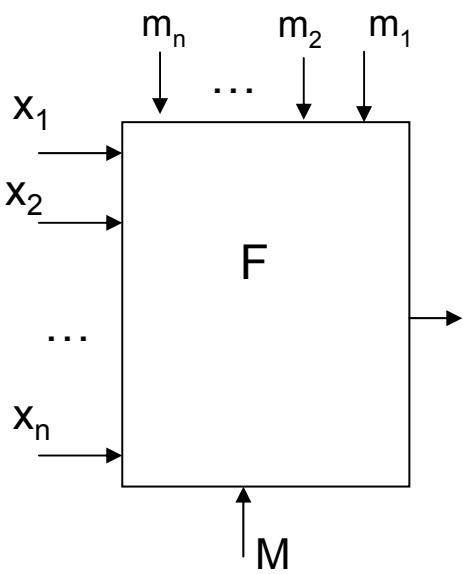


$$F(x_1, x_2, \ldots x_n) = 1 \text{ iff } \Sigma_i\, x_i\, m_i = M$$

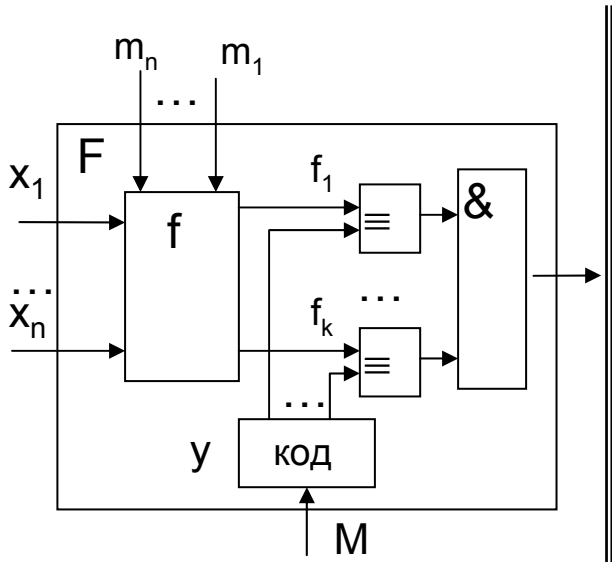
Все дело в том, как строить F

Построение BDD заменяет перебор с откатами

Например, S={4, 5, 12, 97, 124, 13, 34 }, M=140



F=1 на наборе 1101, если $M=m_1+m_3+m_4$



$f_k...f_2f_1$ – код суммы $\Sigma_i\ m_ix_i$

y – код числа M

$F = \&_i\ (f_i \equiv y_i)$

$x_3\ x_2\ x_1$	$f_5\ f_4\ f_3\ f_2\ f_1$	Пояснение
0 0 0	0 0 0 0 0	0
0 0 1	0 0 1 0 0	код m_1 (числа 4)
0 1 0	0 0 1 0 1	код m_2 (числа 5)
0 1 1	0 1 0 0 1	код m_2+m_1 (числа 9)
1 0 0	0 1 1 0 0	код m_3 (числа 12)
1 0 1	1 0 0 0 0	код m_3+m_1 (числа 16)
1 1 0	1 0 0 0 1	код m_3+m_2 (числа 17)
1 1 1	1 0 1 0 1	код $m_3+m_2+m_1$ (21)

Пример. Пусть S={ 4, 5, 12 }

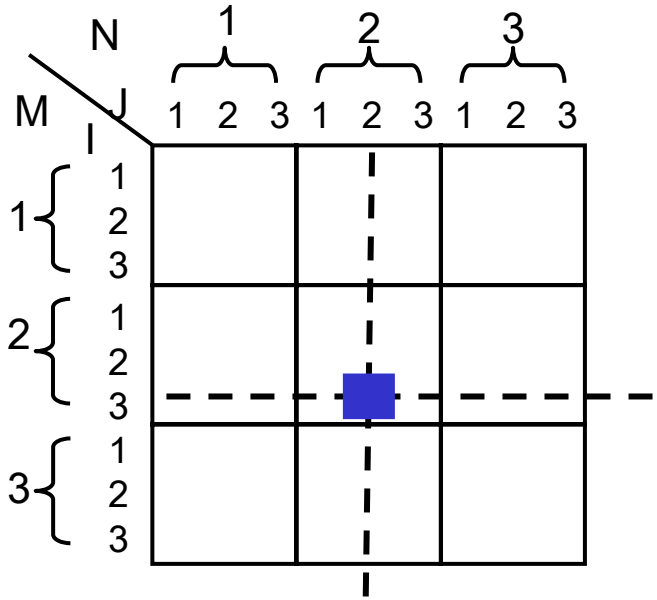
Построим функцию $f= f_1f_2 ... f_k$ – двоичный код суммы $\Sigma_i\ m_ix_i$

