

КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

*Институт вычислительной математики
и информационных технологий
Кафедра технологии программирования*

МЕТОДИЧЕСКОЕ ПОСОБИЕ ПО
ПРЕДМЕТУ «АЛГОРИТМЫ И
СТРУКТУРЫ ДАННЫХ В ПОСТРОЕНИИ
И АНАЛИЗЕ СБИС»

Казань – 2015

УДК 519.6
М54

*Печатается по решению заседания учебно-методической комиссии
Института вычислительной математики и информационных технологий
Протокол № 5 от 25 декабря 2014 г.*

*заседания кафедры технологии программирования
Протокол № 2 от 13 октября 2014 г.*

Составители

канд. физ.-мат. наук, доцент **Р.Г.Мубаракзянов**,
ассист. **М.М.Абрамский**

Рецензент

канд. техн. наук, ст. препод. **В.О.Георгиев**

М54 Методическое пособие по предмету «Алгоритмы и структуры данных в построении и анализе СБИС»: учебно-методическое пособие / Р.Г.Мубаракзянов, М.М.Абрамский. – Казань: Казан. ун-т, 2014. – 58 с.

Данное пособие предназначено для студентов, изучающих программную дисциплину «Алгоритмы и структуры данных в построении и анализе СБИС», а также для преподавателей, ведущих занятия по данному курсу.

© Казанский университет, 2015

Оглавление

Введение.	4
1. Теоретические основы описания и конструирования схем	5
1.1 Булевы алгебры и булевы функции.	5
1.2 Переключательные функции	10
1.3 Важные свойства булевых функций	12
2. Вычислимые функций и сложность вычислений (неформальное введение)	16
2.1 Вычислимость функций	16
2.2 Сложность функций	17
2.3 Примеры труднорешаемых задач	19
2.4 Полиномиальная сводимость и NP -полнота	20
3. Представления функций	26
3.1 Обзор подходов	26
3.2 Ограниченные классы бинарных программ.	32
3.3 Требование к структурам данных в формальной верификации схем	36
3.4 <i>OBDD</i> – эффективная структура данных	39
3.5 Свойства редуцированных <i>OBDD</i>	41
3.6 Алгоритм редукции	44
4. Эффективная реализация <i>OBDD</i>	48
4.1 Основные идеи	48
4.1.1 Представление отдельного узла <i>OBDD</i>	48
4.1.2 Разделяемые (shared) <i>OBDD</i>	48
4.1.3 Уникальная таблица (unique) таблица и строгая совершенство	49
4.1.4 <i>ITE</i> -алгоритм.	50
4.1.5 Дополнительные дуги (complemented edges).	53
4.1.6 Стандартные тройки.	54
4.1.7 Управление памятью	56
4.2 Популярные <i>OBDD</i> -пакеты	57
Литература	58

Введение

Основными компонентами компьютеров и других цифровых систем являются сверхбольшие интегральные схемы (СБИС или VLSI), которые представляют собой сложную комбинацию (со-единение) ограниченного числа основных, или функциональных, элементов, осуществляющих простую логическую операцию.

В схеме один или несколько заряженных входных портов (или просто входов) определенным образом соединяются с одним или несколькими выходными портами (выходами). В простейшей модели различаются два напряжения на входах и выходах: «заряжено», которое обозначается «1», и «не заряжено», которое обозначается «0». В действительности, такое бинарное представление символизирует, что заряд не выходит за пределы двух непересекающихся непрерывных интервалов зарядки. Таким образом, хотя результат работы функционального элемента, зависит не от точного значения входного сигнала, а от соответствующего интервала зарядки, можно моделировать (кодировать) входные и выходные значения функциональных элементов «0» и «1». Элементы и схемы с n входами и m выходами соответствуют функциям $\{0; 1\}^n \rightarrow \{0; 1\}^m$.

Настоящее методическое пособие представляет собой конспект курса лекций, прочитанных для студентов ИВМиИТ КФУ и посвященных анализу алгоритмов и структур данных, используемых в построении и анализе СБИС. Несмотря на актуальность данной тематики, в русскоязычной литературе практически отсутствует материал на эту тему. Курс базируется в основном на книге [3], а также на статьях различных авторов и собственных результатах одного из авторов.

Глава 1. Теоретические основы описания и конструирования схем

1.1 Булевы алгебры и булевы функции

В 1854 году выдающийся английский математик Джордж Буль сформулировал основные законы математической логики (алгебры логики). Американский ученый Клод Шеннон в 1936 году показал, что эти законы могут быть использованы в описании и анализе схем из функциональных элементов, которые и по сей день являются одной из мощных моделей вычисления функций, используемой для реализации алгоритмов вычисления в физических устройствах.

По традиции, основные объекты в алгебре логики называют «булевыми» в честь Дж.Буля (например, булева функция, булева формула, булева алгебра и т.п.). Рассмотрим основные определения и свойства булевых алгебр и булевых функций, которые понадобятся нам в дальнейшем.

Определение 1.1.1 *Булева алгебра – набор $\langle A, +, \cdot, -, 0, 1 \rangle$, где A – множество, содержащее элементы 0 и 1 , с определенными на нем бинарными операциями $+$ и $\cdot$ («сложение» и «умножение») и унарной операцией $-$ (или $\neg$) («отрицание») таких, что для всех a, b из A выполняются:*

1. *Коммутативный закон:*

$$a + b = b + a,$$

$$a \cdot b = b \cdot a.$$

2. *Распределительный закон:*

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c),$$

$$a + (b \cdot c) = (a + b) \cdot (a + c).$$

3. *Существование нейтральных элементов из A для бинарных операций $+$ и $\cdot$:*

$$a + 0 = a$$

$$a \cdot 1 = a$$

4. *Существование обратного элемента из A относительно бинарных операций $+$ и $\cdot$:*

$$a + \bar{a} = 1$$

$$a \cdot \bar{a} = 0$$

Принято считать, что операция $-$ имеет приоритет над операцией $\cdot$, а $\cdot$ имеет приоритет над $+$. То есть, к примеру, выражение $\bar{a} \cdot b$ нужно понимать как $(\bar{a}) \cdot b$, а $a + b \cdot c$ как $a + (b \cdot c)$.

В дальнейшем, будем периодически опускать знак " $\cdot$ " и писать ab вместо $a \cdot b$.

Примеры булевых алгебр.

Пример 1.1.1. Пусть 2^S – множество подмножеств S , $\bar{A} = S - A$. Тогда алгебра подмножеств $\langle 2^S, \cup, \cap, -, \emptyset, S \rangle$ является булевой алгеброй с нейтральными элементами $\emptyset$ и S относительно бинарных операций «объединение» ($\cup$) и «пересечение» ($\cap$) соответственно.

Пример 1.1.2. Пусть $n > 1$ имеет лишь различные простые делители: $n = p_1 p_2 \dots p_k$, $p_1 < p_2 < \dots < p_k$, T_n – множество делителей n . Тогда $\langle T_n, \text{НОК}, \text{НОД}, (\cdot)^{-1}, 1, n \rangle$ является булевой алгеброй с нейтральными элементами 1 и n относительно бинарных операций «Наименьшее общее кратное (НОК)» и «Наибольший общий делитель (НОД)», соответственно; а также:

$$\bar{a} = (a)^{-1} = \frac{n}{a}.$$

Домашнее задание 1.1.1. Доказать, что предложенное в примере определение действительно представляет алгебру.

Булева алгебра удовлетворяет принципу двойственности:

Определение 1.1.2. Пусть G – некоторое равенство в булевой алгебре $\langle A, +, \cdot, -, 0, 1 \rangle$. Равенство G' называется двойственным к G , если оно получено из G взаимной заменой "+" на " $\cdot$ " и 0 на 1 .

Например, для равенства

$$G: a + 0 = a$$

двойственным является

$$G': a \cdot 1 = a.$$

Теорема 1.1.1. (Принцип двойственности) Пусть G' – двойственное равенство для G . Если G – истинное высказывание в булевой алгебре, то G' также истинно.

Справедливость теоремы очевидна, так как аксиомы булевой алгебры сохраняют истинность при переходе к своим двойственным аналогам.

Таким образом, исходя из принципа двойственности, достаточно доказывать высказывания в булевой алгебре лишь в одной из двух версий.

Теорема 1.1.2. (Основные законы вычислений) Для любых элементов a, b, c из булевой алгебры $\langle A, +, \cdot, -, 0, 1 \rangle$ выполняются:

1. Законы идемпотентности:

$$\begin{aligned}a \cdot a &= a, \\a + a &= a.\end{aligned}$$

2. Свойства нейтральности элементов:

$$\begin{aligned}a + 1 &= 1, \\a \cdot 0 &= 0.\end{aligned}$$

3. Законы поглощения:

$$\begin{aligned}a + (a \cdot b) &= a, \\a \cdot (a + b) &= a.\end{aligned}$$

4. Законы ассоциативности:

$$\begin{aligned}a + (b + c) &= (a + b) + c, \\a \cdot (b \cdot c) &= (a \cdot b) \cdot c:\end{aligned}$$

5. Законы Де-Моргана:

$$\begin{aligned}\overline{a + b} &= \bar{a} \cdot \bar{b}, \\\overline{a \cdot b} &= \bar{a} + \bar{b}:\end{aligned}$$

6. Закон двойного отрицания:

$$\bar{\bar{a}} = a.$$

Докажем, например, закон поглощения.

$$a + (a \cdot b) = a \cdot 1 + a \cdot b = a \cdot (1 + b) = a \cdot 1 = a.$$

Домашнее задание 1.1.2. Доказать остальные утверждения.

Определение 1.1.3. Булевы функции – это функции, описываемые выражениями булевой алгебры или булевыми формулами.

Определение 1.1.4. Пусть $B = \langle A, +, \cdot, -, 0, 1 \rangle$ – булева алгебра. Выражение C , содержащее переменные $x_1, \dots, x_n$, символы $+$, $\cdot$, $-$ и элементы A , является булевой формулой в булевой алгебре B над переменными $x_1, \dots, x_n$, если:

1. C – это x , $x \in A$;
2. C – это x , $x \in \{x_1, \dots, x_n\}$;
3. C – это (для некоторых булевых формул F и G):
 - или $(F) + (G)$;
 - или $(F) \cdot (G)$;
 - или (F) .
4. C может быть получено конечным числом применений правил 1, 2, 3.

Замечание 1.1.1 Используя приоритеты операций, можно опускать скобки.

Определение 1.1.5 Булева формула F индуцирует (порождает) функцию $f: A^n \rightarrow A$, если значение функции $f(a_1, \dots, a_n)$ равно значению, получаемому при замене в формуле F переменных x_i на значения $a_i \in A$.

Определение 1.1.6 Пусть $B = \langle A, +, \cdot, -, 0, 1 \rangle$ – булева алгебра. Функция $f: A^n \rightarrow A$ называется булевой функцией, если она может быть порождена булевой формулой. Будем говорить, что формула F представляет функцию f .

Таким образом, булевы функции – это функции, описываемые выражениями булевой алгебры или булевыми формулами.

Определение 1.1.7 Булевы функции f и g от n переменных называются эквивалентными ($f \approx g$), если их значения совпадают на всех 2^n входных наборах из $\{0, 1\}^n$.

Теорема 1.1.3 Булевы функции f и g эквивалентны тогда и только тогда, когда они совпадают ($f = g$), т.е.

$$\forall (a_1, \dots, a_n) \in A^n: f(a_1, \dots, a_n) = g(a_1, \dots, a_n).$$

Доказательство: Очевидно, если функции совпадают, то они эквивалентны.

Покажем, что верно и обратное. Из аксиом и основных законов (Теорема 0.1.2) любая булева формула может быть переписана в виде суммы:

$$F = \sum_{(a_1, \dots, a_n) \in \{0, 1\}^n} b(a_1, \dots, a_n) x_1^{a_1} \dots x_n^{a_n},$$

где

$$x_i^1 = x_i, \quad x_i^0 = \bar{x}_i, \quad b(a_1, \dots, a_n) \in A:$$

Действительно, используя закон Де-Моргана, можно отрицание использовать по отношению только к переменным (то есть получим литералы). Раскрывая скобки, получаем сумму произведений литералов (мономов). Слагаемое, не содержащее одну из переменных x_i , заменяется на соответствующую сумму двух мономов, в один из которых x_i входит без отрицания, а в другое – с отрицанием. Таким образом, любая булева функция однозначно определяется коэффициентами $b(a_1, \dots, a_n)$, которые зависят лишь от конкретных наборов $(a_1, \dots, a_n) \in \{0, 1\}^n$. Следовательно, если $f \approx g$, т.е. для любого набора $(a_1, \dots, a_n) \in \{0, 1\}^n$

$$f(a_1, \dots, a_n) = g(a_1, \dots, a_n) = b(a_1, \dots, a_n),$$

то f совпадает с g ($f = g$). $\square$

Следствие 1.1.1 Количество различных булевых функций равно $|A|^{2^n}$.

1.2 Переключательные функции

Теоретической основой построения схем является специальный случай булевой алгебры с двумя элементами, поэтому далее ограничимся рассмотрением лишь данной модели.

Важность этой алгебры была показана еще в 1910 году физиком Паулем Эренфестом, занимавшимся разработкой переключательных схем в телефонных сетях. Исследования, посвященные этой тематике, получили наибольшее продвижение в 1936-38 гг. в Японии, США, СССР, среди которых выделяются работы Клода Шеннона.

Итак, рассмотрим следующую булеву алгебру:

$$\langle B, +, \cdot, -, 0, 1 \rangle,$$

где $B = \{0, 1\}$, $a + b = \max\{a, b\}$, $a \cdot b = \min\{a, b\}$, $\bar{0} = 1$, $\bar{1} = 0$.

Определение 1.2.1. Функция $f: B^n \rightarrow B$ от n переменных называется переключательной функцией.

Множество переключательных функций будем обозначать как B_n . Легко показать, что $|B_n|$ (количество различных переключательных функций) равно 2^{2^n} .

Из следствия 1.1.1 предыдущей главы следует, что количество различных булевых функций в $|B_n|$ равно 2^{2^n} . Следовательно, любая переключательная функция является булевой функцией. Поэтому в дальнейшем будем использовать термин «булева функция», подразумевая переключательную функцию.

Булева функция с n входами и m выходами описывается « m »-кой $f = (f_1, \dots, f_m)$, где $f_1, \dots, f_m$ – булевы функции. Обозначим такие функции как $B_{n,m}$ ($B_{n,1} = B_n$).

Теорема 1.2.1. Число различных булевых функций от n переменных с m выходами равно $(2^m)^{2^n} = 2^{m \cdot 2^n}$.

