УДК 519.7

А. Е. Хмельнов, А. С. Игнатьев, А. А. Семенов

Институт динамики систем и теории управления СО РАН
ул. Лермонтова, 134, Иркутск, 664033. Россия
E-mail: biclop@rambler.ru

ДВОИЧНЫЕ ДИАГРАММЫ РЕШЕНИЙ В ЛОГИЧЕСКИХ УРАВНЕНИЯХ И ЗАДАЧАХ ОБРАЩЕНИЯ ДИСКРЕТНЫХ ФУНКЦИЙ *

В работе рассматриваются вопросы программной реализации одного подхода к задачам обращения дискретных функций. В основе данного подхода лежит техника представления булевых функций в форме двоичных диаграмм решений (BDD). Предложены новые приемы оптимизации использования памяти при работе с BDD. Описанные технологии тестируются на некоторых задачах криптоанализа.

Ключевые слова: двоичные диаграммы решений, логические уравнения, дискретные функции, криптоанализ.

Ведение

Работа посвящена исследованию возможностей применения двоичных диаграмм решений (BDD) в задачах поиска решений логических уравнений и задачах обращения дискретных функций.

Двоичные диаграммы решений (BDD) – это структуры данных, позволяющие во многих практически важных задачах эффективно оперировать с булевыми функциями. Первое упоминание BDD в форме «ветвящихся программ» приводится в работе [1]. Собственно термин BDD возник в работе [2]. Однако настоящее понимание значимости BDD в ряде областей дискретного анализа пришло после публикации статьи [3]. Именно в этой работе была показана каноничность ROBDD-представлений произвольных всюду определенных булевых функций, а также описаны основные алгоритмы «манипулирования булевыми функциями» при помощи BDD. Рост интереса к BDD в последние годы вызван рядом удачных примеров их использования в задачах верификации моделей программ (model checking, см., например [4]), а также в задачах микроэлектронной диагностики (см., например, [5]).

В настоящей работе предполагается подробно рассмотреть некоторые особенности программной реализации BDD-подхода к поиску решений логических уравнений и задачам обращения дискретных функций. Для тестирования эффективности описанных в работе технологий предлагается использовать ряд задач криптоанализа.

Двоичные диаграммы решений (математические основы)

Двоичные диаграммы решений (Binary Decision Diagram, BDD) – это специальный вид направленных помеченных графов, посредством которых можно представлять булевы функции.

Стандартно BDD определяется как направленный ациклический граф, в котором выделена одна вершина с входной степенью 0, называемая корнем, и две вершины с выходной степенью 0, называемые терминальными. Терминальные вершины помечены константами 0 и 1.

* Работа выполнена при поддержке РФФИ (грант № 07-01-00400-а) и Президента РФ (грант НШ-1676.2008.1).

© А. Е. Хмельнов, А. С. Игнатьев, А. А. Семенов, 2009

Все остальные вершины помечаются переменными из множества $X = \{x_1, ..., x_n\}$. Из любой вершины, за исключением терминальных, выходят в точности 2 дуги. Одну дугу обычно рисуют пунктирной, а другую – сплошной линией. Дуга, обозначенная пунктиром, называется low-ребром, дуга, обозначенная сплошной линией, – high-ребром.

Простейшие примеры BDD можно строить на основе двоичных деревьев решений. Деревья решений – это помеченные деревья, используемые в различных разделах дискретной математики. С их помощью, например, очень удобно представлять процесс означивания переменных при вычислении значений булевой функции. Произвольной всюду определенной булевой функции от n булевых переменных (см., например [6]) $f_n : \{0,1\}^n \to \{0,1\}$ можно поставить в соответствие ее двоичное дерево решений. Делается это следующим образом. Вершины (узлы) дерева соответствуют булевым переменным. Пунктирное (low) ребро, исходящее из узла, помеченного переменной $x_i, i \in \{1, ..., n\}$, означает, что данная переменная принимает значение 0, сплошное (high) ребро соответствует тому, что x_i принимает значение 1. Листья дерева помечены значениями рассматриваемой функции при соответствующих наборах значений истинности переменных. Произвольный путь из корня в лист, помеченный $\alpha \in \{0,1\}$, определяет набор или семейство наборов значений истинности, на которых рассматриваемая функция принимает значение α (см. пример 1, приведенный ниже).