Простой алгоритм построения f оказался достаточным, чтобы время решения проблемы росло линейно в широком диапазоне n – до нескольких сотен
Вместо перебора в пространстве 10^{100} строим BDD логической функции

Пример: сложные комбинаторные задачи

Судоку:

9						1	2	
			1		2			
		5		8		6		
	7		8					4
		4				9		2
	3						8	
4		2		6			9	
5					9	3		
			7	4				



Цифры от 1 до 9, все разные:

на каждой горизонтали,
на каждой вертикали
и в каждом квадрате

$$A[M, N, I, J] \in \{1, 2, \ldots, 9\}$$

$$M, N, I, J, m, n, i, j \in \{1, 2, 3\}$$

Горизонталь: $\forall M\ \forall N\ \forall I\ \forall J\ \forall n\ \forall j\ [(n \neq N) \vee (j \neq J) \Rightarrow A[M, N, I, J] \neq A[M, n, I, j]]$

Вертикаль: $\forall M\ \forall N\ \forall I\ \forall J\ \forall m\ \forall i\ [(m \neq M) \vee (i \neq I) \Rightarrow A[M, N, I, J] \neq A[m, N, i, J]]$

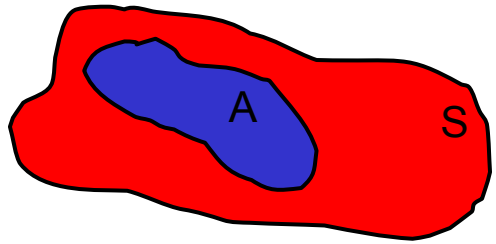
Квадрат: $\forall M\ \forall N\ \forall I\ \forall J\ \forall i\ \forall j\ [(i \neq I) \vee (j \neq J) \Rightarrow A[M, N, I, J] \neq A[M, N, I, J]]$

Для решения задачи нужна BDD для ф-ции от 324 переменных

Конечные структуры данных и BDD: характеристическая функция множества

$x_1x_2x_3$	f
000	1
001	1
010	1
011	0
100	0
101	0
110	1
111	0

Пусть S – конечное множество из m элементов, $k=\lceil \log_2(m) \rceil$. Закодируем все элементы S векторами двоичных переменных $x_1, x_2, \ldots, x_k$.



Пусть S состоит из 8 элементов: $\{a_0, a_1, \ldots, a_7\}$.
Закодируем элементы S (k=3): $a_0 \Leftrightarrow 000$, $a_1 \Leftrightarrow 001$, … , $a_7 \Leftrightarrow 111$

Пусть $A \subseteq S$, $A= \{a_0, a_1, a_2, a_6\}$. Соответствующее множество двоичных кодов { 000, 001, 010, 110}.

Булеву функцию можно задать множеством наборов, на которых функция принимает значение 1, например, $f(x_1,x_2,x_3)$ задается множеством { 000, 001, 010, 110}. Таким образом, подмножество A может быть представлено f

Любое подмножество конечного множества может быть задано с помощью булевой функции и, следовательно, с помощью ее BDD.

Эта функция называется характеристической.

Характеристические функции

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

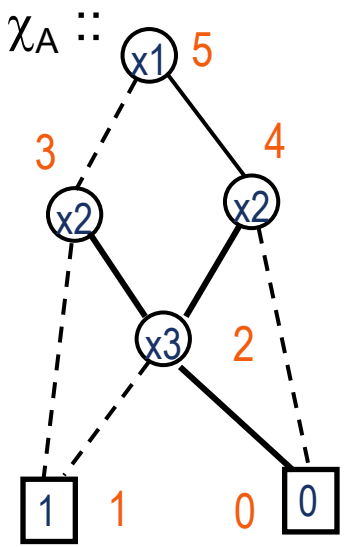
	f
000	1
001	1
010	1
011	0
100	0
101	0
110	1
111	0

Пусть x_1, x_2, x_3 – разряды кодировки.