Определение 1.2.2. Для функции $f \in B_n$ набор $a = (a_1, \dots, a_n) \in B_n$ называется 1-набором или выполнимым набором, если $f(a) = 1$.

Определение 1.2.3. 1-множеством функции $f \in B_n$ называется множество $L_f^1 = \{(a_1, \dots, a_n) \mid f(a_1, \dots, a_n) = 1\}$.

Определение 1.2.4. Переменная x_i называется существенной для функции $f \in B_n$, если существует набор $(a_1, \dots, a_n)$ такой, что

$$f(a_1, \dots, a_{i-1}, 0, a_{i+1}, \dots, a_n) \neq f(a_1, \dots, a_{i-1}, 1, a_{i+1}, \dots, a_n)$$

Функция f может быть отождествлена со своим 1-множеством L_f^1 . Фактически, f является характеристической функцией своего 1-множества:

$$f(a) = X_{L_f^1}(a) = \begin{cases} 1, & a \in L_f^1 \\ 0, & a \notin L_f^1 \end{cases}$$

Перечислим булевы функции от 2 или меньше переменных:

- а) от 0 переменных: константы 0, 1: $1(x_1, \dots, x_n) = 1$; $0(x_1, \dots, x_n) = 0$;
- б) от 1 переменной: 4 функции $\{0, 1, x, \bar{x}\}$, $x, \bar{x}$ – литералы;
- с) от 2 переменных: 16 функций (дизъюнкция ($V, +$), конъюнкция ($\&, \wedge, \cdot$), сложение по модулю 2 ($\oplus$) и др.).

Фиксируя значения некоторых переменных функции f , можно получать ее подфункции.

Замечание 1.2.1 (разложение Шеннона). Пусть f – булева функция от n переменных, тогда для подфункций $g, h \in B_{n-1}$ таких, что

$$\begin{aligned} g(x_1, \dots, x_n) &= f(x_1, \dots, x_{n-1}, 0) \\ h(x_1, \dots, x_n) &= f(x_1, \dots, x_{n-1}, 1) \end{aligned}$$

выполняется

$$f = \bar{x}_n g + x_n h$$

Доказательство очевидно.

Замечание 1.2.2. Для определения подфункции поставим в соответствие каждой переменной x_i элемент $c(x_i)$ из $\{0, 1, n\}$ так, что

$$\begin{aligned} c(x_i) &= 0 \leftrightarrow x_i = 0; \\ c(x_i) &= 1 \leftrightarrow x_i = 1; \end{aligned}$$

$c(x_i) = n$, если x_i нефиксирована (свободна)

Таким образом, подфункция – это $f_c(x_1, \dots, x_n) = f(c(x_1), \dots, c(x_n))$. Следовательно, существует не более 3^n подфункций от n переменных.

Следствие 1.2.1. $\forall f \in B_n$ и $\forall i: 1 \leq i \leq n$ верно:

1. Разложение Шеннона по i -му аргументу:

$$f(x_1, \dots, x_n) = x_i f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) + \bar{x}_i f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n);$$

2. «Двойственное» разложение Шеннона:

$$f(x_1, \dots, x_n) = (x_i + f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n)) \cdot (\bar{x}_i + f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n));$$

3. Разложение Шеннона относительно $\oplus$:

$$f(x_1, \dots, x_n) = x_i f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) \oplus \bar{x}_i f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n);$$

Доказательство:

1. очевидно;
2. следует из принципа двойственности;
3. следует из того, что одно из слагаемых равно нулю. $\square$

1.3 Важные свойства булевых функций

Существует ряд свойств булевых функций, известных из курса дискретной математики: монотонность, симметричность и т.д. В этой главе рассмотрим некоторые из них.

Определение 1.3.1. Пусть $f \in B_n$. f монотонно возрастает (убывает) по i -му аргументу, если для любого $a \in B^n$:

$$f(a_1, \dots, a_{i-1}, 0, a_{i+1}, \dots, a_n) \leq f(a_1, \dots, a_{i-1}, 1, a_{i+1}, \dots, a_n)$$

$$(f(a_1, \dots, a_{i-1}, 0, a_{i+1}, \dots, a_n) \geq f(a_1, \dots, a_{i-1}, 1, a_{i+1}, \dots, a_n))$$

Если функция монотонно возрастает (убывает) по каждому своему аргументу, она называется *возрастающей (убывающей)*.

Теорема 1.3.1. Для того чтобы f была монотонно возрастающей (убывающей), необходимо и достаточно выполнения условия $\forall a, b \in B^n$

$$a \leq b \Rightarrow f(a) \leq f(b) (f(a) \geq f(b)).$$

Теорема 1.3.2. $f \in B_n$ является монотонно возрастающей (убывающей) по i -му аргументу тогда и только тогда, когда f может быть представлена в виде:

$$f = x_i g + h \quad (f = \bar{x}_i g + h) \text{ для } g \text{ и } h, \text{ независимых от } x_i:$$

Определение 1.3.2. Функция $f \in B_n$ называется *симметрической*, если любая перестановка π значений переменных не изменяет значения функции, т.е. $f(x_1, \dots, x_n) = f(x_{\pi(1)}, \dots, x_{\pi(n)})$.

Таким образом, функция является симметрической тогда и только тогда, когда она зависит только от количества единиц среди переменных, но не от их позиций. Следовательно, симметрические функции однозначно определяются их значениями на векторах:

$$v_i = (0, 0, \dots, 0, 1, \dots, 1) - \text{количество единиц равно } i \\ v_i \in B^n, i = \overline{0, n}$$

Замечание 1.3.1. Существует 2^{n+1} симметрических функций.

Примеры важных симметрических функций.

Пример 1.3.1. Функция Четности (Parity):

$$Par_n(x_1, \dots, x_n) = \bigoplus_{i=1}^n x_i.$$

Пример 1.3.2. Функция голосования (Majority):

$$\text{Maj}_n(x_1, \dots, x_n) = 1 \Leftrightarrow \sum_{i=1}^n x_i \geq \frac{n}{2}$$

Пример 1.3.3. Пороговая функция

$$T_k^n \in B_n, \quad 0 \leq k \leq n:$$

$$T_k^n(x_1, \dots, x_n) = 1 \Leftrightarrow \sum_{i=1}^n x_i \geq k.$$

Пример 1.3.4. Обратная пороговая функция

$$T_{\leq k}^n \in B_n, \quad 0 \leq k \leq n:$$

$$T_{\leq k}^n(x_1, \dots, x_n) = 1 \Leftrightarrow \sum_{i=1}^n x_i \leq k.$$

Пример 1.3.5. Интервальная функция:

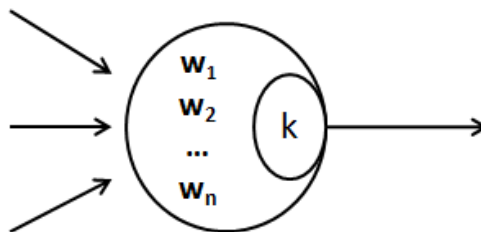
$$I_{k,m}^n(x_1, \dots, x_n) = 1 \Leftrightarrow k \leq \sum_{i=1}^n x_i \leq m.$$

Замечание 1.3.2. Каждая симметрическая функция может быть представлена, как дизъюнкция интервальных функций.

Пример 1.3.6. Взвешенная пороговая функция с весами $(w_1, \dots, w_n) \in R_n$ и порогом $k \in R$:

$$T_{w_1, \dots, w_n}(x_1, \dots, x_n) = 1 \Leftrightarrow \sum_{i=1}^n w_i x_i \geq k.$$

Замечание 1.3.3. Эта функция играет большую роль при моделировании нейронов и конструкции нейронных сетей.



В заключение хотелось бы отметить еще один важный класс булевых функций – *частично определенные функции*. Они возникают в том случае, когда некоторые входные наборы можно не рассматривать (т.е. они не могут возникнуть или реакция системы на них несущественна). В этом случае, кроме 1-множества и 0-множества, определяется N -множество – множество несущественных наборов.

Глава 2. Вычислимые функций и сложность вычислений (неформальное введение)

Прежде, чем говорить о представлениях булевых функций, нам необходимо ввести некоторые понятия теории вычислимости и теории сложности.

Наряду с понятием функции как таковой, часто говорят об алгоритме ее вычисления, который может быть как интуитивным, так и формализованным с помощью какой-либо модели вычислений.

В этом контексте обычно рассматривают два вопроса:

- Существует ли в принципе алгоритм вычисления той или иной функции?
- Насколько существующий алгоритм эффективен, какие ресурсы он требует для вычисления и как можно эти затраты минимизировать?

Первый вопрос относится к теории вычислимости, в то время как второй – к теории сложности. Одна из преследуемых нами целей – разбить класс всех функций в зависимости от их характеристик на классы. Из курса «Теория автоматов и грамматик» мы знакомы с иерархией Хомского:

$$Reg \subset КСЯ (CFL) \subset КЗЯ \subset Gram$$

где КСЯ – контекстно-свободные языки, а КЗЯ – контекстно-зависимые языки.

В следующих разделах введем другую иерархию функций, основанную на их сложностных характеристиках.

2.1 Вычислимость функций

Определение 2.1.1. *Под вычислимостью функции понимается существование алгоритма, удовлетворяющего ряду свойств (массовость, дискретность, конечность и т.д.), который за конечное число шагов по входу, кодирующему аргументы функции, выдает ответ, который содержит закодированное значение функции на этих аргументах. Такое представление вычислимости как существование абстрактного алгоритма, называется интуитивной вычислимостью.*

По-другому, понятие вычислимости ассоциируется с существованием алгоритма, воплощенного в конкретной модели вычислений (машины

Тьюринга, схемы, нормального алгоритма), кото-рая за конечное число шагов получает значение функции по ее аргументу, поданному на вход в закодированном виде.

Тезис Тьюринга-Черча. *Класс интуитивно вычислимых функций совпадает с классом функций, вычислимых машинами Тьюринга.*

В курсе теории алгоритмов о вычислимых функциях говорят как о классе. Также вводят и другие классы, образуя между ними иерархию. Например, множество вычислимых функций является собственным подмножеством множества рекурсивно-перечислимых функций, т.е. функций, для которых существует эффективный алгоритм перечисления всех аргументов, на которых функция принимает значение 1.

О более «практической» иерархии речь пойдет в следующем разделе.

2.2 Сложность функций

При реализации алгоритма вычисления той или иной функции «в железе», важны не только сам факт существования алгоритма ее вычисления, но и ресурсы, которые этот алгоритм использует, а также возможность их минимизации. Теория сложности изучает характеристики алгоритмов с точки зрения необходимых ресурсов, возможность их оптимизации. Одна из целей теории сложности – разбиение функций в зависимости от сложности алгоритмов их вычисления на различные классы. Обычно рассматривают два типа вычислительных ресурсов – время вычисления и память, использованная при вычислении. Таким образом, говорят соответственно о временной и пространственной сложностях. Опираясь на тезис Тьюринга-Черча, а также на полиномиальную эквивалентность большинства моделей вычислений, будем использовать в этой главе машину Тьюринга как базовую модель для построения вычислительных алгоритмов. Дадим ее формальное определение.

Машина Тьюринга (МТ) – модель вычислений, состоящая из устройства управления с конечным множеством состояний, имеющая входную ленту, а также одну или несколько рабочих лент. Для простоты рассмотрим упрощенную модель машины Тьюринга с одной лентой, которую можно представить как шестерку:

$$M = \langle \Sigma, \{0,1\}, Q, \delta, q_s, q_0, q_1 \rangle,$$

где Σ – алфавит рабочей ленты; $\{0,1\}$ – входной алфавит (т.к. рассматриваются булевы функции); Q – конечное множество состояний, в который входят q_s – начальное состояние, q_1 – финальное принимающее состояние, q_0 – финальное отвергающее состояние; $\delta: Q \times \Sigma \times \{0,1\} \rightarrow Q \times \Sigma \times \{L, S, R\}$ – функция переходов.

На каждом шаге работы машина Тьюринга по текущим символам входной и рабочих лент и текущему состоянию определяет, согласно функции переходов, новое состояние, символы, записываемый на рабочие ленты, а также сдвиг считывающей головки по рабочей ленте.

Определение 2.2.1. *Машина Тьюринга M вычисляет булеву функцию f , если на входном наборе $x = (x_1, \dots, x_n)$, записанном на входную ленту, M за конечное число шагов останавливается в состоянии q_0 для всех x : $f(x) = 0$ и в состоянии q_1 для всех x : $f(x) = 1$.*

Временной сложностью машины Тьюринга M является количество шагов ее работы. Очевидно, что временная сложность будет зависеть от размера входных данных, поданных на входную ленту M . Обозначим временную сложность для МТ M через $time_M(n)$, как максимальное количество шагов МТ для всех входных слов длины не более n .

Сложностью по памяти (пространственной сложностью) машины Тьюринга принято считать количество использованных ячеек рабочих лент. В данном случае МТ имеет одну входную ленту, с которой осуществляется лишь чтение и запрещена запись, и несколько рабочих лент.

Определение 2.2.2. *Класс P – класс всех функций, вычислимых на машинах Тьюринга с полиномиальной от размера входа временной сложностью. Формально:*

$$P = \{f | \exists MT M: M \text{ вычисляет } f \ \& \ time_M(n) = O(poly(n))\},$$

где $poly(n)$ - некоторый полином постоянной степени.

Существуют такие функции, для которых не удастся получить полиномиальной оценки временной сложности работы алгоритма вычисления. Некоторые из этих функций могут быть вычислены за полиномиальное время на *недетерминированных машинах Тьюринга* (НМТ), для которых функции переходов при одинаковой левой части могут иметь

разные правые части. Таким образом, процесс вычисления будет иметь уже не линейную, а древовидную структуру.

Определение 2.2.3. Недетерминированная машина Тьюринга M' вычисляет функцию f , если для любого входного набора x такого что $f(x) = 1$, существует хотя бы один путь вычислений, при котором M' останавливается в q_1 , а на всех входных наборах x таких, что $f(x) = 0$, не существует ни одного пути вычисления, приводящего в q_1 .

Обозначим временную сложность для недетерминированной машины Тьюринга M' как $ntime_{M'}(n)$.

Определение 2.2.4. Класс **NP** – класс всех функций, вычислимых с помощью машин Тьюринга с полиномиальной от размера входа временной сложностью. Формально:

$$\mathbf{NP} = \{f | \exists \text{НМТ } M: M \text{ вычисляет } f \ \& \ ntime_M(n) = O(poly(n))\},$$

Очевидно, что класс **P** является подмножеством класса **NP**. Вопрос же о том, равны ли эти классы, т.е. существует ли для каждой задачи из **NP** детерминированный алгоритм, решающий эту задачу за полиномиальное время, остается открытым.