Пусть $f_n : \{0,1\}^n \to \{0,1\}$ – произвольная всюду определенная (тотальная) булева функция над множеством булевых переменных $X = \{x_1, ..., x_n\}$ и $T(f_n)$ – ее дерево решений. Если склеить в одну вершину все листья дерева $T(f_n)$, помеченные 0, и то же самое проделать с листьями, помеченными 1, получится BDD. Если произвольный путь π в BDD из корня в терминальную вершину не содержит вершин, помеченных одинаковыми переменными, и его прохождение подчинено общему для всех путей порядку (например, $x_1 \prec x_2 \prec ... \prec x_{n-1} \prec x_n$), то такая BDD называется упорядоченной (OBDD). В записи «$x_1 \prec ... \prec x_n$» здесь и далее подразумевается, что корень рассматриваемой OBDD помечен переменной x_1.

При использовании BDD как структур данных, представляющих булевы функции, довольно часто требуется отличать различные вершины, помеченные одной и той же переменной. Далее для этой цели используем обозначения типа «$v_1(x), v_2(x), ...$». Дети произвольной нетерминальной вершины $v(x)$ обозначаются соответственно через $low(v(x))$ и $high(v(x))$. Также используем обозначение «$\text{var}(v(x)) = x$» или более краткое «$\text{var}(v) = x$».

В произвольной OBDD можно выделять фрагменты (подграфы), которые сами являются OBDD. Для этой цели достаточно объявить соответствующую нетерминальную вершину корнем. Идея сокращенной OBDD (кратко «ROBDD») заключается в склейке повторяющихся фрагментов: ROBDD-граф не должен содержать одинаковых OBDD-подграфов меньших размерностей. Таким образом, ROBDD можно рассматривать как «наиболее сжатое» графическое представление некоторой булевой функции. Сказанное означает, что ROBDD – это OBDD, в которой:

1) равенства

$$\text{var}(v) = \text{var}(u),\ high(v) = high(u), low(v) = low(u)$$

означают, что $v = u$;

2) для любой нетерминальной вершины v имеет место: $high(v) \neq low(v)$.

В работе [3] было показано, что произвольная всюду определенная булева функция при фиксированном порядке означивания переменных имеет однозначное (с точностью до изоморфизма соответствующих графов) ROBDD-представление.

Пример 1. Дана булева функция $f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$. Зафиксируем следующий порядок означивания переменных: $x_1 \prec x_2 \prec x_3$.

На рис. 1 приведен вариант дерева решений для данной функции. На втором рисунке (слева) изображена OBDD, полученная склейкой одноименных терминальных вершин дерева решений. Последний рисунок – это ROBDD рассматриваемой функции, полученная из OBDD в результате склеивания вершины $v_2(x_3)$ с $v_3(x_3)$ и склеивания $v_1(x_3)$ с $v_4(x_3)$.