Тогда:

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

задает A = {000, 001, 010, 110}



Итак, подмножества конечного множества можно задавать логической формулой, представляющей характеристическую функцию

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

Операции над множествами

Нульарные операции (константы):

- полное множество: $\chi_S = \text{True}$
- пустое множество: $\chi_\varnothing = \text{False}$

Унарная операция:

- дополнение множества: $\chi_{S-Q} = \neg\chi_Q$

Бинарные операции:

- пересечение множеств: $\chi_{P\cap Q} = \chi_P \wedge \chi_Q$
- объединение множеств: $\chi_{P\cup Q} = \chi_P \vee \chi_Q$
- разность множеств: $\chi_{P-Q} = \chi_P \wedge \neg\chi_Q$

ВСЕ операции над множествами можно выразить через булевы операции над характеристическими булевыми функциями

Представление отношений с помощью BDD

Пусть $S=\{a_0, \dots, a_{n-1}\}$ – множество

Введем кодирование: $a_0 \Leftrightarrow 000$, $a_1 \Leftrightarrow 001$, … ,$a_7 \Leftrightarrow 111$

Бинарное отношение на S – подмножество пар из S, R= { $(a_2,a_3),(a_7,a_4), (a_5, a_7)$ }

χ_R - характеристическая функция R равна 1 на кодировках $\{(a_2,a_3), (a_7,a_4), (a_5,a_7)\}$, т.е. на наборах 6 булевых переменных { 010 011, 111 100, 101 111 }

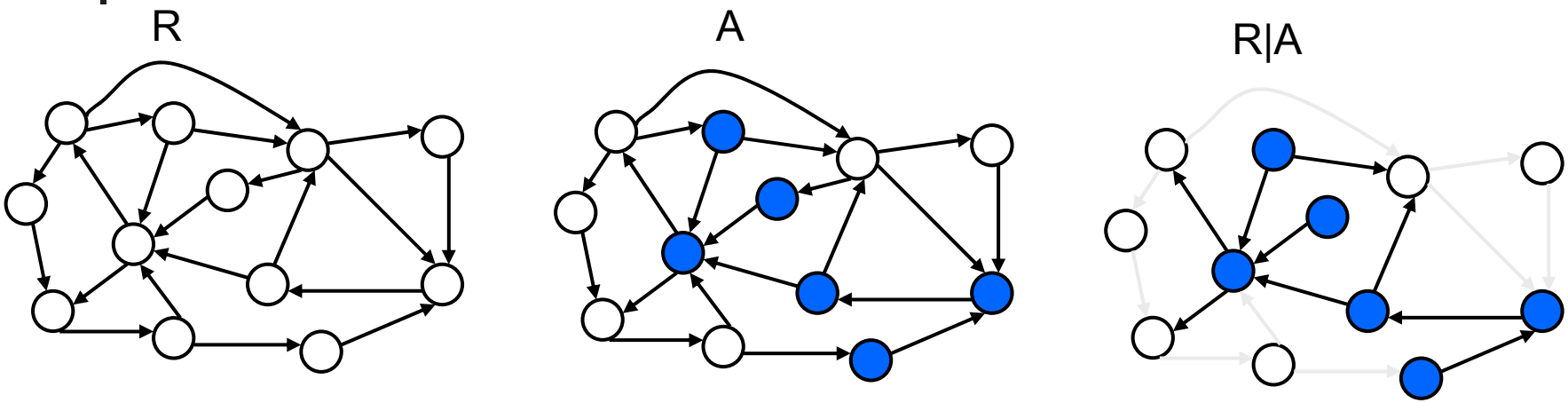
Первый элемент отношения R – это текущее состояние (pre-state), разряды его кода обозначим v (например, x_1, x_2, x_3). Второй элемент отношения – следующее состояние (post-state) и его код – v' (например, x_1', x_2', x_3')

Характеристическую функцию χ_R отношения R можно записать:

$(\neg x_1 x_2 \neg x_3 \neg x_1' x_2' x_3') \vee (x_1 x_2 x_3 x_1' \neg x_2' \neg x_3') \vee (x_1 \neg x_2 x_3 x_1' x_2' x_3')$

Характеристическая функция отношения может быть определена как логическая формула над штрихованныими и нештрихованными переменными с одним и тем же порядком

Ограничение отношения на подмножестве



Пусть R – бинарное отношение на S, и A – подмножество S.