Функции (в частности, из класса **NP**), для которых до сих пор не найдено полиномиального (эффективного) алгоритма, назовем труднорешаемыми.

2.3 Примеры труднорешаемых задач

На первый взгляд функции распознавания, определяющие определенное свойство, например, принадлежность входного слова языку, и имеющие лишь 2 возможных выходных значения, и вычислительные функции, имеющие в качестве результата, возможно, более 2-х значений, представляют принципиально различные классы. Однако, с точки зрения сложности вычисления, эти классы «одинаковы». Любую вычислительную функцию можно вычислить через соответствующие функции распознавания, сложность вычисления которых соответствует сложности вычисления исходной функции. Поэтому, в дальнейшем, будем рассматривать задачи распознавания. Поскольку любую задачу распознавания можно представить в виде булевой функции, а ее в свою очередь в виде некоторого формального

языка, будем использовать термины «задача», «функция» и «язык» равноправно, также как и «алгоритм решения задачи», «алгоритм вычисления функции», «алгоритм распознавания языка».

Приведем примеры задач из класса **NP** для которых не найдено эффективного, то есть полиномиального алгоритма решения.

SAT (Выполнимость). Дан набор $C = \{c_1, c_2, \dots, c_m\}$ дизъюнкций на множестве булевых переменных $x = (x_1, \dots, x_n)$. Существует ли такой набор значений переменных из x , при котором выполняются все дизъюнкции из C ?

3-SAT (3-Выполнимость). Ограниченный вариант задачи SAT, где в каждую дизъюнкцию из C входят ровно три литерала.

Гамильтонов цикл. Дан граф $G = (V, E)$. Верно ли, что G содержит гамильтонов цикл, т.е. такую последовательность $\langle v_1, v_2, \dots, v_n \rangle$ различных вершин графа G , что $n = |V|$, $\{v_n, v_1\} \in E$ и $\{v_i, v_{i+1}\} \in E$ для всех $i, 1 \leq i \leq n$?

Разбиение. Заданы конечное множество A и целочисленный вес $s(a)$ для каждого $a \in A$. Существует ли подмножество $A' \subset A$ такое, что

$$\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$$

2.4 Полиномиальная сводимость и **NP**-полнота

Существуют задачи, для которых исследование их сложностных характеристик достаточно трудоемко. Иногда отношения сводимости позволяют упростить процесс получения оценок сложности и установления для конкретной задачи факта принадлежности какому-либо классу.

Определение 2.4.1. Функция $f_1: \{0,1\}^n \rightarrow \{0,1\}$ полиномиально сводится к функции $f_2: \{0,1\}^m \rightarrow \{0,1\}$, если существует функция $g: \{0,1\}^n \rightarrow \{0,1\}^m$, для которой выполняется два условия:

- g может быть вычислена детерминировано за полиномиальное время,
- Для любого набора $x = (x_1, \dots, x_n)$ $f_1(x) = 1$ тогда и только тогда, когда $f_2(g(x)) = 1$.

Иными словами, задача T_1 полиномиально сводится к задаче T_2 , если существует такая полиномиально вычислимая функция g , что S является решением задачи T_1 когда $g(S)$ является решением задачи T_2 .

Приведем без доказательства следующие свойства полиномиальной сводимости, которые следуют из ее определения.

Теорема 2.4.1. *Если функция f_1 полиномиально сводится к функции f_2 , а f_2 полиномиально сводится к f_3 , то f_1 полиномиально сводится к f_3 .*

Теорема 2.4.2. *Если f_1 полиномиально сводится к f_2 , а f_2 принадлежит некоторому классу (например, **P** или **NP**), то и f_1 принадлежит этому классу.*

Последняя теорема дает принципиально новый способ исследования сложностного класса **NP**.

Определение 2.4.2. *Задача T называется **NP-трудной**, если любая задача $T' \in \mathbf{NP}$ полиномиально сводится к T .*

Определение 2.4.3. *Задача T называется **NP-полной**, если*

- $T' \in \mathbf{NP}$,
- T является **NP-трудной**.

Значимость **NP-полных** задач заключается в том, что если найдется детерминированный полиномиальный алгоритм хотя бы для одной из них, то это будет означать совпадение классов **P** и **NP**.

Одной из первых задач, для которых **NP-полнота** была доказана по определению, была **SAT**. Рассмотрим основные пункты доказательства **NP-полноты** данной задачи.

Теорема 2.4.3 (Кука [2]) *Задача **SAT** является **NP-полной**.*

Доказательство: (набросок).

Очевидно, что **SAT** лежит в классе **NP**, поскольку для недетерминированно выбранного набора значений переменных можно за полиномиальное время проверить, что все дизъюнкции истинны.

Покажем, что произвольная задача из класса **NP** полиномиально сводится к задаче **SAT**. По определению класса **NP**, для любой задачи из **NP**

существует недетерминированный алгоритм ее решения (недетерминированная машина Тьюринга, решающая данную задачу). Используя такое «единое» представление задач из **NP**, можно специализировать полиномиальную сводимость в терминах НМТ и доказать теорему.

Рассмотрим недетерминированную машину Тьюринга M_L , распознающую язык L за полиномиальное время. Функцию сводимости f_L задачи L к задаче SAT будем формулировать в терминах НМТ M_L , чтобы она являлась отображением «поведения» машины M_L в наборы дизъюнкций $f_L(x)$. Т.е. функция f_L будет обладать тем свойством, что если слово x принадлежит языку L , то для набора дизъюнкций $f_L(x)$ существует выполняющий набор значений.

Если входное слово x принимается машиной Тьюринга M_L , то существует соответствующее вычисление осуществляемое не более чем за $p(n)$ тактов для некоторого полинома p , где $n=|x|$. В таком вычислении будут участвовать не более чем $p(n)$ ячеек, поскольку за один шаг M_L сдвигается не более чем на одну ячейку ленты. Соответствующее вычисление можно описать, используя полиномиально ограниченное число булевских переменных.

Введем три типа булевских переменных, которые будем использовать в дальнейшем.

- $Q[i, k]$ – в момент времени i машина M_L находится в состоянии q_k ,
- $H[i, j]$ – в момент времени i машина M_L просматривает ячейку с номером j ,
- $S[i, j, k]$ – в момент времени i в ячейке j записан символ s_k .

Вычисление машины M_L очевидно порождает на этих переменных набор значений истинности. С другой стороны, произвольный набор значений истинности не обязательно соответствует какому-нибудь вычислению, а уж тем более принимающему слово x . Таким образом, описание функции f_L осуществляется с помощью построения такого набора дизъюнкций из указанных переменных, что набор значений истинности будет выполняющим тогда и только тогда, когда этот набор значений кодирует некоторое принимающее вычисление на входе x , причем стадия проверки этого вычисления выполняется не более чем за $p(n)$ шагов.

Т.е. нам необходима эквивалентность следующих утверждений:

- $x \in L$,

- на входе x машина Тьюринга M_L останавливается в принимающем состоянии не более чем за $p(n)$ шагов,

- существует выполняющее задание значений истинности для набора дизъюнкций задачи $f_L(x)$.

Дизъюнкции для описания $f_L(x)$ можно подразделить на 6 групп, каждая из которых будет налагать ограничение определенного типа на выполняющий набор значений истинности.

- $G1$ – в любой момент времени i M_L находится ровно в одном состоянии,
- $G2$ – в любой момент времени i головка просматривает ровно одну ячейку,
- $G3$ – в любой момент времени i каждая ячейка содержит ровно один символ алфавита M_L ,
- $G4$ – в момент времени 0 вычисление находится в исходной конфигурации стадии проверки при входе x ,
- $G5$ – не позднее чем через $p(n)$ шагов M_L переходит в принимающее состояние (и значит, принимает x),
- $G6$ – Для любого момента времени $i, 0 \leq i \leq p(n)$, конфигурация программы M_L в момент времени $i + 1$ получается из конфигурации в момент времени i одноразовым применением функции перехода.

Очевидно, что если все 6 групп дизъюнкций осуществляют поставленные цели, то выполняющий набор значений истинности обязан соответствовать принимающему вычислению на входе x .

Укажем способ построения групп дизъюнкций, осуществляющих эти цели:

$G1$:

$$\{Q[i, 0], Q[i, 1], \dots, Q[i, r]\}, 0 \leq i \leq p(n), \\ \{\overline{Q[l, j]}, \overline{Q[l, j']}\}, 0 \leq i \leq p(n), 0 \leq j < j' \leq r$$

$G2$:

$$\{H[i, -p(n)], H[i, -p(n) + 1], \dots, H[i, p(n) + 1]\}, 0 \leq i \leq p(n), \\ \{\overline{H[l, j]}, \overline{H[l, j']}\}, 0 \leq i \leq p(n), -p(n) \leq j \leq j' \leq p(n) + 1$$

$G3$:

$$\{S[i, j, 0], S[i, j, 1], \dots, S[i, j, v]\}, 0 \leq i \leq p(n), -p(n) \leq j \leq p(n) + 1, \\ \{S[l, j, k], S[l, j, k']\}, 0 \leq i \leq p(n), -p(n) \leq j \leq p(n) + 1, 0 \leq k < k' \leq v$$

G4:

$$\{Q[0,0]\}, \{H[0,1]\}, \{S[0,0,0]\}, \\ \{S[0,1, k_1]\}, \{S[0,2, k_2]\}, \dots, \{S[0, n, k_n]\}, \\ \{S[0, n + 1, 0]\}, \{S[0, n + 2, 0]\}, \dots, \{S[0, p(n) + 1, 0]\}, x = s_{k_1} s_{k_2} \dots s_{k_n}$$

G5:

$$\{Q[p(n), 1]\}$$

Несколько сложнее выглядит описание последней группы дизъюнкций G6, которая гарантирует, что каждая следующая конфигурация машины получается из предыдущей в результате применения одной команды M_L . Эта группа состоит из двух подгрупп дизъюнкций. Одна из них гарантирует, что если головка в момент i не просматривает ячейку с номером j , то символ в ячейке j на данном шаге не изменится. Вторая гарантирует, что перестройка одной машинной конфигурации в следующую происходит согласно функции перехода машины M_L .

Таким образом, мы показали, как построить группы дизъюнкций. Если $x \in L$, то у M_L на входе x есть принимающее вычисление длины не более $p(n)$, и при заданной интерпретации это вычисление дает набор значений истинности для выполнения всех дизъюнкций из G1–G6. Таким образом, для $f_L(x)$ имеется выполняющий набор значений истинности тогда и только тогда когда $x \in L$.

Предоставим читателю доказательство полиномиальности построенной функции f_L , т.е. вычислимости функции алгоритмом с полиномиальной временной сложностью. $\square$

Доказательство **NP**-полноты не обязательно нужно проводить по определению. Для того, чтобы показать **NP**-полноту некоторой задачи T , необходимо доказать принадлежность T классу **NP**, а затем показать, что какая-нибудь из известных **NP**-полных задач может быть полиномиально сведена к T .

Теорема 2.4.4. 3-SAT является **NP**-полной

Доказательство: (набросок)

Недетерминированный алгоритм может угадать набор значений истинности для дизъюнкций и проверить его за полиномиальное время. Поэтому очевидно, что $3\text{-SAT} \in \mathbf{NP}$.

Сведем задачу SAT к 3-SAT для доказательства ее $\mathbf{NP}$ -полноты. Для этого построим набор C' дизъюнкций, каждая из которых включает в себя ровно 3 литерала, и докажем, что выполнимость построенного набора эквивалентна выполнимости исходного набора C для задачи SAT .

C' строится заменой каждой конъюнкции c_j эквивалентным набором «трехлитеральных» дизъюнкций на множестве переменных U задачи SAT и на множестве U'_j дополнительных переменных, которые будут использоваться в соответствующей дизъюнкции c_j .

Покажем, как из c_j построить C'_j и U'_j . Пусть c_j задана множеством $\{z_1, \dots, z_k\}$, где z_i – литералы на множестве U . Способ построения C'_j и U'_j зависит от числа k .

$k = 1$. Тогда $U'_j = \{y_j^1, y_j^2\}$,

$C'_j = \{\{z_1, y_j^1, y_j^2\}, \{z_1, y_j^1, \bar{y}_j^2\}, \{z_1, \bar{y}_j^1, y_j^2\}, \{z_1, \bar{y}_j^1, \bar{y}_j^2\}\}$,

$k = 2$. Тогда $U'_j = \{y_j^1\}$, $C'_j = \{\{z_1, z_2, y_j^1\}, \{z_1, z_2, \bar{y}_j^1\}\}$,

$k = 3$. Тогда $U'_j = \emptyset$, $C'_j = \{c_j\}$,

$k > 3$. Тогда $U'_j = \{y_j^i: 1 \leq i \leq k - 3\}$,

$C'_j = \{\{z_1, z_2, y_j^i\} \cup \{\bar{y}_j^i, z_{i+2}, y_j^{i+1}\}: 1 \leq i \leq k - 4\} \cup \{\{\bar{y}_j^{k-3}, z_{k-1}, z_k\}\}$

Используя построенные формулы, можно показать, что набор дизъюнкций C' выполним тогда и только тогда, когда выполним C . Доказательство полиномиальности построенной сводимости предоставляется читателю (воспользоваться тем, что количество дизъюнкций с тремя литералами ограничено полиномом от mn). $\square$

Глава 3. Представления функций

3.1 Обзор подходов

После введения понятия сложности вычисления функций, приходим к вопросу выбора представления функций. Представление должно удовлетворять следующим требованиям:

1. Быть достаточно коротким и эффективным.
2. Позволять манипулировать и вычислять функции.
3. Визуализировать определенные свойства функций.
4. Подсказывать идеи для технологий реализации.

Рассмотрим известные представления с учетом указанных выше требований.

1. Таблицы истинности.

Плюсы этого представления в быстром вычислении и выполнении бинарных операций. Однако, хотя сложность вычисления функции по таблице истинности линейна относительно размера таблицы, она экспоненциальна относительно количества переменных, а размерность таблицы экспоненциальна относительно числа переменных. Это в свою очередь является существенным недостатком данного представления.

2. Дизъюнктивные/конъюнктивные нормальные формы (ДНФ/КНФ), полином Жегалкина (без отрицаний) – двухуровневые нормальные формы.

Введем ряд дополнительных свойств представлений, которые понадобятся в дальнейшем:

Определение 3.1.1. *Представление называется универсальным, если для любой булевой функции существует представление такого типа.*

Определение 3.1.2. *Представление называется совершенным, если для любой булевой функции существует единственное представление такого типа.*

ДНФ и КНФ в общем случае не являются совершенным представлением, в отличие от совершенных ДНФ и КНФ. В случае совершенных

представлений для определения эквивалентности функций, достаточно проверить идентичность их представлений.