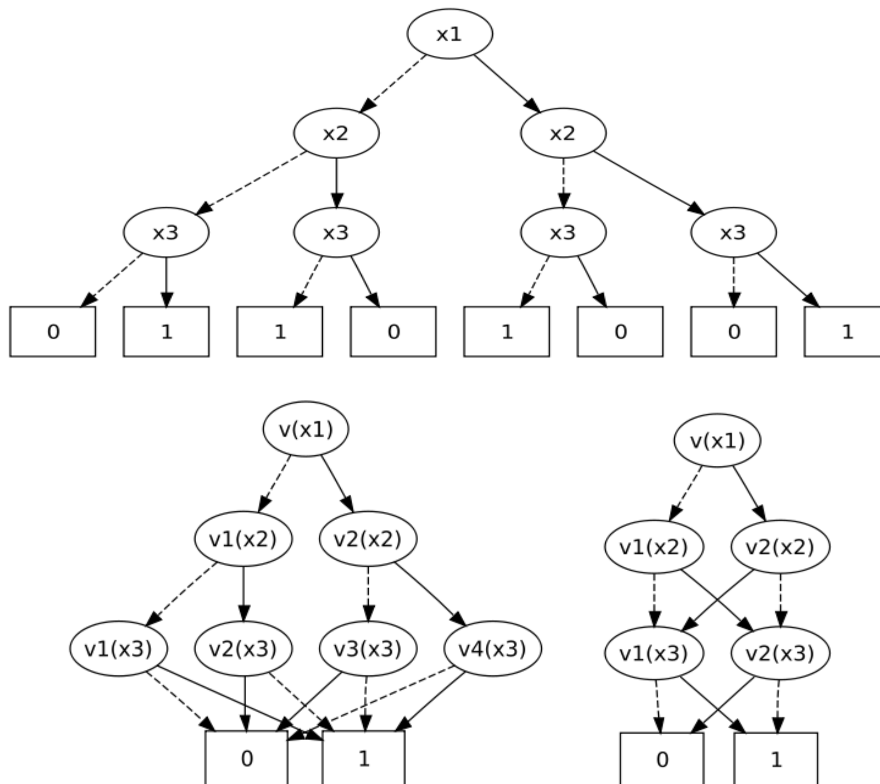


Рис. 1. Процесс перехода от дерева решений линейной функции к ее ROBDD

Выдающимся достижением работы [3] можно считать семейство приведенных в ней алгоритмов «манипуляций с булевыми функциями». Прежде всего, это алгоритмы, реализующие операции над BDD, являющиеся интерпретациями операций над соответствующими булевыми функциями. Далее мы кратко поясняем суть основных алгоритмов работы с BDD на примерах.

Первый алгоритм, названный в [3] BUILD, – это алгоритм прямого преобразования дерева решений рассматриваемой функции в ROBDD. Поскольку при этом предполагается полный обход дерева решений, сложность Build, очевидно, есть экспонента от числа булевых переменных.

Существенно более важные в практическом отношении результаты можно получать при помощи алгоритма APPLY. Данный алгоритм позволяет по паре ROBDD B_{f^1}, B_{f^2}, представляющих булевы функции f^1 и f^2 над множеством булевых переменных $X = \{x_1, ..., x_n\}$, построить ROBDD, представляющую булеву функцию $f^3 = f^1 * f^2$, где «$*$» – произвольная бинарная логическая связка. Порядок означивания переменных в B_{f^1} и B_{f^2} при этом обязательно должен совпадать. При выполнении этого требования сложность процедуры APPLY ограничена сверху величиной $O(|B_{f^1}| \cdot |B_{f^2}|)$; через $|B|$ здесь и далее обозначается размерность ROBDD B, т. е. число вершин в B. Факт построения ROBDD

B_{f^3} при помощи алгоритма APPLY по известным ROBDD B_{f^1} и B_{f^2} обозначается следующим образом:

$$B_{f^3} = \text{APPLY}\left(B_{f^1} * B_{f^2}\right).$$

Основа алгоритма APPLY заключается в одновременном прохождении обеих ROBDD в соответствии с выбранным порядком означивания переменных. Такому обходу B_{f^1} и B_{f^2} можно поставить в соответствие дерево $T_*\left(f^3\right)$, представляющее функцию f^3 (однако это дерево существенно компактнее дерева решений данной функции!). Каждая вершина в $T_*\left(f^3\right)$ определяется парой «координат» – соответствующими текущими вершинами в B_{f^1} и B_{f^2}. Так как порядок означивания переменных в B_{f^1} и B_{f^2} совпадает, то при построении $T_*\left(f^3\right)$ не происходит возвратов, поэтому число вершин в нем не превосходит величины $|B_{f^1}| \cdot |B_{f^2}|$. После построения дерева $T_*\left(f^3\right)$ оно усекается до ROBDD (соответствующая процедура линейна от размерности $T_*\left(f^3\right)$). На практике применяется более эффективная схема, использующая в своей основе принцип «динамического программирования». В соответствии с такой схемой ROBDD B_{f^3} строится как динамически заполняемая таблица – новая вершина заносится в таблицу лишь тогда, когда таблица не содержит дубликата этой вершины.

Продемонстрируем использование BDD в задачах поиска решений систем логических уравнений. Напомним, что логическими уравнениями называются выражения вида

$$L(x_1, ..., x_n) = 0, \ L(x_1, ..., x_n) = 1,$$

где через $L\left(x_1, ..., x_n\right)$ обозначается формула (исчисления высказываний), задающая некоторую булеву функцию над множеством булевых переменных $X = \left\{x_1, ..., x_n\right\}$. Тот факт, что при подстановке $x_1 = \alpha_1, ..., x_n = \alpha_n, \alpha_i \in \{0, 1\}, i \in \{1, ..., n\}$, формула $L\left(x_1, ..., x_n\right)$ принимает значение $\beta \in \{0, 1\}$ обозначается через $L\left(x_1, ..., x_n\right)|_{(\alpha_1, ..., \alpha_n)} = \beta$ (или более кратко через $L(\alpha_1, ..., \alpha_n) = \beta$). Решить логическое уравнение $L\left(x_1, ..., x_n\right) = \beta$, $\beta \in \{0, 1\}$, означает найти такой набор $\left(\alpha_1, ..., \alpha_n\right)$, $\alpha_i \in \{0, 1\}$, $i \in \{1, ..., n\}$, значений истинности переменных $x_1, ..., x_n$, что $L\left(\alpha_1, ..., \alpha_n\right) = \beta$. Если такой набор существует, то он называется решением рассматриваемого логического уравнения. Системой логических уравнений называется выражение вида

$$\begin{cases} L_1\left(x_1, ..., x_n\right) = \beta_1 \\ .... \\ L_m\left(x_1, ..., x_n\right) = \beta_m \end{cases} \quad (1)$$