Обозначим χ_R - и χ_A – характеристические функции R и A

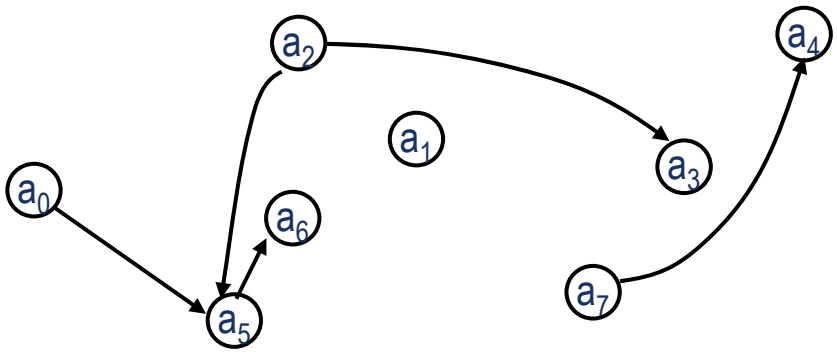
Характеристическая функция ограничения R на A строится как χ_A& χ_R

Это все те пары отношения R, первый элемент которых принадлежит A

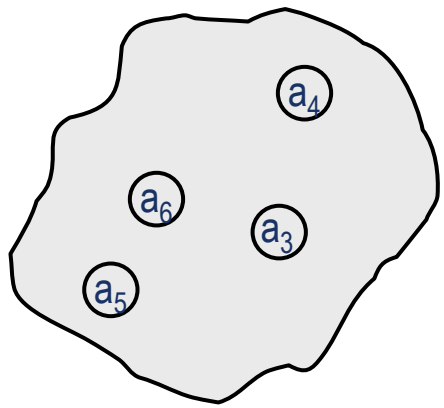
Операции над отношениями: Прямой и обратный образы

Пусть ХФ $\chi_R(v,v')$ определяет бинарное отношение (множество переходов)

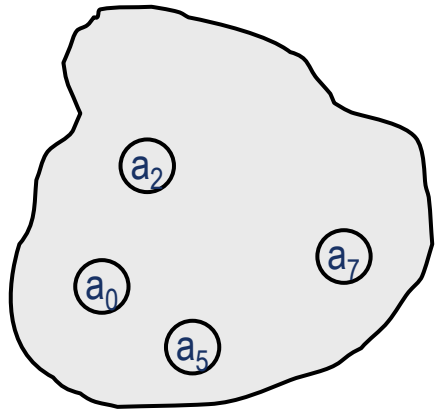
$S = \{a_0, a_1, \ldots, a_7\}$



$\exists v.\chi_R(v')$



$\exists v'.\chi_R(v)$



ХФ множества тех состояний, в которые переходы возможны, задается операцией Прямой образ: $Post(R) = \exists v.\chi_R(v')$ - это ХФ множества $\{a_3, a_4, a_5, a_6\}$ – в них есть переходы из каких-то элементов множества S.

ХФ множества тех состояний, из которых переходы возможны, задается операцией Обратный образ: $Pre(R) = \exists v'.\chi_R(v)$ - это ХФ множества $\{a_0, a_2, a_5, a_7\}$ – из этих элементов есть переходы в какие-либо элементы множества S

Как реализовать эти функции в BDD?