Определение 3.1.3. Пусть D – класс представления функций. Язык EQU_D соответствует парам (C_1, C_2) представлений в классе D , которые эквивалентны.

Определение 3.1.4. Язык $INEQU_{CNF}$ соответствует парам КНФ (C_1, C_2) , которые неэквивалентны.

Теорема 3.1.1. Задача $INEQU_{CNF}$ **NP**-полна.

Доказательство: 3-SAT полиномиально сводится к $INEQU_{CNF}$. Действительно, если C, C_1, C_2 – КНФ и $C_1 = C, C_2 = 0$, то $C \in 3\text{-SAT}$ тогда и только тогда, когда $(C_1, C_2) \in INEQU_{CNF}$. $\square$

Для любого класса сложности $\mathbf{Q}$ можно определить «обратный» класс сложности $\mathbf{coQ}$ следующим образом. Если язык $L \in \mathbf{Q}$, то для любого $x \in L$ есть возможность проверки, что x находится в L в пределах ресурсов $\mathbf{Q}$. Если язык $L \in \mathbf{coQ}$, то для любого $x \notin L$ есть возможность проверки, что x не находится в L в пределах ресурсов $\mathbf{Q}$.

Следствие 3.1.1. Эквивалентность КНФ **coNP**-полна.

Совершенные ДНФ и КНФ имеют большую длину: сложность ДНФ (количество входящих в нее литералов) для СДНФ максимальна среди всех эквивалентных ДНФ.

Теорема 3.1.2. Полином Жегалкина – совершенное представление булевых функций.

Доказательство: Любая ДНФ представима через полином Жегалкина, следовательно любая булева функция представима в виде полинома Жегалкина. С другой стороны, количество полиномов Жегалкина равно 2^{2^N} , т.е. количеству всех булевых функций, что и доказывает теорему. $\square$

Теорема 3.1.3. Функция $R(x_1, \dots, x_n) = x_1 + x_2 + \dots + x_n =$

$$\oplus_{I \subseteq \{1, \dots, n\}, I \neq \emptyset} m_I = P(x_1, \dots, x_n), \text{ где } m_I = x_{i_1} \dots x_{i_s} \text{ для } I = \{i_1, \dots, i_s\}$$

Доказательство: $P(0, \dots, 0) = 0$.

Пусть в наборе ровно l единиц. Тогда в P количество мономов, равных 1 и содержащих k переменных, равно количеству сочетаний из l по k : $\binom{l}{k}$, $1 \leq k \leq l$. Следовательно, количество единичных мономов $\sum_{k=1}^l \binom{l}{k} = \sum_{k=1}^n \binom{l}{k} - \binom{l}{0} = 2^l - 1$, то есть это число нечетно.

Следовательно, $P(x_1, \dots, x_n) = 1 = R(x_1, \dots, x_n)$. $\square$

Следствие 3.1.2. Функция $x_1 + \dots + x_n$ – экспоненциально сложна в представлении через полином Жегалкина.

3. СФЭ.

Переход от двухуровневых к многоуровневым формам позволяет найти более компактное представление булевых функций. Пусть $\Omega = \{w_i \in B_n, i \in I\}$ – множество базовых функций (называется базисом, если нельзя удалить ни одной функции из Ω без уменьшения замыкания Ω).

Определение 3.1.5. Ω -СФЭ S от n переменных – ациклический граф с вершинами 2-х типов:

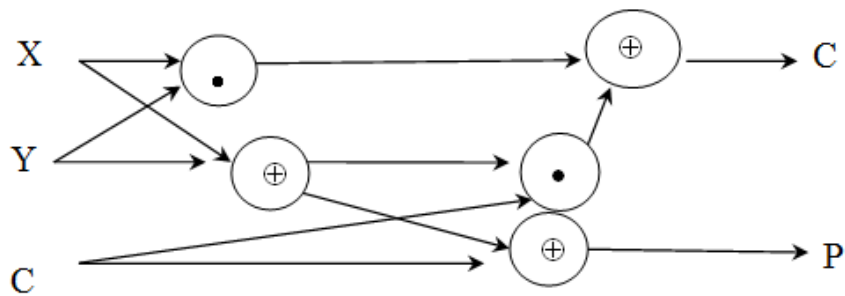
- входные вершины (без входящих дуг), помеченные 0, 1 или $x_i, i = 1, \dots, n$;
- функциональные вершины – вершины, помеченные w_i , имеют n_i входящих дуг: $w_i \in \Omega, i \in I$.

Каждая вершина v представляет булеву функцию, соответствующую следующим индуктивным правилам:

1. Если v помечена 0(1), то $f_v(x_1, \dots, x_n) = 0(1)$;
2. Если v помечена x_i , то $f_v(x_1, \dots, x_n) = x_i$;
3. Если v помечена w_i и имеет n_i входящих дуг, выходящих из $v_1 \dots v_{n_i}$: $f_v(x_1, \dots, x_n) = w_i(f_{v_1}(x), \dots, f_{v_{n_i}}(x))$.

Обозначив m узлов $v_1, \dots, v_m$ как выходные узлы СФЭ, получим схему, представляющую функцию $f_s(x) = (f_{v_1}(x), \dots, f_{v_m}(x))$.

Пример 3.1.1. Полный сумматор – хорошо известная базовая компонента в построении чипов. Эта схема вычисляет $x+y+c$ для 3 входных битов x, y, c (c – бит переноса, $\Omega = \{\oplus, \cdot, \neg\}$ – стандартный базис):



Определение 3.1.6. Глубина СФЭ – число функциональных уровней в СФЭ (соответствует временным затратам).

Определение 3.1.7. Базис полон, если Ω -СФЭ соответствует универсальному представлению, то есть любая булева формула может быть представлена в Ω -СФЭ.

Пример 3.1.2.

1. $\{+, \cdot, \neg\}$ (стандартный базис) – полон (ДНФ, КНФ);
2. $\{\oplus, \cdot\}$ – полон;
3. $\{+, \cdot\}$ – неполон, так как он генерирует лишь монотонно возрастающие функции.

Даже для стандартного базиса представление булевой функции неоднозначно. Интересно представление функций в СФЭ с небольшим количеством вершин.

Определение 3.1.8. Ω -СФЭ-сложность булевой функции f – это минимальное число вершин Ω -СФЭ, представляющей f .

ДНФ и КНФ – специальные формы СФЭ, поэтому СФЭ-сложность в стандартном базисе не больше ДНФ-(КНФ)-сложности. С другой стороны, в большинстве случаев СФЭ-сложность меньше ДНФ-(КНФ)-сложности.

СФЭ-модель по-прежнему имеет недостаток: неприемлемая временная сложность проверки эквивалентности схем.

Теорема 3.1.4. *Задача проверки эквивалентности СФЭ над стандартным базисом $\mathbf{NP}$ -полна.*

Доказательство: Сформулированная проблема лежит в классе $\mathbf{NP}$, EQU_{DNF} полиномиально сводится к $EQU_{\text{СФЭ}}$. $\square$

4. Формулы как СФЭ.

В СФЭ экономия происходит, в частности, в связи с тем, что выход элемента может подаваться на вход нескольких элементов. То есть, вершины могут иметь несколько выходных дуг. Именно это свойство часто позволяет построить компактное представление в виде СФЭ. Но это приводит к трудности решения различных задач.

Определение 3.1.9. *Для Ω -базиса Ω -формула – это Ω -СФЭ, степень исхода каждой вершины которой равна 1.*

Следствие 3.1.3. *Ω -СФЭ – формула $\Leftrightarrow$ она состоит из деревьев.*

Очевидно, существует взаимно-однозначное соотношение между Ω -формулами и булевыми формулами.

Анализ представления функции в виде Ω -формулы намного проще, чем в виде СФЭ. Например, проще определяются нижние оценки сложности конкретных функций. Но задача проверки эквивалентности булевых формул $\mathbf{NP}$ -полна.

5. Бинарные диаграммы решений.

Определение 3.1.10. *Бинарное дерево решений (б.д.р.) – дерево, в котором:*

- *внутренние вершины помечены переменными x_i и имеют ровно 2 выходных дуги.*
- *листья помечены 0 и 1.*

Замечание 3.1.1. *Без потери общности можно считать, что каждая переменная читается не более одного раза, то есть на любом пути от корня до листа встречается не более одного раза.*

Пример 3.1.3. $f = (x_1 \oplus x_2)(x_3 \oplus x_4)$

а) Полное дерево.

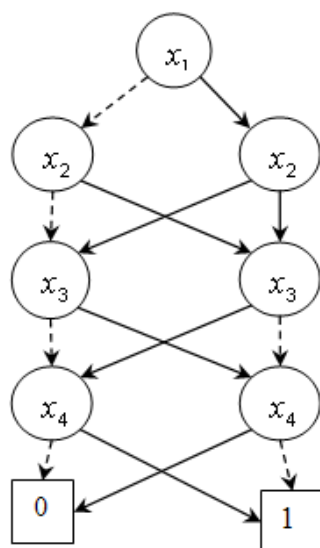
б) Неполное дерево.

в) Изменение порядка чтения переменных существенно влияет на размер б.д.р: порядок x_1, x_2, x_3, x_4 позволяет получить более экономное представление f , чем порядок x_1, x_3, x_2, x_4 .

Определение 3.1.11. Ориентированный ациклический граф с двумя стоками, помеченными 0 и 1, и внутренними вершинами, помеченными x_i и имеющими две выходных дуги, помеченные 0 и 1, имеет разные названия: бинарные диаграмма решений (Lee, 1959), ветвящаяся программа (branching program), бинарная программа (BP) (Mask, 1976).

Определение 3.1.12. Размер (сложность) бинарной программы – количество ее вершин.

Пример 3.1.4. $x_1 \oplus x_2 \oplus x_3 \oplus x_4$



Определение 3.1.13. Пусть P – бинарная программа. Тогда:

1. k -й уровень P – это множество всех узлов, которые могут быть достигнуты из корня путем длины $k - 1$ (таким образом, никакая вершина не может принадлежать разным уровням);

2. максимальная мощность $w(P)$ уровня BP P называется шириной P

Определение 3.1.14. Пусть P – бинарная программа. Тогда P :

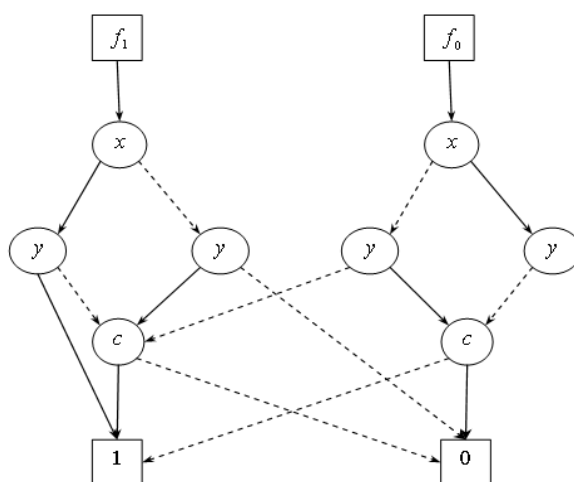
1. имеет ограниченную ширину k , если каждый уровень имеет ширину не более k (обозначается BP_{w-k});
2. синхронная, если для каждой вершины $v \in P$ все пути от корня до v имеют одинаковую длину (sBP);
3. забывающая, если P – синхронная и все узлы одного уровня помечены одной и той же переменной (oBP);
4. k -раз читающая, если каждая переменная встречается на любом пути не более k раз (BP_k).

Замечание 3.1.2. Можно совмещать введенные обозначения, например, $sBP4_{w3}$ – синхронная 4-раз читающая BP ширины 3.

В отличие от бинарных деревьев решений, в бинарных программах повторное чтение переменных может дать бинарную программу меньшего размера (вершину v , в которой переменная читается повторно, не всегда возможно удалить, например, если v можно достичь различными путями).

Особый интерес представляет один-раз читающие бинарные программы ($BP1$). Например, бинарное дерево решений – $BP1$. Любой путь в $BP1$ от корня до финальной вершины – это путь вычисления.

Бинарные программы можно рассматривать и для представления функций из $B_{n,m}$ с несколькими выходами. При этом выделяются m начальных вершин.



Часто важно уметь переводить одно представление булевой функции в другое эффективно.

1. *От бинарной программы можно перейти к ДНФ-представлению.*

Строим моном для каждого пути. Затем рассматриваем дизъюнкцию мономов.

2. *От бинарной программы можно перейти к -СФЭ (Ω -полный базис).*

Рассмотрим булеву функцию $sel(x, y, z) = xy + \bar{x}z$.

- а) Заменим каждую вершину, помеченную x_i , на sel -вершину с одним из входов, равным x_i , двигаясь от финальных вершин.
- б) Изменим направление всех дуг.
- в) Заменяем все sel -вершины на -СФЭ.

3.2 Ограниченные классы бинарных программ.

Проблема 3-SAT/3-появление – это язык, состоящий из выполнимых КНФ C таких, что каждая дизъюнкция C содержит не более 3 литералов и каждая переменная встречается не более 3 раз.

Лемма 3.2.1. *Проблема 3-SAT/3-появление NP-полна.*

Доказательство:

- 1) Очевидно, что проблема 3-SAT/3-появление лежит в классе **NP**.
- 2) Покажем, что 3-SAT сводится к проблеме 3-SAT/3-появление. Пусть КНФ $C = \prod_{i=1}^n c_i$, где

$$c_i = x_{i_1}^{\alpha_{i_1}} + x_{i_2}^{\alpha_{i_2}} + x_{i_3}^{\alpha_{i_3}}, \alpha_{i_j} \in \{0,1\}, 1 \leq j \leq 3$$

Построим новую КНФ C' . Каждую переменную x_i при j -м появлении заменяем на новую переменную x_{ij} . Добавляем к полученной КНФ C' конъюнкции следующего вида:

$$(x_{i_1} + \bar{x}_{i_2})(x_{i_2} + \bar{x}_{i_3}) \dots (x_{i_k} + \bar{x}_{i_1})$$

где k_i – количество появлений x_i в C , если $k_i \geq 2$.

Последняя конъюнкция выполнима тогда и только тогда, когда $x_{ij} = x_{im}, 1 \leq j \leq k_i, 1 \leq m \leq k_i$.

КНФ C выполнима тогда и только тогда когда выполнима КНФ C' . Так как сводимость может быть осуществлена за полиномиальное время, теорема доказана. $\square$

Пример 3.2.1. Пусть $C = (x_1 + \overline{x_2} + x_4)(x_2 + x_3 + \overline{x_4})$. Тогда
 $C = (x_{11} + \overline{x_{21}} + x_{41})(x_{22} + x_{31} + \overline{x_{42}})(x_{21} + \overline{x_{22}})(x_{22} + \overline{x_{21}})(x_{41} + \overline{x_{42}})(x_{42} + \overline{x_{41}})$.