при фиксированных значениях булевых констант $\beta_j, j \in \{1, ..., m\}$. Решением системы (1) называется набор значений истинности переменных $\left(x_1, ..., x_n\right)$, являющийся решением всех уравнений системы. Если такого набора не существует, то говорят, что система (1) несовместна. Очевидно, что произвольную систему логических уравнений вида (1) можно эффективно преобразовать к виду

$$\begin{cases} L_1\left(x_1, ..., x_n\right) = 1 \\ .... \\ L_m\left(x_1, ..., x_n\right) = 1 \end{cases} . \quad (2)$$

Система (2) эквивалентна одному логическому уравнению вида

$$L_1(x_1, ..., x_n) \cdot ... \cdot L_m(x_1, ..., x_n) = 1.$$

Введем в рассмотрение булеву функцию $L : \{0, 1\}^n \to \{0, 1\}$, определяемую следующим образом:

$$L(\alpha) = (L_1(x_1,...,x_n) \cdot ... \cdot L_m(x_1,...,x_n))|_\alpha, \ \alpha \in \{0,1\}^n.$$

Назовем данную функцию характеристической функцией системы (2). Пусть B_L – ROBDD-представление функции L. Очевидно, что по B_L за линейное от $|B_L|$ время можно найти некоторое решение системы (2) либо убедиться в ее несовместности. Весь вопрос в том, насколько эффективно можно построить B_L.

Строить B_L можно, применяя алгоритм APPLY к уравнениям и подсистемам (2). Например, следующим образом:

$$B_1 = \text{APPLY}(B_{L_1} \cdot B_{L_2}), B_2 = \text{APPLY}(B_1 \cdot B_{L_3}), ..., B_{m-1} = \text{APPLY}(B_{m-2} \cdot B_{L_m})$$

(логической связкой в данном случае является конъюнкция). Порядок означивания переменных при этом должен быть одинаков для всех рассматриваемых фрагментов. В силу сказанного выше решение системы (2) может быть найдено по $B_{m-1} = B_L$ в общем случае за $O(|B_{m-1}|)$ шагов.

Несмотря на естественность описанного подхода к решению систем логических уравнений, несложно привести примеры ситуаций, когда он неэффективен. Действительно, даже если построены относительно компактные ROBDD-представления характеристических функций всех уравнений системы вида (2), размер итоговой ROBDD может быть экспоненциальным от объема двоичной кодировки исходной системы, например, если каждое очередное применение APPLY дает ROBDD, число вершин в которой примерно в 2 раза больше, чем в предыдущей.

Подобно другим подходам к решению систем логических уравнений, например SAT-подходу (см. [7]), BDD-подход предполагает использование разного рода эвристик. Главным образом, подразумевается эвристический выбор порядка означивания булевых переменных. Различные подходы к этой проблеме анализируются в книге [8].

Интересным направлением, в котором BDD являются очень перспективной структурой данных, являются задачи обращения дискретных функций. Дискретными функциями от n булевых переменных называются любые функции вида $f_n : \{0,1\}^n \to \{0,1\}^*$. Здесь $\{0,1\}^n$ – множество, образованное всевозможными двоичными последовательностями длины n,

$$\{0,1\}^* = \cup_{n \in \mathrm{N}} \{0,1\}^n.$$

Функции вида $f_n : \{0,1\}^n \to \{0,1\}$ называются булевыми. Через $Dom\, f_n \subseteq \{0,1\}^n$ обозначается область определения функции f_n, а через $Ran\, f_n \subset \{0,1\}^*$ –область ее значений. Функцию f_n, для которой $Dom\, f_n = \{0,1\}^n$, назовем всюду определенной или тотальной (см., например [9]). Особый интерес представляют семейства функций вида $\{f_n\}_{n \in N}$, процесс вычисления которых может быть представлен в форме не зависящего от n набора предписаний (программы) для некоторой формальной вычислительной модели. Такого рода функции называются алгоритмически вычислимыми. Далее в качестве формальной модели рассматривается бинарная машина Тьюринга (см., например, [10]). Для алгоритмически вычислимых тотальных функций корректно вводится понятие вычислительной сложности [Там же]. Проблема обращения тотальной алгоритмически вычислимой функции f_n ставится следующим образом. По данному $y \in Ran\, f_n$ требуется найти такой $x \in \{0,1\}^n$, что $f_n(x) = y$. Проблема, представляющая собой универсальную формулировку для обширного класса задач криптоанализа, состоит в обращении тотальных дискретных функций, вычислимых за полиномиальное время.