Операция квантификации над функциями

Пусть множество A={0010, 0101, 1011, 0110, 0011, 1100} представлено характеристической функцией от булевых переменных (x_1, x_2, x_3, x_4)

Квантификация по переменной x_2 (это просто выбрасывание этой переменной) определяет множество

B = {0~~0~~10, 0~~1~~01, 1~~0~~11, 0~~1~~10, 0~~0~~11, 1~~1~~00}
= {010, 001, 111, 010, 011, 100}
= {010, 001, 111, 011, 100}
от переменных (x_1,x_3,x_4)

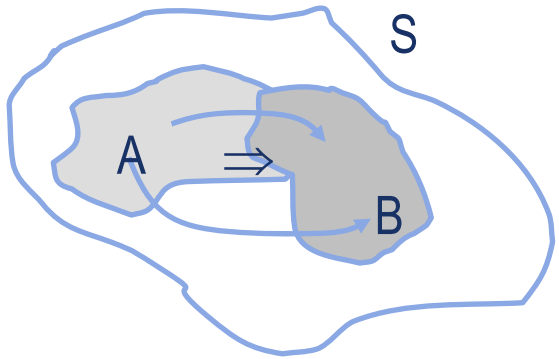
Определение квантификации характеристической функции:
$\exists(x).f(x,y) = f(0,y) \vee f(1, y)$;
– результат не зависит от x Это проекция на оставшиеся переменные

Пример: Операция квантификации: $\exists x_2.\chi_A$ определяет ХФ множества
{0010, 0110, 0101, 0001, 1011, 1111, 0010, 0111, 1100, 1000 } (x_1, x_2, x_3, x_4) = { 010, 001, 111, 011, 100 } (x_1,x_3,x_4)

Квантификация по нескольким переменным вычисляется последовательно

Прямой Образ для бинарных отношений

Пусть $A \subseteq S$ – подмножество S,
R – бинарное отношение на S
В какие элементы S можно перейти из A?



1. Определяем ограничение отношения R на тех начальных элементах R из S, которые принадлежат A:

$$\chi_{A\,(v)} \;\&\; \chi_{R(v,\,v')}$$

2. Строим B= Forward Image (A,R)=Прямой Образ A относительно R:

$$\chi_{Image(A,R)\ (v)} = \exists v.[\ \underbrace{\chi_{A\,(v)} \;\&\; \chi_{R\,(v,\,v')}}\ ] < v/v' >$$

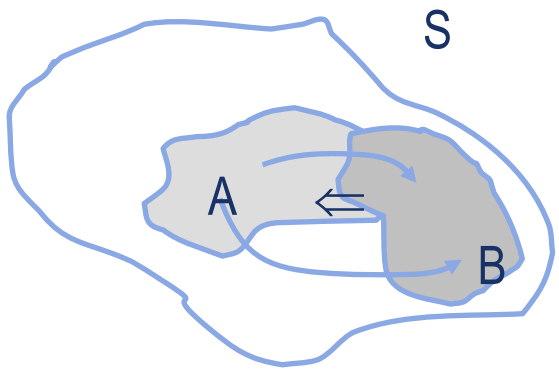
Переходы из элементов $\in A$

(< v / v > - это замена переменными v штрихованных значений v')

Итак, чтобы найти множество B всех тех элементов S, которые достижимы за один шаг отношения R из элементов множества A, нужно выполнить с булевыми характеристическими функциями $\chi_{A\,(v)}$ и $\chi_{R\,(v,\,v')}$ две операции: конъюнкцию и квантификацию, и выполнить переименование переменных

Обратный Образ для бинарных отношений

Пусть $B \subseteq S$ – подмножество S,
R – бинарное отношение на S
Из каких элементов S можно перейти в B?



1. Строим ограничение отношения R на тех вторых элементах R из S, которые принадлежат B:

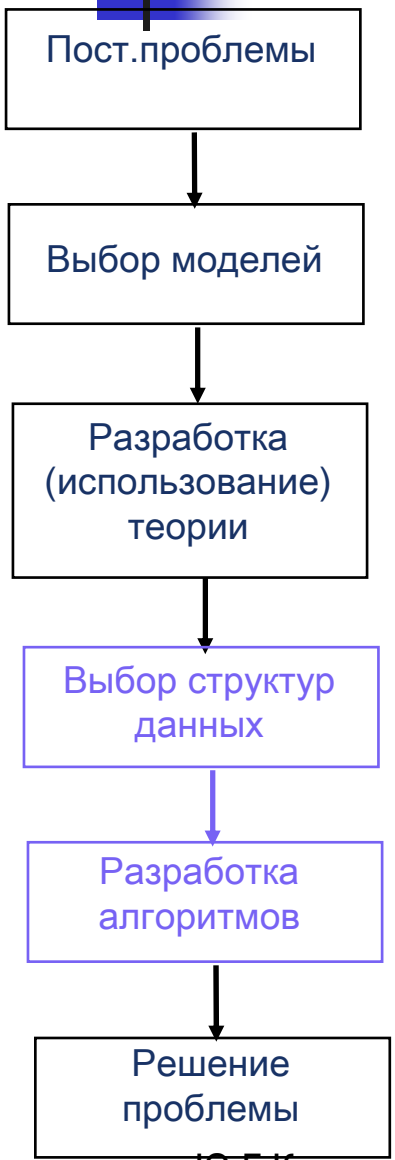
$$\chi_{B\,(v')} \;\&\; \chi_{R\,(v,v')}$$

2. Строим A= Reverse Image (B,R) = обратный образ B относительно R:

$\chi_{RImage(B,R)\ (\ v\ )} = \exists v'.(\ \chi_{B\ (v')}\ \&\ \chi_{R\ (v,v')}\ )$ – выбрасываем все вторые элементы

Чтобы найти множество A всех тех элементов S, из которых за один шаг отношения R достижимы элементы заданного множества B, нужно выполнить с булевыми характеристическими функциями $\chi_{B\ (v')}$ и $\chi_{R\ (v,\ v'\ )}$ переименование и две операции: конъюнкцию и квантификацию

Решение проблем с помощью BDD



Пример: анализ достижимости на графе $\Gamma=(S,R,S_0)$

Пусть S_i – множество вершин, достижимых после i или меньшего числа переходов из S_0.

Очевидно, $S_{i+1} = S_i \cup$ Прямой Образ (S_i, R)

Алгоритм с помощью BDD

1. Представляем множества S_0 и R в форме BDD, т.е. определяем множество v переменных, кодирующих состояния из S, а также функции $\chi_{S0\ (v)}$ и $\chi_{R\ (v,v')}$ в форме BDD
2. Полагаем $S_i = S_0$, т.е. $\chi_{Si\ (v)} = \chi_{S0\ (v)}$

Решение проблем с помощью BDD

Пост.проблемы

Выбор моделей

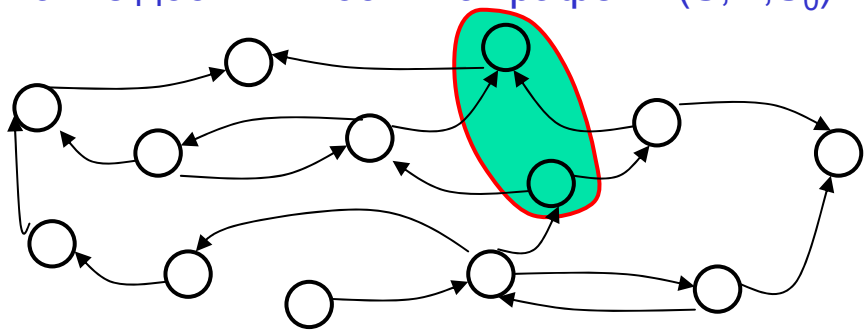
Разработка (использование) теории

Выбор структур данных

Разработка алгоритмов

Решение проблемы

Пример: анализ достижимости на графе $\Gamma=(S,R,S_0)$



1. Представляем множества S_0 и R в форме BDD, т.е. определяем множество v переменных, кодирующих состояния из S, а также функции $\chi_{S0\ (v)}$ и $\chi_{R\ (v,v')}$ в форме BDD

2. Полагаем $S_i = S_0$, т.е. $\chi_{Si\ (v)} = \chi_{S0\ (v)}$

3. Итеративно вычисляем ограничение R на S_i

$\chi_{R|Si\ (v,v')} = \chi_{Si\ (v)}\ \&\ \chi_{R\ (v,v')}$,

и куда можем перейти из S_i:

$\chi_{Si+1(v)} = [\ \chi_{Si\ (v)} \vee \exists v.\ \chi_{R|Si\ (v,v')}\ (\ v\ /\ v'\ )\ ]$

Решение проблем с помощью BDD

Пост.проблемы

Выбор моделей

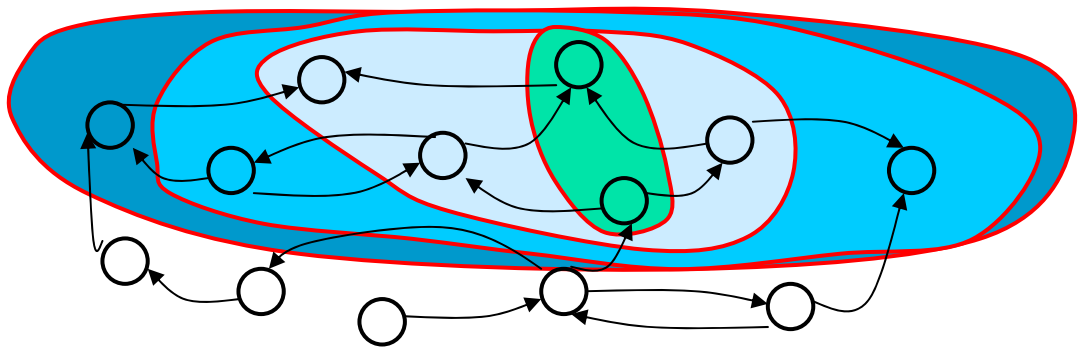
Разработка (использование) теории

Выбор структур данных

Разработка алгоритмов

Решение проблемы

Пример: анализ достижимости на графе $\Gamma=(S,R,S_0)$



1. Представляем множества S_0 и R в форме BDD, т.е. определяем множество v переменных, кодирующих состояния из S, а также функции $\chi_{S0\ (v)}$ и $\chi_{R\ (v,v')}$ в форме BDD

2. Полагаем $S_i = S_0$, т.е. $\chi_{Si\ (v)} = \chi_{S0\ (v)}$

3. Итеративно вычисляем ограничение R на S_i

$\chi_{R|Si\ (v,v')} = \chi_{Si\ (v)}\ \&\ \chi_{R\ (v,v')}$,

и куда можем перейти из S_i:

$\chi_{Si+1(v)} = [\ \chi_{Si\ (v)} \vee \exists v.\ \chi_{R|Si\ (v,v')}\ \ (\ v\ /\ v'\ )\ ]$

4. Алгоритм останавливается, когда $\chi_{Si+1(v)} = \chi_{Si\ (v)}$

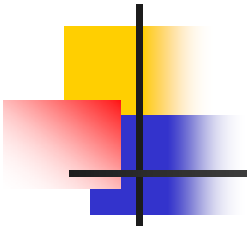
Используем характеристические булевы функции для S_0 и R, а не работаем с каждой вершиной и ребром - сложность полиномиальна на структурах, растущих экспоненциально

BDD в обработке данных

- BDD широко используется для повышения эффективности алгоритмов, работающих с конечными структурами данных
- Можно провести некоторую аналогию с использованием позиционной числовой записи вместо унарного кода (в унарной системе счисления количество N представляется N единицами)
- Это, фактически, революция в областях, связанных с обработкой дискретной информации
- В последнее время 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)

Заключение

- БФ удобны для решения многих задач, но реально с классическими представлениями БФ можно работать только с 4-6 переменными
- BDD – новая компактная форма представления полностью определенных БФ в виде направленного графа. Сложность BDD зависит от порядка переменных
- Для большинства БФ представление в BDD линейно или полиномиально. Но существуют классы БФ, для которых представление в BDD экспоненциально
- BDD имеют множество привлекательных свойств:
  - каноническое представление
  - логические операции с BDD выполняются эффективно (линейно)!!!
  - многие задачи (например, SAT) решаются эффективно
  - BDD может представлять любые конечные структуры данных на основе использования булевых характеристических функций множеств, отношений, функций и т.п.
  - ...
- Основное свойство BDD – возможность работы с полиномиальным временем на структурах, растущих экспоненциально
- Существует много расширений BDD



Спасибо за внимание!