Пусть x – 1-набор C . Тогда если $x_{ij} = x_i$, то получаем 1-набор для C' .

Пусть x – 1-набор C' . Тогда если $x_{ij} = x_i$ (для любого i все x_{ij} совпадают), то получаем 1-набор для C .

Проблема SAT_X . Пусть P – представление типа X . Выполнимо ли P ?

Теорема 3.2.1. Согласно обозначениям замечания 2.1.2, проблемы $SAT_{sBP3_{w2}}$ и $SAT_{oBP3_{w2}}$ – **NP**-полны.

Доказательство:

1) 3-SAT/3-появление полиномиально сводится к $SAT_{sBP3_{w2}}$.

Пусть КНФ $C = \prod_{i=1}^n c_i$, где

$$c_i = x_{i_1}^{\alpha_{i_1}} + x_{i_1}^{\alpha_{i_1}} + x_{i_1}^{\alpha_{i_1}}, \alpha_{ij} \in \{0,1\}, 1 \leq j \leq 3$$

и $C \in 3\text{-SAT/3-появление}$.

Функция, соответствующая литералу, представляется $BP1$ с корнем и двумя стоками. Если отождествить 0-стоки соответствующих программ с корнем «следующего литерала», получим $BP1_{w1}$, реализующую дизъюнкцию.

Если отождествить 1-стоки соответствующих программ с корнем «следующей дизъюнкции», получим $BP3_{w1}$, от которой можно перейти к синхронной $sBP3_{w2}$ введением фиктивных узлов, из которых обе дуги ведут в одну вершину. При этом на одном уровне с нефинальными вершинами будут находиться и 0-стоки.

2) Продолжаем построение.

Каждое j -е появление переменной x_i заменим новой переменной x_{ij} . Получаем $oBP1_{w2} P_1$.

Затем добавим блоки, соответствующие проверке $x_{i1} = x_{i2} = x_{i3}$. Эти блоки можно представить $oBP1_{w3}$: две ветви ведут в 1-сток (для значений переменных, равных 0 и 1), а одна ветвь, состоящая из фиктивных вершин, ведет в 0-сток.

Полученные блоки соединяем в $oBP1_{w3}$ P_2 следующим образом. От 0-стока дуги направляем в среднюю фиктивную линию, а 1-сток отождествим с корнем следующего блока.

Естественным образом соединяя P_1 и P_2 , получим искомую $oBP2_{w3}$. $\square$

Следствие 3.2.1. $SAT_{BP2}, SAT_{oBP}, SAT_{sBP}, SAT_{BP} - \mathbf{NP}$ -полны.
 $EQU_{BP2}, EQU_{oBP}, EQU_{sBP}, EQU_{BP} - \mathbf{coNP}$ -полны.

Проблема. Общий путь BP1 (comm-path-BP1). Существуют ли значения переменных BP1 P_1 и P_2 такие, что на них P_1 и P_2 вычисляют 1.

Теорема 3.2.2. *comm-path-BP1 – $\mathbf{NP}$ -полная проблема.*

Действительно, в доказательстве второй части теоремы 2.2.1 BP2 состоит из BP1, которые должны последовательно вычислять 1.

Лемма 3.2.2. $SAT_{BP1} \in \mathbf{P}$.

Алгоритм осуществляет проход в ширину для проверки достижимости из корня 1-стока.

Проблема $$ – SY N_X .* Пусть P_1, P_2 – представления типа X функций f_1 и f_2 . Найти представление типа X для $f_1 f_2$.

В доказательстве теоремы 3.2.1 показано, что $*-SY N_{oBP}, *-SY N_{sBP}, *-SY N_{BP}$ для $*$ $\in \{\wedge, +\}$ решается легко.

Теорема 3.2.3. $*-SY N_{BP1} - \mathbf{NP}$ -трудная проблема.

Проблема *comm-path-BP1* может быть решена через решение проблемы $\wedge -SY N_{BP1}$.

Следствие 3.2.2. $+ -SY N_{BP1}, \oplus -SY N_{BP1} - \mathbf{NP}$ -полные проблемы.

3.3 Требование к структурам данных в формальной верификации схем

Итак, существует несколько типов представления булевых функций:

1. Задание значений функции (таблица истинности);
2. Метод вычисления булевой функции (СФЭ);
3. Описание схемы определения значений функции.

Эти представления имеют существенные различия:

1. *Размер*: таблица истинности и ДНФ $(x_1 + x_2 + \dots + x_n)$, с другой стороны $x_1 + x_2 + \dots + x_n$, представленная в виде полинома Жегалкина, имеет слагаемыми всевозможные мономы;
2. *Ресурсы*, требующиеся для вычисления функции: например, время (для таблицы истинности значение функции получается сразу, для формулы требуется вычисление).
3. *Ресурсы*, требуемые для представления результата булевых операций над двумя функциями $(f * g)$. (Для СФЭ быстро, для $BP1 - NP$ -сложно).
4. *Сложность определения свойств функции* (например, «функция – константа?»): просто для $BP1$; NP -сложно для КНФ: $f \notin SAT \Leftrightarrow f = const, f(0,0, \dots, 0) = 0$.
5. *Сложность определения фиктивности переменной* (полином Жегалкина: переменная существенна $\Leftrightarrow$ входит в полином Жегалкина, для КНФ – NP -сложно: $f \in SAT$ для новой переменной x_0 , f существенно зависит от x_0).

Верификация схем

Таким образом, невозможно найти идеальную форму представления булевых функций. Поэтому важно расставить приоритеты. Нас интересует построение схем. Рассмотрим две проблемы из области верификации схем. Эти проблемы очень важны, и кроме того, являются подпроблемами больших проблем.

Для логических схем выделяют 2 основных типа: комбинационные схемы и схемы с элементами памяти.

Комбинационные схемы не имеют элементов памяти.

Хотя схемы с элементами памяти более функциональны, исследование комбинационных схем очень важно.

1. Они интересны сами по себе: как вычисление функции за один шаг.
2. Схемы с элементами памяти являются расширением комбинационных схем.

При построении схем важна их функциональная корректность. Описание свойств схемы называется спецификацией. Разработка логической схемы сводится к итерационным шагам: спецификация – реализация: реализация на i -ом шаге задает спецификацию $i + 1$ шага.

Доказательство функциональной корректности может осуществляться следующим образом:

1. Оценка поведения спецификации и реализации на большом числе входящих векторов.
2. Формальная верификация (математическое доказательство).
3. Частичная верификация (математическое доказательство, что реализация удовлетворяет, по крайней мере, некоторым важным свойствам поведения спецификации): свойство надежности (определенные «плохие» события не могут произойти); свойство жизнеспособности (возможно, что определенные «хорошие» события произойдут).

Формальная верификация комбинационных схем.

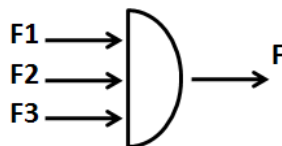
При моделировании схем рассматриваются сети логических элементов. Логический элемент – это элементарная схема, соединяющая транзисторы так, чтобы вычислялась определенная Булева операция на входах. Задача «логического синтеза» – оптимизировать схему соответственно определенным критериям: количество элементов, размер чипа, экономия энергии, задержки и т.п.

Для решения этих задач разрабатывались системы логического синтеза. Разрабатывались такие системы и в академических центрах:

- MINI (IBM Research, 1974).
- ESPRESSO (University of California at Berkley, 1984).
- MIS (University of California at Berkley, 1987).
- BOLD (University of Colorado at Boulder, 1989).
- SIS (University of California at Berkley, 1992).

Основные требования к представлениям функций:

1. Так как в основном схемы – это сети логических элементов, то возникает необходимость представления функции, являющейся выходной для элемента на входы которого поданы другие булевы функции:

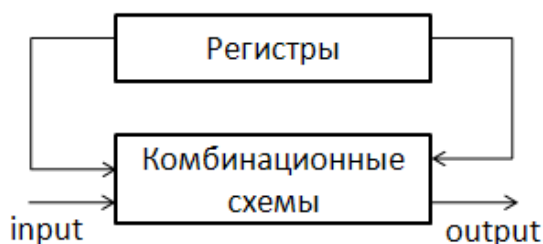


Булевы операции должны осуществляться эффективно.

2. Минимизация представления (например, числа элементов схемы) важна. Например, если в результате минимизации можно получить совершенное представление, то можно проверить эквивалентны ли спецификация и реализация.

Функциональная эквивалентность должна проверяться эффективно (проблема SAT, т.е. выполнимость схемы, должна вычисляться эффективно).

3. Схемы с памятью.



На первый взгляд, описание таких схем может быть осуществлено при помощи конечных детерминированных автоматов (KDA), для которых минимизация (а значит и проверка эквивалентности) выполняется эффективно относительно количества состояний. Но, если состояние KDA кодируется 80 битами, то количество возможных состояний равно 2^{80} . Время существования Вселенной 2^{34} лет $\approx 2^{44}$ часов $\approx 2^{44}$ секунд, следовательно, если даже обрабатывать 2 миллиона состояний в секунду, то не хватило бы всего времени существования Вселенной для анализа эквивалентности состояний такого автомата.

Довольно долго работа системы оценивалась на большом множестве входов. Однако в 1994 компанией Intel проверки ненадежности процессора Pentium кончилась весьма плачевно. Ошибка Pentium FDIV – это ошибка в модуле операций с плавающей запятой в оригинальных процессорах Pentium, выпускавшихся фирмой Intel в 1994 году. Ошибка выражалась в том, что при проведении деления над числами с плавающей запятой при помощи команды процессора FDIV в некоторых случаях результат мог быть некорректным.

Данная ошибка была впервые обнаружена и опубликована профессором Линчбургского колледжа Томасом Найсли в октябре 1994 года. Стремление производителя утаить проблему и реакция на ее обнаружение вызвали недовольство потребителей и обширную критику в СМИ, в том числе жесткий репортаж CNN. В результате компания объявила, что будет свободно обменивать дефектные процессоры всем желающим. История стоила Intel более половины прибыли за последний квартал 1994 г. – \$475 млн.

Современная формальная верификация основана на эффективном представлении множества состояний, или характеристической функции множества состояний (т.е. булевой функции).

3.4 OBDD – эффективная структура данных

Начало изучения OBDD для VLSI-дизайна положили работы Брайнта [1].

Определение 3.4.1. Пусть π – порядок множества переменных $\{x_1, \dots, x_n\}: \{x_{\pi(1)}, \dots, x_{\pi(n)}\}$. OBDD (Ordered Binary Decision Diagram – упорядоченная бинарная диаграмма решений) с порядком чтения переменных π – это ациклический орграф, имеющий ровно 1 корень, 2 финальные вершины, не имеющие выходных дуг. Эти финальные вершины помечены 0 и 1 (0-вершина, 1-вершина). Каждая нефинальная вершина помечена переменной x_i и имеет 2 выходные дуги, помеченные 0 и 1. Порядок, в котором переменные встречаются на пути в графах соответствуют порядку π , т.е. если дуга ведет от вершины, помеченной x_i , к вершине, помеченной x_j , то $\pi^{-1}(i) < \pi^{-1}(j)$.

Определение 3.4.2. OBDD представляет булеву функцию $f \in B_n$, если $\forall a \in B^n$ вычисленный путь на a , т.е. путь от корня до финальной вершины в соответствии с a , достигает вершину, помеченную $f(a)$.

Следующие свойства OBDD очень важны:

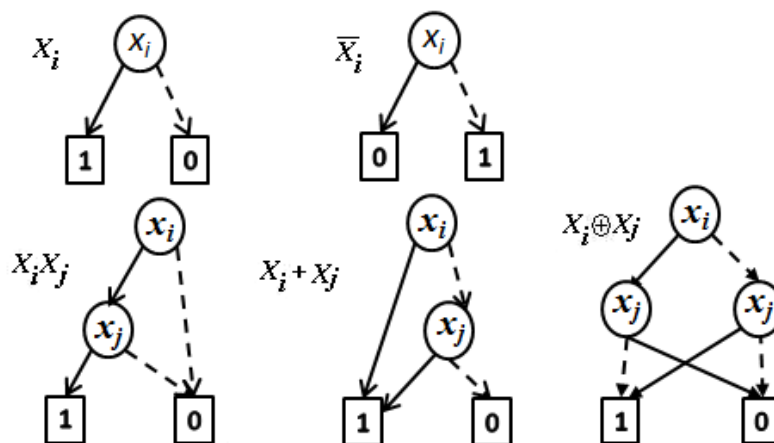
1. Редуцированная OBDD – это совершенное представление булевой функции.
2. Манипулирование редуцированными OBDD может выполняться эффективно.
3. Для многих практических интересных функций OBDD имеют маленький размер.

Вершина *OBDD* с меткой x_i определяет *разложение Шеннона*:

Если *OBDD* имеет корень, помеченный x_i , и представляет функцию $f(x_1, \dots, x_n)$, то $f = x_i f_{x_i} + \bar{x}_i f_{\bar{x}_i}$, где

$$f_{x_i} = f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n), f_{\bar{x}_i} = f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n).$$

Пример 3.4.1.



Определение 3.4.3. Пусть P_1, P_2 – *OBDD*. P_1 и P_2 – *изоморфны* ($P_1 \approx P_2$), если существует биекция ϕ между вершинами P_1 и P_2 :

1. пометка v и $\phi(v)$ совпадает;
2. для $a \in \{0,1\}$, если a -дуга от v ведет к w , то a -дуга от $\phi(v)$ ведет к $\phi(w)$.

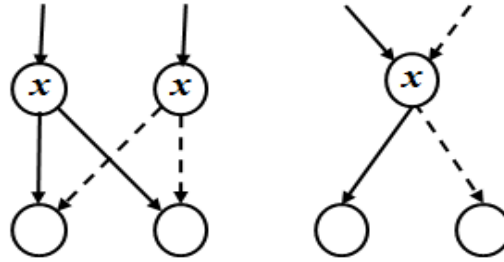
Определение 3.4.4. *OBDD* называется *редуцированной*, если

- не существует вершины v такой, что 1-дуга и 0-дуга от v ведут к одной и той же вершине w ;
- не существует 2-х вершин v и w таких, что *OBDD* с корнями v и w – *изоморфны*: $OBDD(v) \approx OBDD(w)$.

Определение 3.4.5. 2 правила *редукции OBDD*:

- **Правило «удаления»:** если 1-дуга и 0-дуга от вершины v ведут к одной и той же вершине w , то можно перенаправить дуги, входящие в v , на вход w .
- **Правило «склеивания»:** если u и v помечены одной и той же переменной, а 1-дуги от u и v ведут к одной и той же вершине, и 0-дуги ведут к одной и той же вершине, то можно отождествить u и v , удалив

одну из этих вершин, перенаправив все ее входящие дуги к оставшейся вершине.