Используя известные результаты о представлении алгоритмов логическими уравнениями [11] и схемами из функциональных элементов [12], можно показать, что проблема обращения дискретной функции, вычислимой за полиномиальное время, сводится к проблеме построения ROBDD-представления некоторой булевой функции, допускающей полиномиальное по сложности описание в форме многоэтапной композиции более простых булевых функций.

Формальное доказательство данного факта не приводится ввиду громоздкости. Однако соответствующая идея совершенно прозрачна и может быть проиллюстрирована на примере.

Пример 2. Рассмотрим функцию $f:\{0,1\}^2 \to \{0,1\}^2$, которая задана в следующем виде:

$$f(x_1,x_2)=(y_1,y_2);\ y_1 = j(g(x_1,x_2),h(x_1,x_2)),\ y_2 = h(x_1,x_2),$$

$$g(u,v)=u\oplus v, h(u,v)=u\cdot\bar{v}, j(u,v)=(u\to v).$$

Предположим, что требуется решить задачу обращения данной функции «в точке» $(y_1,y_2)=(0,0)$. Представим процесс вычисления функции f в форме схемы из функциональных элементов над множеством $\{\neg,\&,\oplus,\to\}$ (рис. 2).

Обозначим через $B_{x_1\oplus x_2}$ ROBDD-представление функции $x_1\oplus x_2$, через B_{x_1} – ROBDD-представление функции x_1, через $B_{\neg x_2}$ – ROBDD-представление функции $\neg x_2$. Построим следующие ROBDD:

$$B_{\&} = \mathrm{APPLY}\left(B_{x_1}\ \&\ B_{\neg x_2}\right), B_{\to} = \mathrm{APPLY}\left(B_{x_1\oplus x_2}\to B_{\&}\right),\ B=\mathrm{APPLY}\left(\neg B_{\&}\ \&\ \neg B_{\to}\right).$$

Здесь через $\neg B_g$ обозначена ROBDD, представляющая функцию $\neg g$ ($\neg B_g$ получается из B_g в результате перестановки терминальных вершин «0» и «1»). Несложно понять, что любой путь из корня в терминальную вершину «1» в ROBDD B дает решение рассматриваемой задачи.

Элементы архитектуры BDD-решателя систем логических уравнений

Здесь мы описываем архитектуру «типового BDD-решателя», ориентированного на задачи поиска решений систем логических уравнений. Общая схема работы такого решателя включает перечисленные ниже пункты.

1. Система логических уравнений подается на вход решателю в форме текстового описания.

2. Используя данное описание, решатель переводит исходную систему в специальный формат, удобный для последующего построения ROBDD. В этом формате характеристические функции уравнений системы представляются в виде бинарных деревьев (более детальное описание данного этапа приводится ниже).

3. Далее к полученным древовидным представлениям характеристических функций уравнений системы применяется «идеология» последовательного построения ROBDD-представления характеристической функции исходной системы при помощи алгоритма APPLY.

4. На выходе решатель выдает ROBDD характеристической функции исходной системы логических уравнений, а также одно из ее решений.

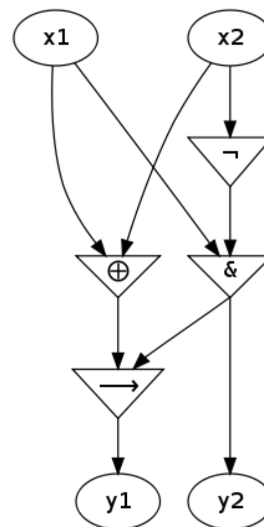
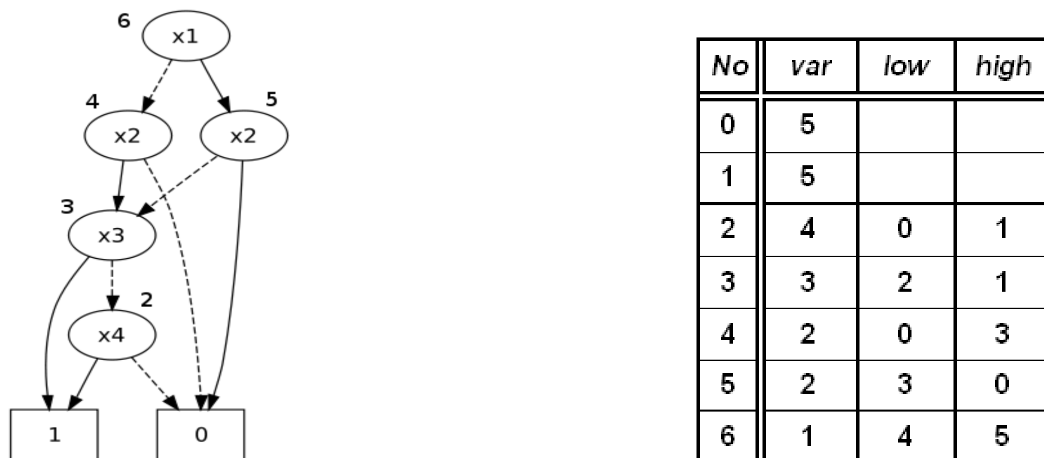


Рис. 2. Представление дискретной функции схемой из функциональных элементов

Следует отметить, что простейшим способом «машинного» представления двоичных диаграмм решений небольшого объема является обычная таблица. Таблица, представляющая ROBDD, состоит из трех столбцов. Каждая строка таблицы соответствует

определенной вершине ROBDD и, таким образом, содержит три позиции: номер переменной данной вершины, номер нижнего потомка и номер верхнего потомка. Вообще говоря, можно использовать различные схемы нумерации вершин BDD. Наиболее естественной представляется нумерация, при которой терминальные вершины получают номера 0 и 1 (в соответствии с представляемыми ими значениями булевой функции). Последующие вершины (от терминальных к корню) получают номера, начиная с 2.

Пример 3. На рис. 3 приведена ROBDD для функции $(x_1 \oplus x_2) \wedge (x_3 \vee x_4)$ с порядком переменных $x_1 \prec x_2 \prec x_3 \prec x_4$ и представляющая ее в памяти ЭВМ таблица.



No	var	low	high
0	5		
1	5		
2	4	0	1
3	3	2	1
4	2	0	3
5	2	3	0
6	1	4	5

Рис. 3. Табличное представление ROBDD

Таблицу, которая представляет ROBDD так, как в примере 3, далее называем таблицей T. В тех случаях, когда ROBDD является результатом работы некоторого алгоритма, оперирующего либо с рассматриваемой булевой функцией, либо с несколькими ROBDD, помимо таблицы T используется хеш-таблица, которую будем называть таблицей H. Данная таблица является хеш-образом T. Хеш-таблица H позволяет ускорять процедуру построения итоговой ROBDD за счет возможности быстрой проверки наличия в T вершины, которую необходимо добавить.

Представление уравнений рассматриваемой системы также характеризуется рядом особенностей. Как было сказано выше, весьма привлекательным форматом представления характеристических функций логических уравнений являются бинарные деревья. На рис. 4 приведено древовидное представление характеристической функции уравнения $(x_1 \oplus (x_2 \vee x_3)) \wedge x_4 = 1$.

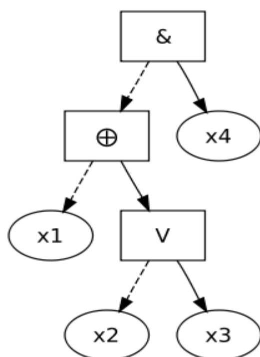


Рис. 4. Древовидное представление характеристической функции

Представления такого рода дают ряд преимуществ. Они удобны как для эвристических алгоритмов получения «оптимального» порядка означивания переменных, так и для базовых алгоритмов работы с BDD (алгоритмы BUILD и APPLY).

Одним из важнейших элементов всякого BDD-решателя является механизм выбора порядка означивания переменных, поскольку данный фактор зачастую оказывает очень существенное (а иногда решающее) влияние на размер итоговой ROBDD. Выбор порядка означивания переменных, как правило, производится в соответствии с некоторыми эвристиками. Ниже приведена

общая «идеология» такого рода эвристик: эвристика выстраивает «иерархию влияния» одних переменных на другие, анализируя статистику появления переменных в уравнениях исходной системы.