3.5 Свойства редуцированных OBDD

Теорема 3.5.1. *OBDD - редуцирована тогда и только тогда, когда неприменимо ни одно из правил редуцирования.*

Доказательство:

Необходимость. OBDD редуцирована, следовательно ни одно из правил не может быть применено.

Достаточность. Пусть не может быть применимо правило «удаления». Следовательно, 1 свойство редуцированных OBDD выполнено.

Пусть OBDD имеет 2 вершины v и w и $OBDD(v) \approx OBDD(w)$. Следовательно, возможны два случая:

- $1 - \text{сын}(u) = 1 - \text{сын}(v)$, $0 - \text{сын}(u) = 0 - \text{сын}(v)$, следовательно, может быть применимо правило «склеивания».

- Пусть $u' = 1 - \text{сын}(u) \neq 1 - \text{сын}(v) = v'$ или $0 - \text{сын}(u) \neq 0 - \text{сын}(v)$. Пусть 1 неравенство выполнено, следовательно $OBDD(v') \approx OBDD(w')$. Повторим процедуру с $u0$ и $v0$. На каждом шаге уменьшается множество переменных, которые могут встречаться в под-OBDD, поэтому не более, чем за n шагов процесс приведет к возможности применить правило «склеивания», т.к. 1-вершина и 0-вершина одинаковы. $\square$

Покажем, что любая булева функция имеет совершенное представление в виде OBDD при фиксированном порядке чтения переменных.

Далее для простоты будем рассматривать естественный порядок: $x_1, \dots, x_n$. Рассмотрим x_i -вершину v .

На пути от корня до v читаются x_j : $j \leq i - 1$.

Если вход $c_1, \dots, c_n$ соответствует вычисленному пути, ведущему через v , то $OBDD(v)$ вычисляет подфункцию $f_{x_1=c_1, \dots, x_{i-1}=c_{i-1}}$.

Теорема 3.5.2. Пусть S_i – множество подфункций f , полученных при фиксировании переменных $\{x_j: j \leq i-1\}$, для которых x_i – существенна. Вплоть до изоморфизма существует единственная OBDD минимального размера для f с порядком $x_1, \dots, x_n$. Эта OBDD имеет $|S_i|$ вершин, помеченных x_i .

Доказательство:

1. Сначала построим минимальную OBDD P для f , которая для любого i содержит ровно $|S_i|$ вершин, помеченных x_i .

- Если f константа, то P является графом с 2-мя вершинами (обе финальные).

- Индукционное предположение: для любого $j \geq i+1$, для любого $g \in S_j$ в P имеется ровно 1 вершина x_j -вершина v : $OBDD(v)$ вычисляет g .

- Индукционный переход: рассмотрим i . Пусть $h \in S_i$, следовательно, h_{x_i} и $h_{\bar{x}_i}$ являются константами или принадлежат S_j для некоторого $j \geq i+1$.

Для h введем вершину v_h . Пометим ее переменной x_i и направим из нее 1-дугу в h_{x_i} , а 0-дугу в $h_{\bar{x}_i}$.

P вычисляет f , что доказывается также индуктивно, т.к. P правильно вычисляет подфункции: это верно для финальных вершин: $h = x_i h_{x_i} + \bar{x}_i h_{\bar{x}_i}$

2. Минимальность P .

Доказательство от противного. Если P не минимальна, то существует OBDD P' , для которой существует индекс i такой, что P' , содержит менее $|S_i|$ вершин, помеченных x_i .

Но существует $|S_i|$ различных подфункций f вида

$$f_{x_1=c_1, \dots, x_{i-1}=c_{i-1}}$$

существенно зависящих от x_i .

Рассмотрим $Q = \{(c_1, \dots, c_n) | f_{x_1=c_1, \dots, x_{i-1}=c_{i-1}} \in S_i\}$.

Т.к. $f_{x_1=c_1, \dots, x_{i-1}=c_{i-1}}$ принадлежат S_i , пути, помеченные $x_1 = c_1, \dots, x_{i-1} = c_{i-1}$, ведут в вершины, помеченные x_i , следовательно, существуют $\bar{a}, \bar{b} \in Q: f_{\bar{a}} \neq f_{\bar{b}}$, но пути, помеченные $\bar{a}$ и $\bar{b}$, ведут в одну и ту же вершину, что

приводит к противоречию, т.к. корень *OBDD* соответствует лишь одной подфункции.

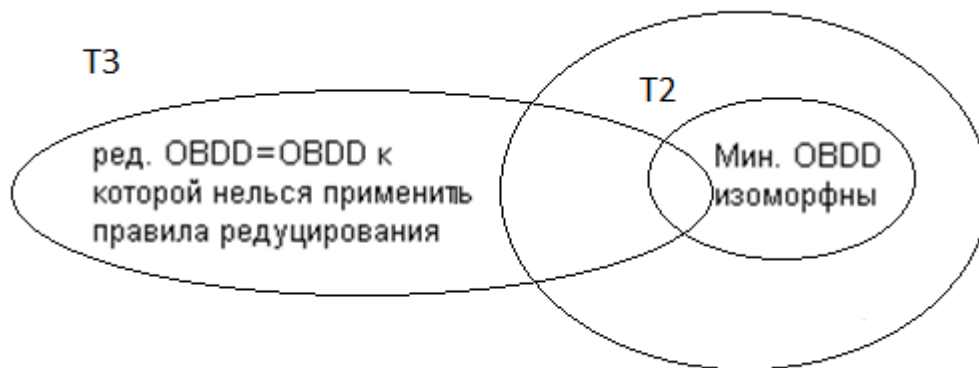
3. Минимальная *OBDD* изоморфна *P*.

Следуя предыдущим рассуждениям: любая минимальная *OBDD* *R* с порядком $x_1, \dots, x_n$ содержит для любой подфункции $g \in S_i$ вершину, помеченную x_i с сыновьями, $g_{x_i=1}$ и $g_{x_i=0}$, следовательно, любая *OBDD* с тем же количеством вершин изоморфна *P*, а любая *OBDD* *R* неизоморфная *P* имеет дополнительные вершины – следовательно, *R* не является минимальной. $\square$

Теорема 3.5.3. Пусть *P* – *OBDD* булевой функции *f* с порядком π , *P* – изоморфна минимальной *P'* для *f* с π тогда и только тогда, когда ни одно из правил редуцирования не может быть применимо к *P*.

Справедлива и обратная теорема.

Поясним смысл утверждения:



1. Для минимальных *OBDD* невозможно использовать правила редуцирования.

2. Если *P* – неминимальна, то можно принять одно из правил редуцирования.

Доказательство:

Пусть *f* – не константа, $\pi = (x_1, \dots, x_n)$. Из теоремы 3.5.2 следует, что любая *OBDD*, вычисляющая *f*, содержит хотя бы одну x_i -вершину для каждой подфункции из S_i . Если *P* не совпадает с минимальной *P'*, то существует индекс $i \in \{1, \dots, n\}$ такой, что *P* содержит x_i -вершин больше, чем $|S_i|$. Пусть *k* – максимум из таких *i*. Тогда в *P* любая подфункция из S_i для $j > k$

представляется в точности одной вершиной. Так как существует более $|S_k|$ x_k -вершин,

1. или существует x_k -вершина u , вычисляющая $g \notin S_k$, т.е. g не зависит от x_k ,
2. или существуют x_k -вершины v и w , в которых вычисляется одна и та же функция $h \in S_k$.

В случае,

1. если $g = g_{x_k} = g_{\overline{x_k}}$, то можно применить правило удаления для u , т.к. $g_{x_k} = g_{\overline{x_k}}$;
2. если сыновья v , w вычисляют $h_{x_k} = h_{\overline{x_k}}$, то применимо правило склеивания.

В обоих случаях получаем противоречие. $\square$

Следствие 3.5.1. *Для любого порядка чтения переменных π редуцированная OBDD для любой булевой функции с порядком чтения π определяется однозначно (с точностью до изоморфизма).*

3.6 Алгоритм редукции

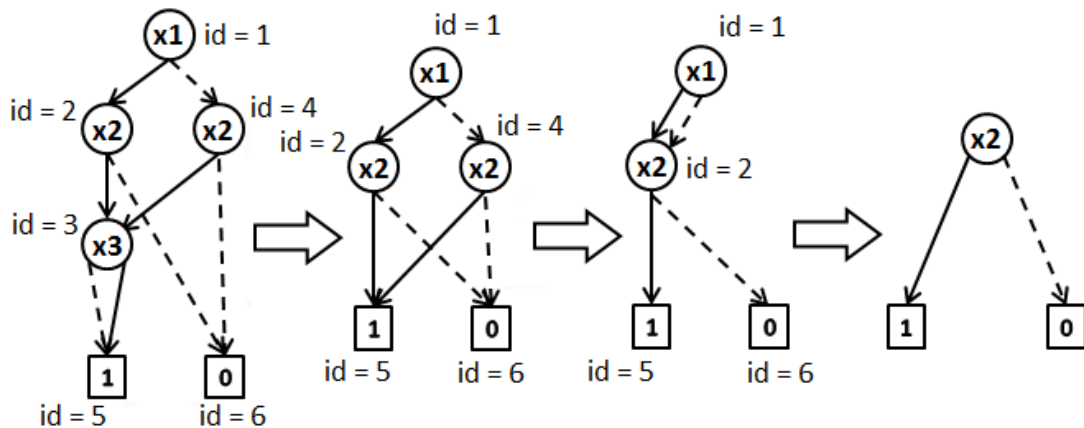
Основная идея: алгоритм представляет собой несколько фаз, в каждой из которых используются оба правила редукции. Алгоритм заканчивает свою работу, когда после очередной фазы размер OBDD остается неизменным. В каждой фазе просмотр OBDD осуществляется снизу вверх.

Приведем описание одной фазы (для простоты возьмем естественный порядок чтения переменных, т.е. $(x_1, \dots, x_n)$).

1. Перенумеруем все вершины OBDD (функция id).
2. Для всех i от n до 1:
 - 2.1 находим V_i – множество x_i -вершин.
 - 2.2 для всех $v \in V_i$ делаем следующее:
 - 2.2.1. Если $id(0 - \text{сын}(v)) = id(1 - \text{сын}(v))$, то применяем к v правило удаления, в противном случае формируем значение функции $key(v) = (id(0 - \text{сын}(v)), id(1 - \text{сын}(v)))$.

2.3. Сортируем $key(v)$ для V_i так, что $(key(v_j) \leq key(v_{j+1}))$.

2.4. Для всех $v_j \in V_i$: если $key(v_j) = key(v_{j-1})$, то удаляем v_{j-1} , переводя все входящие в нее дуги в v_j



Основные конструкции.

Существует 2 основных метода построения редуцированной *OBDD*

1 метод.

а) Строим бинарное дерево решений.

б) Отождествляем финальные вершины, имеющие одинаковые пометки.

в) Применяем алгоритм редуцирования (недостаток: большой размер).

2 метод. Начинаем с корня:

а) Определяем, существенна ли переменная x_{i_1} , если «да», то строим вершину с двумя отростками.

б) Для каждой новой вершины определяем подфункции, т.е. пометку и новую вершину (или отождествляем с какой-то старой). (недостаток: проверка существенности переменной и эквивалентности функций – сложны).

Пример 3.6.1.

$$f(x_1, x_2, x_3, x_4) = x_2(x_3 + \bar{x}_4) + \bar{x}_1\bar{x}_2x_4 + x_1\bar{x}_2\bar{x}_4,$$

$$\pi = (x_1, x_2, x_3, x_4):$$

$$f_{x_1} = x_2(x_3 + \bar{x}_4) + \bar{x}_2\bar{x}_4$$

$$f_{\bar{x}_1} = x_2(x_3 + \bar{x}_4) + \bar{x}_2x_4$$

$$f_{x_1x_2} = f_{\bar{x}_1x_2} = x_3 + \bar{x}_4$$

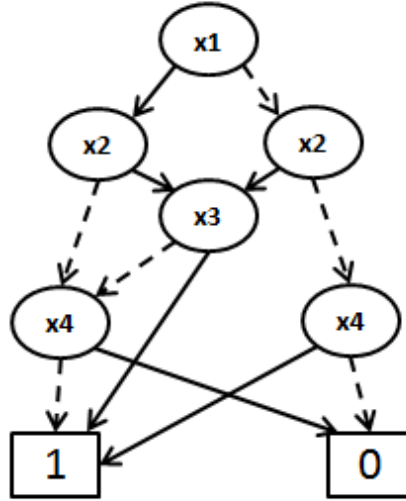
$$f_{x_1\bar{x}_2} = \bar{x}_4$$

$$f_{\bar{x}_1\bar{x}_2} = x_4$$

$$f_{x_1x_2x_3} = f_{\bar{x}_1x_2x_3} = 1$$

$$f_{\bar{x}_1\bar{x}_2x_3} = f_{\bar{x}_1\bar{x}_2\bar{x}_3} = x_4$$

$$f_{x_1 x_2 \bar{x}_3} = f_{\bar{x}_1 x_2 \bar{x}_3} = f_{x_1 \bar{x}_2 x_3} = f_{x_1 \bar{x}_2 \bar{x}_3} = \bar{x}_4$$



Пусть $*$ – булева операция: например, конъюнкция или дизъюнкция. Покажем, как построить *OBDD* для функции $f * g$, если известны *OBDD* для f и g . Верно следующее разложение:

$$f * g = x_i(f_{x_i=1} * g_{x_i=1}) + \bar{x}_i(f_{x_i=0} * g_{x_i=0}).$$

Рекурсивная конструкция.

Строим *OBDD* P_1 и P_0 для $(f_{x_i=1} * g_{x_i=1})$ и $(f_{x_i=0} * g_{x_i=0})$.

Вводим x_i -вершину, а-сын которой есть корень $P_a, a \in \{0,1\}$. Разложение f и g на подфункции связано с необходимостью рассматривать 2^n подфункций. Выходом является рекурсивная процедура с движением по узлам P_1 и P_0 . Каждому узлу новой *OBDD* ставится в соотношение пара (f, g) , где f, g – узлы P_1 и P_0 соответственно (пары образуют Table).