1. Производится обход всех деревьев, представляющих характеристические функции уравнений системы, с присвоением переменным весов (натуральных чисел), учитывающих степень взаимного влияния переменных (см. пример 4).

2. Производится сортировка переменных по убыванию весов. Сформированный порядок – это порядок означивания переменных при построении ROBDD-представления характеристической функции системы.

Пример 4 (формирование порядка на основе оценки степени «взаимного влияния» переменных). Для произвольной переменной x, встречающейся в некотором уравнении системы, ее вес вычисляется как функция от соответствующего дерева T: инициальное значение веса равно 0, затем в процессе обхода дерева T данное значение изменяется при помощи некоторой рекурсивной процедуры. Примером задания подобного рода процедуры является следующая формула:

$$wt_T(x) := wt_T(x) + g_T(d_T(x)).$$

В данной формуле фигурирует функция $g_T(\cdot)$, аргументом которой является «глубина расположения» переменной x в дереве T. Обычно функция $g_T(\cdot)$ подбирается эвристически на основе некоторых «разумных предположений».

Для большей иллюстративности сказанного рассмотрим следующую систему из двух логических уравнений:

$$\begin{cases} (x_1 \oplus (x_2 \vee x_3)) \wedge x_4 = 1 \\ x_2 \oplus (x_1 \vee (x_3 \wedge x_4)) = 1 \end{cases}.$$

Дерево T_1, «представляющее» первое уравнение, приведено на рис. 4. В данном дереве $d_{T_1}(x_4) = 1$, $d_{T_1}(x_1) = 2$, $d_{T_1}(x_2) = d_{T_1}(x_3) = 3$. Допустим, что с использованием некоторой функции $g_{T_1}(\cdot)$ вычислены следующие значения весов переменных, входящих в первое уравнение: $wt_{T_1}(x_1) = 597, wt_{T_1}(x_2) = 594, wt_{T_1}(x_3) = 594, wt_{T_1}(x_4) = 600$ (если бы система состояла только из первого уравнения, то следовало бы строить ROBDD, используя следующий порядок означивания переменных: $x_4 \prec x_1 \prec x_2 \prec x_3$). Дерево T_2, «представляющее» второе уравнение рассматриваемой системы, выглядит следующим образом (рис. 5).

Предположим, что в соответствии с описанной процедурой найдены значения весов $wt_{T_2}(x_1) = 597, wt_{T_2}(x_2) = 600, wt_{T_2}(x_3) = wt_{T_2}(x_4) = 594$. В качестве итоговых значений весов переменных могут быть взяты суммы весов по отдельным деревьям ($wt(x) = \sum_T wt_T(x)$). В рассматриваемом примере имеем:

$$wt(x_1) = 1194, wt(x_2) = 1197,$$
$$wt(x_3) = 1188, wt(x_4) = 1197.$$

Откуда следует, что порядок означивания переменных при построении ROBDD-представления характеристической функции рассматриваемой системы в соответствии с описанным подходом должен быть следующим: $x_2 \prec x_4 \prec x_1 \prec x_3$.

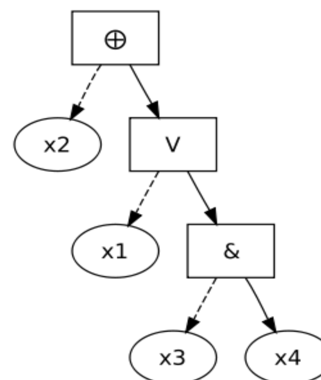


Рис. 5. «Представляющее» дерево функции

Предположим, что выбран некоторый порядок означивания переменных (например, в соответствии с представленной выше эвристикой) – $x_{i_1} \prec x_{i_2} \prec \ldots \prec x_{i_{n-1}} \prec x_{i_n}$. Число $r \in \{1,\ldots,n\}$ называем индексом переменной x_{i_r} относительно вы-

бранного порядка. На следующем шаге рассматриваемая система логических уравнений разбивается на подмножества, называемые слоями. Первый слой образован всеми уравнениями системы, в которые входит переменная x_{i_1}. Второй слой образован всеми оставшимися уравнениями, содержащими переменную x_{i_r}, причем x_{i_r} – переменная наименьшего индекса по всем уравнениям, не входящим в первый слой. И так далее. Пусть R – число определяемых описанным образом слоев системы. Очевидно, что $1 \le R \le n$. ROBDD каждого слоя $B_j, j \in \{1,..., R\}$, строится по следующей рекурсивной схеме:

$$B_j = \mathrm{APPLY}\left(B_{j_1} \cdot \mathrm{APPLY}\left(B_{j_2} \cdot ... \cdot \mathrm{APPLY}\left(B_{j_{k-1}} \cdot B_{j_k}\right)\right)\right),$$