Алгоритм:

вход: *OBDD* F и G для f и g с порядком чтения переменных π ,

$*$ – бинарная операция,

выход: *OBDD* для $f * g$

Oper($F, G, *$)

if (F и G – финальные вершины) then

Return($F * G$)

else if $(F, G) \in Table$ then

Return($F G$)

else {

```

//       $x_i$  - первая переменная в  $\pi$ 
//      существенная для  $F$  или  $G$ 
//      Строим новую  $x_i$ -вершину  $v$ :

0 – сын( $v$ ) =  $Oper(F_{x_i=0}, G_{x_i=0}, *)$ ;
1 – сын( $v$ ) =  $Oper(F_{x_i=1}, G_{x_i=1}, *)$ ;
InsertTable( $F, G, v$ );
Return( $v$ );
}

```

Полученные *OBDD* в общем случае не редуцированы.

Теорема 3.6.1. Пусть f_1 и f_2 представлены *OBDD* P_1 и P_0 соответственно. Тогда для любой булевой операции $*$ редуцированная *OBDD* P функции $f_1 * f_2$ может быть построена за время порядка $O(\text{size}(P_1) * \text{size}(P_0))$.

Без доказательства.

Теорема 3.6.2. Пусть булевы функции f_1 и f_2 представлены *OBDD* P_1 и P_2 соответственно, тогда тест на эквивалентность представлений может быть выполнен за время $O(\text{size}(P_1) + \text{size}(P_2))$.

Для этого сначала редуцируем P_1 и P_2 . Затем проверяется изоморфизм полученных *OBDD* (переход в глубину по обеим *OBDD* параллельно).

Глава 4. Эффективная реализация OBDD

Для практических приложений также необходимо эффективно осуществлять реализацию OBDD по времени и памяти.

4.1 Основные идеи

4.1.1 Представление отдельного узла OBDD

Центральной идеей является *представление отдельного узла OBDD* при помощи трех полей:

- index (2 байта) – индекс i переменной x_i ;
- left, right (2 поля по 4 байта) – ссылки на сыновей узла.

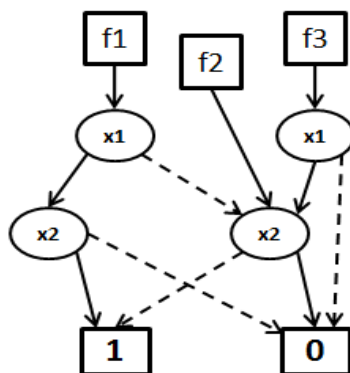
Таким образом, количество переменных ≤ 65536 . Это позволит получать ссылку на адреса полного виртуального адресного пространства (при 32-битной архитектуре полное адресное пространство имеет размерность $2^{32} \approx 4 \times 10^9 \approx 4Gb$ узлов).

Для практической реализации может быть полезна дополнительная информация, связанная с каждым узлом. Но большие редуцированные OBDD требуют осторожного увеличения хранимой информации. Компромисс между эффективностью введения дополнительной информации для любого узла и общим размером OBDD требует большой работы.

4.1.2 Разделяемые (shared) OBDD

Некоторые функции могут быть представлены одним и тем же ациклическим орграфом с несколькими корнями. Такое представление назовем разделяемыми OBDD.

Пример 4.1.1. $f_1 = (x_1 \equiv x_2)$, $f_2 = \bar{x}_2$, $f_3 = x_1 \bar{x}_2$



В таких разделяемых *OBDD* можно добиться того, что каждая подфункция представляется не несколько раз, а лишь один. Тогда представление *OBDD* будет не только совершенным, но и строго совершенным. При этом две эквивалентные функции f и g не только имеют единую редуцированную *OBDD*, но при реализации соответствуют ссылке на один и тот же элемент памяти, следовательно, проверка на эквивалентность требует лишь одной операции сравнения.

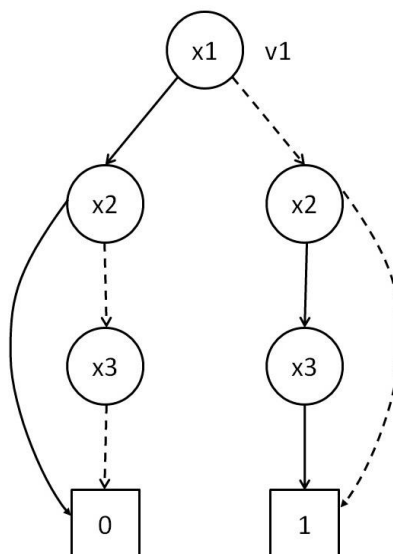
4.1.3 Уникальная таблица (unique) таблица и строгая совершенность

Строгая совершенность в сочетании с представлением в виде разделяемых *OBDD* позволяет сохранять все *OBDD* (конструируемые последовательно) в редуцированной форме. То есть каждая частичная функция представляется в *OBDD* ровно 1 раз.

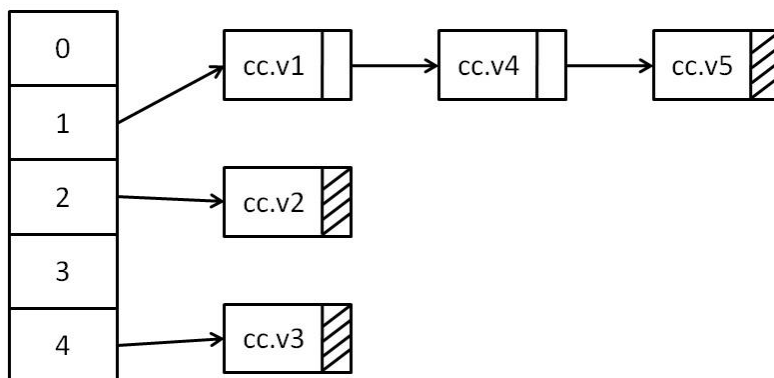
Каждый раз, когда должна добавляться новая x_i -вершина, определяется ее 1-сын и 0-сын (которые уже существуют). Затем проверяется, нет ли уже вершины с такой спецификацией. Если «нет», строится новая вершина; если «да», используется существующий узел.

Для определения существования узла (тройки $\langle x_i, 0 - \text{сын}, 1 - \text{сын} \rangle$) используется уникальная таблица. Чтобы ускорить поиск тройки, используются хэш-функция h . При этом определяется число, соответствующее тройке. Различные тройки могут соответствовать одному значению h . Такие тройки удобно сохранять в виде списка.

Пример 4.1.2. Пусть функция $f(x_1, x_2, x_3)$ задается бинарной программой со следующей нумерацией вершин.



Можно ускорить алгоритм, храня узлы в виде списков. Используем для тройки (x_i, v_j, v_k) хеш-функцию $h(x_i, v_j, v_k) = (i + j + k) \bmod 5$. Тогда в основную конструкцию узла стоит добавить поле *NEXT*. Тогда



4.1.4 ITE-алгоритм.

Пусть имеется представление функций f и g в виде *OBDD*. Для представления в виде *OBDD* функции $f * g$ для некоторой бинарной операции используется разложение Шеннона:

$$f * g = x_i(f_{x_i} * g_{x_i}) + \bar{x}_i(f_{\bar{x}_i} * g_{\bar{x}_i})$$

ITE-оператор — это булева функция от трех переменных, соответствующая оператору *if x then y else z*.

$$ITE(x, y, z) = x \cdot y + \bar{x} \cdot z$$

Все булевы функции от двух переменных можно представить в виде *ITE* - оператора:

	Функция (f, g)	ITE-оператор	
0000	0	0	
0001	$f \cdot g$	$ITE(f, g, 0)$	
0010	$f \rightarrow g$	$ITE(f, \bar{g}, 0)$	
0011	f	f	
0100	$f \leftarrow g$	$ITE(f, 0, g)$	
0101	g	g	
0110	$f \oplus g$	$ITE(f, \bar{g}, g)$	
0111	$f + g$	$ITE(f, 1, g)$	

1000	$\overline{f + g}$	$ITE(f, 0, \bar{g})$	стрелка Пирса
1001	$f \equiv g$	$ITE(f, g, \bar{g})$	
1010	$\bar{g}$	$ITE(g, 0, 1)$	
1011	$f \leftarrow g$	$ITE(f, 1, \bar{g})$	
1100	$\bar{f}$	$ITE(f, 0, 1)$	
1101	$f \rightarrow g$	$ITE(f, g, 1)$	
1110	$f g$	$ITE(f, \bar{g}, 0)$	штрих Шеффера
1111	fg	1	

Пусть f, g, h – булевы функции и x_i – первая переменная в порядке. Для вычисления $ITE(f, g, h)$ используется следующая рекурсивная декомпозиция:

$$\begin{aligned} ITE(f, g, h) &= f \cdot g + \bar{f} \cdot h = x_i(f_{x_i} \cdot g_{x_i} + \bar{f}_{x_i} \cdot h_{x_i}) + \bar{x}_i(f_{\bar{x}_i} \cdot g_{\bar{x}_i} + \bar{f}_{\bar{x}_i} \cdot h_{\bar{x}_i}) \\ &= (x_i, ITE(f_{x_i}, g_{x_i}, h_{x_i}), ITE(f_{\bar{x}_i}, g_{\bar{x}_i}, h_{\bar{x}_i})) \end{aligned}$$

Эта тройка соответствует вершине, которая определялась для уникальной таблицы.

Рекурсивное построение ITE заканчивается, если

- а) первый аргумент – константа: $ITE(1, f, g) = f$; $ITE(0, f, g) = g$;
- б) или $ITE(f, 1, 0) = f$; $ITE(f, g, g) = g$.

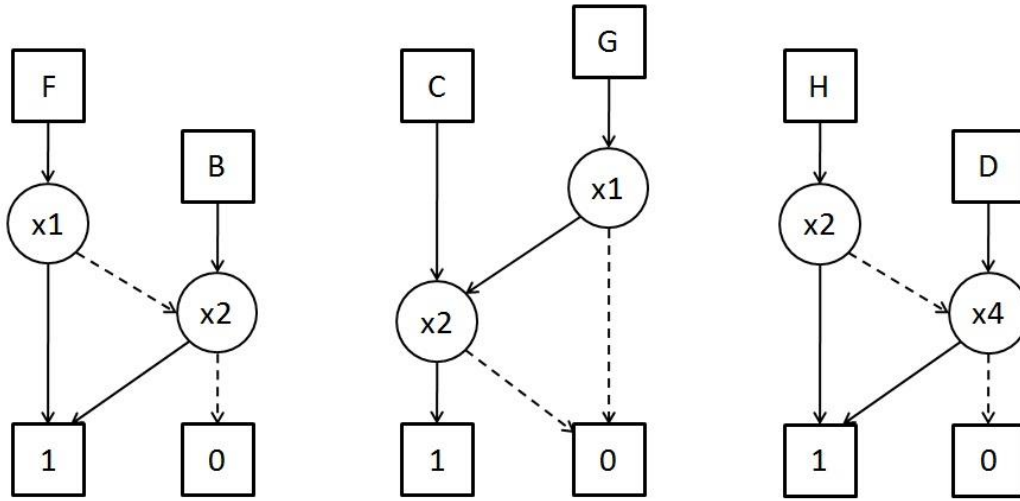
Пусть f, g, h представлены с помощью $OBDD$ F, G, H , тогда временная сложность построения $ITE(f, g, h)$ ограничена кубической функцией

$$O(\text{size}(F) \cdot \text{size}(G) \cdot \text{size}(H)),$$

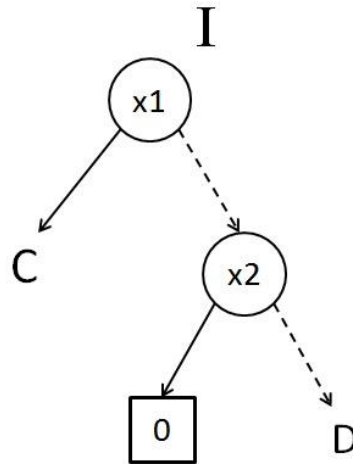
так как любая тройка вершин из F, G, H в данном алгоритме встречается не более одного раза.

Если рассматривать бинарную операцию для f, g , представленных с помощью $OBDD$ F, G сложность построения соответствующей $OBDD$ ограничена даже квадратичной функцией. Действительно, если один из аргументов ITE – константа, то будет изменяться только пара вершин. Тот же эффект наблюдается, если два аргумента ITE совпадают или, как будет показано позже, противоположны.

Пример 4.1.3. Пусть $f = x_1 + x_2$, $g = x_1 \cdot x_2$, $h = x_2 \cdot x_4$.



$$\begin{aligned}
I &= ITE(F, G, H) = (x_1, ITE(F_{x_1}, G_{x_1}, H_{x_1}), ITE(F_{\bar{x}_1}, G_{\bar{x}_1}, H_{\bar{x}_1})) \\
&= (x_1, ITE(1, C, H), ITE(B, 0, H)) \\
&= (x_1, C, (x_2, ITE(B_{x_2}, 0, H_{x_2}), ITE(B_{\bar{x}_2}, 0, H_{\bar{x}_2}))) \\
&= (x_1, C, (x_2, ITE(1, 0, 1), ITE(0, 0, D))) = (x_1, C, (x_2, 0, D))
\end{aligned}$$



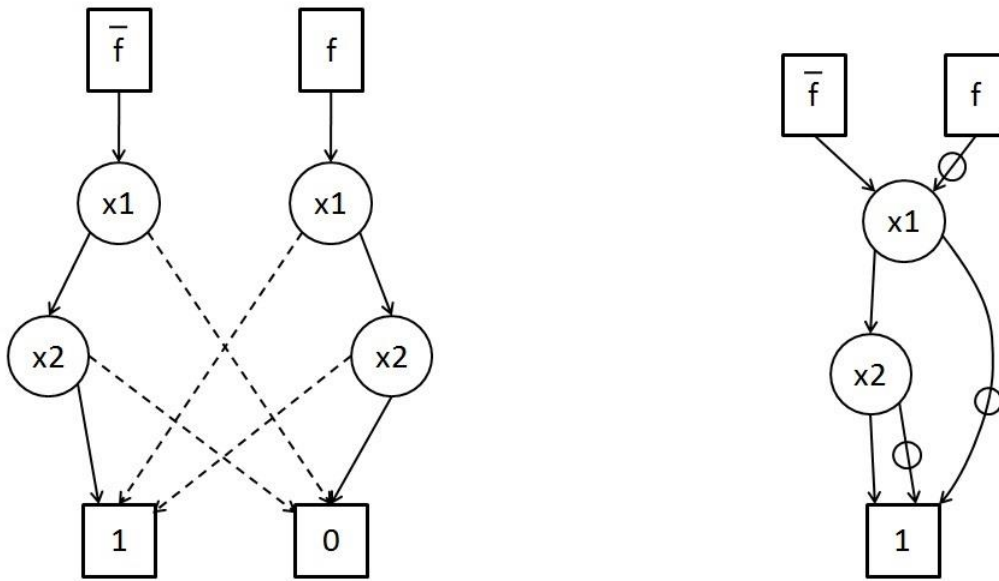
Для *ITE* -алгоритма можно рассматривать не только временную сложность, но и затраты на сохранение информации. Для запоминания троек функций, соответствующих *ITE*, используются кэш-базированные хэш-таблицы. В них хранятся не всевозможные тройки, а лишь ограниченное количество ($\leq k$) для каждого значения хэш-функции.

Старые значения затираются новыми значениями. Если какое-то из затертых значений необходимо восстановить, оно перевычисляется. Это может привести к существенному росту времени реализации, но это происходит крайне редко. Кроме того, в *ITE*-алгоритме результат вычисления требуется довольно быстро после своего вычисления. Чем больше времени прошло после вычисления узла, тем меньше вероятность, что он понадобится.

4.1.5 Дополнительные дуги (complemented edges).

Пусть функции f и $\bar{f}$ представляются графами, у которых различаются лишь финальные вершины. Как использовать этот факт? Свяжем с каждой дугой *бит дополнения*. Если он равен 0, то вычисляется оригинальная функция, если он равен 1, то вычисляется дополнение оригинальной функции, тогда f и $\bar{f}$ вычисляются на одном и том же графе. При этом из *OBDD* можно удалить оригинальную 0-вершину.

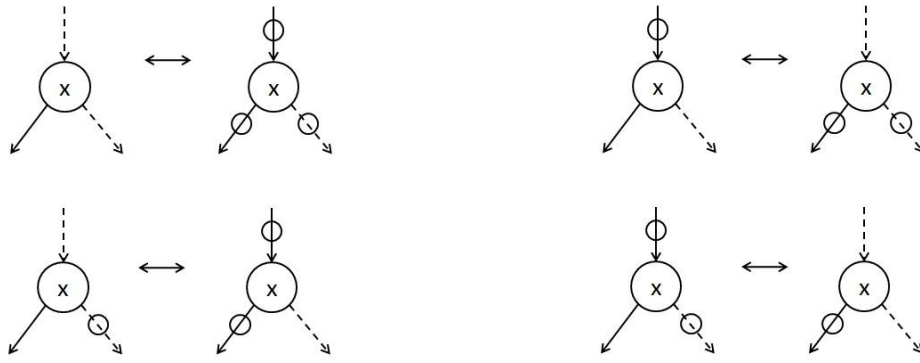
Далее будем помечать на рисунках каждую дополнительную дугу кружком.



При добавлении дополнительных дуг совершенность представления теряется. Дело в том, что одна и та же функция может теперь представляться по-разному:

$$\begin{aligned}
 \overline{x_i \cdot f + \bar{x}_i \cdot g} &= \overline{x_i \cdot f} \cdot \overline{\bar{x}_i \cdot g} = (\bar{x}_i + \bar{f}) \cdot (x_i + \bar{g}) = x_i \cdot \bar{f} + \bar{x}_i \cdot \bar{g} + \bar{f} \cdot \bar{g} \\
 &= x_i \cdot \bar{f} + \bar{x}_i \cdot \bar{g} + (x_i + \bar{x}_i) \cdot \bar{f} \cdot \bar{g} = x_i \cdot (\bar{f} + \bar{f} \cdot \bar{g}) + \bar{x}_i \cdot (\bar{g} + \bar{f} \cdot \bar{g}) \\
 &= x_i \cdot \bar{f} + \bar{x}_i \cdot \bar{g}
 \end{aligned}$$

С тройкой дуг около одного узла (две выходные дуги, одна входная) можно связать 8 реализуемых функций, в зависимости от того, какая из дуг заменена на дополнительную дугу. Но количество полученных при этом функций равно 4.



Таким образом, для совершенности представления никогда не дополняются 1-дуги. Когда при построении *OBDD* добавляется вершина, изменяются биты дополнительных дуг так, чтобы это свойство сохранялось. Это требование достаточно, чтобы сохранить совершенность представления. При этом важно, что, как правило, дуга к корню должна быть дополнена. Это важно в частности для представления функций 0-вершины.

Преимущество представления *OBDD* при помощи дополнительных дуг:

- Компактность представления (размер может уменьшиться до 50 процентов)
- Отрицание функции реализуется за константное время
- Выполнение булевских операций ускоряется при использовании ряда правил, например, $f \cdot \bar{f} = 0, f + \bar{f} = 1$.

Использование дополнительных дуг для *ITE*-алгоритма ограничивает временную сложность квадратично.

Теорема 4.1.1. Пусть функции f_1 и f_2 представляются *OBDD* P_1 и P_2 с дополнительными дугами. Для любой булевой операции $*$ редуцированная *OBDD* для $f_1 * f_2$ определяется при помощи *ITE*-алгоритма за время $O(\text{size}(P_1) \cdot \text{size}(P_2))$.

На практике применение дополнительных дуг дает экономию размера около 10 процентов, а экономию времени – до двух раз.

4.1.6 Стандартные тройки.

Существует разложение троек функций (f_1, f_2, f_3) и (g_1, g_2, g_3) такое, что $\text{ITE}(f_1, f_2, f_3) = \text{ITE}(g_1, g_2, g_3)$

$$\begin{aligned} f \vee h &= f + g = ITE(f, f, g) = ITE(f, 1, g) \\ &= ITE(g, 1, f) = ITE(g, g, f) \end{aligned}$$

Рассмотрим некоторые правила, переводящие тройки в стандартную форму. Первая серия заменяет функцию на константу, если это возможно.

$$\begin{aligned} ITE(f, f, g) &\Rightarrow ITE(f, 1, g) \\ ITE(f, g, f) &\Rightarrow ITE(f, g, 0) \\ ITE(f, g, \bar{f}) &\Rightarrow ITE(f, g, 1) \\ ITE(f, \bar{f}, g) &\Rightarrow ITE(f, 0, g) \end{aligned}$$

Если использовать дополнительные дуги, то проверка, что две функции, использованные в ITE , являются дополнением друг друга, требует константного времени. Серия финальных правил ITE -алгоритма:

$$\begin{aligned} ITE(f, 1, 0) &= f \\ ITE(f, 0, 1) &= \bar{f} \\ ITE(1, f, g) &= f \\ ITE(0, f, g) &= g \\ ITE(f, g, g) &= g \end{aligned}$$

Следующие правила используют коммутативность ITE , если второй или третий аргумент - константа или дополнение друг друга:

$$\begin{aligned} ITE(f, 1, g) &= ITE(g, 1, f) \\ ITE(f, g, 0) &= ITE(g, f, 0) \\ ITE(f, g, 1) &= ITE(\bar{g}, \bar{f}, 1) \\ ITE(f, 0, g) &= ITE(\bar{g}, 0, \bar{f}) \\ ITE(f, g, \bar{g}) &= ITE(g, f, \bar{f}) \end{aligned}$$

Типичен выбор такого распределения пар функций, чтобы первый аргумент зависел от переменной, которая является первой в порядке. Если первый аргумент – это просто переменная x_i , а остальные аргументы не зависят от x_i , получаем x_i -вершину.

$$ITE(f, g, h) = (x_i, g, h),$$

если

$$(f = x_i) \ \& \ (x_i < \text{topvar}(g)) \ \& \ (x_i < \text{topvar}(f));$$

где $topvar(g)$ – первая в порядке переменная, существенная для g .

Следующие правила связаны с использованием дополнительных дуг:

$$ITE(f, g, h) = ITE(\bar{f}, g, h) = \overline{ITE(f, \bar{g}, \bar{h})} = \overline{ITE(\bar{f}, \bar{g}, \bar{h})}$$

Каждая из трех функций f, g, h может быть представлена обычной или дополнительной дугой. Из четырех эквивалентных представлений есть ровно одно, где 2 первых аргумента не представлены дополнительной дугой. Эта тройка и является стандартной тройкой представления в вычислительной таблице.

Указанные правила отображают правила де Моргана.

Пример 4.1.4. Пусть реализована функция $f + g = ITE(f, 1, g)$.

Если нужно реализовать $\bar{f} \cdot \bar{g} = ITE(\bar{f}, \bar{g}, 0) = \overline{ITE(f, 1, g)}$, то используется уже сформированная стандартная тройка $(f, 1, g)$, соответствующий узел соединяется дополнительной дугой.

4.1.7 Управление памятью

Важное значение при реализации *OBDD* имеет управление памятью.

Указанный принцип построения основан на постепенной замене одних *OBDD* другими: каждый функциональный элемент – малая *OBDD*, к соответствующим *OBDD* применяется булева операция (*ITE*-оператор).

Для экономии памяти в частности используются:

1. Механизм сбора мусора.
2. Для хранения уникальной таблицы для каждого узла заводится поле *NEXT* (ссылка на следующий узел в списке коллизий).

Дополнительные дуги кодируются следующим образом. Узлы *OBDD* кодируются адресом, кратным 2. Поэтому адреса узлов сыновей, записанных в полях узла, имеют в последнем бите 0. Если ссылка на сына имеет в последнем бите 1, то соответствующая дуга рассматривается как дополнительная.

4.2 Популярныe OBDD-пакеты

1. OBDD-пакет Brace, Rudell и Bryant.

Разработан в 1989-1990г и представлен в 1990 – Carnegie Mellon University.

Их цель состояла в том, чтобы расширить возможность верификационного программного пакета, позволявшего тестировать транзисторные схемы, когда транзисторы рассматриваются на абстрактном уровне как переключатели.

В этом пакете были разработаны основные принципы представления информации.

2. OBDD-пакет Лонга (Long)

Carnegie Mellon, 1993. Этот пакет использовался в SIS (последовательный синтез, Университет Беркли), и в AT&T.

3. CUDD-пакет (Colorado University Decision Diagrams).

F.Somenzi, апрель 1996. Важное свойство пакета – использование множества алгоритмов, улучшающих порядок чтения переменных.

Литература

1. Meinel, Ch. Algorithms and data Structures in VLSI Design: OBDD-foudations and applications/ Ch. Meinel, T.Theobald.— Springer-Verlag, Berlin, Heidelberg, 1998.— 267 P.
2. Meinel Ch., Mubarakzjanov R. Nonlinear Sifting of Decision Diagrams// Proc. of the 2002 International Conference on VLSI, June 24-27, Monte Carlo Resort, Las Vegas, Nevada, USA. P. 117-123.
3. Frank Pfenning. Principles of Imperative Computation // Lecture Notes on Binary Decision Diagrams. Lecture 19.— 2010.— <https://www.cs.cmu.edu/~fp/courses/15122-f10/lectures/19-bdds.pdf>
4. William Lovas. Principles of Imperative Computation // Lecture Notes on Binary Decision Diagrams. Lecture 25.— 2011. — <http://www.cs.cmu.edu/~wlovas/15122-r11/lectures/25-bdds.pdf>
5. Гэри, М.Р. Вычислительные машины и трудно-решаемые задачи/ М. Р. Гэри, М. Джонсон. — М.: Мир, 1982.— 416 с. (Оригинал: Garey M.R., Jonson M. Computers and intractability: a guide to the theory of \$NP\$-completeness.— San Francisco, USA: W.H. Freeman.— 1978.)
6. Bryant, R.E. Graph-based algorithms for Boolean function manipulation/ R.E. Bryant// IEEE Transactions on Computers. — 1986. — No. 35. — P. 677-691.
7. Motvani R. and Raghavan P. Randomized Algorithms. Cambridge University Press, Cambridge, 1995. — 367 P.
8. Mubarakzjanov R. On probabilistic OBDDs with constant width// Proc. of the Workshop on Computer Science and Information Technologies CSIT'2000, Ufa, Russia, September 18-23, 2000. 2000. V.2. P.62-68.
9. Mubarakzjanov R. Probabilistic OBDDs: on Bound of Width versus Bound of Error// Electronical Colloquium on Computational Complexity, Trier University, Trier.- ECCC TR00-085, 2000, <http://eccc.hpi-web.de/report/2000/085/>
10. Mubarakzjanov R. Bounded-width probabilistic OBDDs and read-once branching programs are incomparable// Electronical Colloquium on Computational Complexity, Trier University, Trier. ECCC TR01-037, 2001, <http://eccc.hpi-web.de/report/2001/037/>
11. Newman I. Testing of function that have small width branching programs// Proc. of the 41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000. IEEE Computer Society. Redondo Beach, California, USA, 2000. P. 251-258.
12. Okol'nishnikova E.A. On lower bounds for branching programs// Siberian Advances in Mathematics. 1993. 3(1). P. 152-166.

13. Paturi R., Simon J. Probabilistic communication complexity// Journal of Computer and System Sciences. 1986. 33(1). P. 106-123.
14. Ponzio S. A lower bound for integer multiplication with read-once branching programs//Proc. 27-th STOC. 1995. P. 130-139.
15. Rabin M.O. Probabilistic Automata// Information and Control. 1963. 6, No. 3. P. 230-245. (Рус. пер.: Кибернетический сборник. М.: Мир. вып. 9.- 1964. С. 123-141.)
16. A.A. Razborov, Lower bounds for deterministic and nondeterministic branching programs}, Proc. Fundamentals in Computing Theory, Springer, Lecture Notes in Computer Science, 529, 47-60.-1991.
17. Sauerhoff M. On nondeterminism versus randomness for read-once branching programs/Electrical Colloquium on Computational Complexity, Trier University, Trier. ECCC, TR97-030. 1997. <http://eccc.hpi-web.de/report/1997/030/>
18. Sauerhoff M. Comment Correction to "Randomness and Nondeterminism are Incomparable for Read-Once Branching Programs/Electrical Colloquium on Computational Complexity, Trier University, Trier. ECCC TR98-018, 1998, <http://eccc.hpi-web.de/report/1998/018/>
19. Sauerhoff M. Lower bounds for randomized read-k-times branching programs//Proc. of STACS, LNCS 1373. Springer Verlag. 1998. P. 105-115.
20. Sauerhoff M. Complexity theoretical results for randomized branching programs/ PhD. thesis, Univ. of Dortmund, Shaker. 1999.
21. Savicky P., Zak S. A large lower bound for 1-branching programs/Electrical Colloquium on Computational Complexity, Trier University, Trier. ECCC, Revision 01 of TR96-036, 1996, <http://eccc.hpi-web.de/report/1996/036/>.
22. Savicky P., Zak, S. A hierarchy for $(1,+k)$ -branching programs with respect to k // Proc. MFCS'97, LNCS 1295. 1997. P. 478-487.
23. Sieling D. und Wegener I. Graph driven BDDs – a new data structure for Boolean functions// Theoretical Computer Science. 1995.– No. 141. P. 283-310.
24. Wegener I. Branching Programs and Binary Decision Diagrams: Theory and Applications. SIAM Monographs on Discrete Mathematics and Applications. Philadelphia, USA, 2000.– 408 P.