где $B_{j_1},..., B_{j_k}$ – ROBDD уравнений, образующих j-й слой. Итоговая ROBDD системы строится по аналогичной рекурсивной схеме:

$$B = \mathrm{APPLY}\left(B_1 \cdot \mathrm{APPLY}\left(B_2 \cdot ... \cdot \mathrm{APPLY}\left(B_{R-1} \cdot B_R\right)\right)\right).$$

Оптимизация представления BDD в памяти ЭВМ

При программной реализации приложений, использующих ROBDD, основным ограничением, с которым сталкивается программист, является объем оперативной памяти компьютера. Использование виртуальной памяти в работе с ROBDD непродуктивно, поскольку эта структура содержит многократные ссылки на свои собственные фрагменты. В результате в том случае, когда объем логического выражения, представляемого в виде ROBDD, превышает объем оперативной памяти, проявляется эффект, известный как *cache trashing* [1]. Данный эффект состоит в том, что очень часто после выгрузки некоторой информации из оперативной памяти на жесткий диск она почти сразу же оказывается необходимой для проведения дальнейших вычислений. Если наблюдать за работой операционной системы, например в Windows Task Manager, то *cache trashing* проявляется в падении загрузки процессора при работе приложения со 100 до 0–10 % (основное время расходуется системой на ввод-вывод при работе с файлом подкачки).

Далее будут рассмотрены некоторые приемы снижения объема памяти, требуемой для представления BDD. Во-первых, речь пойдет о снижении расхода памяти при представлении отдельных узлов (за счет снижения накладных расходов менеджера памяти в расчете на один узел). Во-вторых, будут рассмотрены методики многократного использования одних и тех же областей памяти для записи информации о различных узлах ROBDD.

Эффективное выделение динамической памяти для большого количества блоков одинакового размера. Основной объем памяти, используемой для представления ROBDD, составляют записи, содержащие информацию об узлах ROBDD (далее «записи»). Все записи занимают одинаковый объем, который для 32-разрядного компьютера, как правило, составляет 16 байт (см. далее). В ходе построения ROBDD выполняется многократное выделение и освобождение динамической памяти, используемой для хранения информации об узлах.

Если выделять память под каждую запись с использованием отдельного обращения к системной функции для динамического выделения памяти (new, malloc, GetMem и т. п.), то необходимо учитывать, что вместе с каждым выделяемым блоком потребуется хранить служебную информацию менеджера памяти. К такой информации, как минимум, относятся сведения о размере блока (4 байта для 32-разрядного компьютера), поскольку все современные компиляторы не требуют передачи параметра с размером блока в функцию его освобождения. Размер блока менеджер памяти, как правило, записывает перед самим блоком. Таким образом, реальный расход памяти на выделение 16-байтного блока будет составлять не менее 20 байт. Дальнейшее увеличение этого объема может быть связано с выравниванием размера блока до кратного некоторой величине (составляющей более 4 байт), которое может выполняться некоторыми менеджерами памяти.

[1] См. подробнее: http://sviazexpo.ru/glossary/137540.html

Повышение эффективности динамического выделения памяти для большого числа структур одного размера может быть достигнуто за счет использования собственного менеджера памяти для таких блоков. Такой прием применяется, например, в программных разработках для решения задач вычислительной геометрии (см., например, [13]). Далее следует описание аналогичного подхода в отношении ROBDD (поскольку авторы не обнаружили подобных описаний в доступных источниках).

Предоставление памяти для блоков одинакового размера реализуется посредством типа данных (класса), который будем далее называть TFreeList. При инициализации TFreeList задаются два параметра: размер блока δ и шаг Δ. Блоки размера δ далее будем называть элементами. Величина Δ определяет, на какое число элементов запрашивается память за одно обращение к системному менеджеру памяти. Фрагмент памяти, содержащий Δ элементов (размера δ), далее называем большим блоком. Помимо Δ элементов каждый большой блок содержит 4 байта для организации связного списка больших блоков и, возможно, еще несколько байт для обеспечения выравнивания адресов элементов на границы, кратные некоторой величине, превышающей 4 байта.

Класс TFreeList содержит связный список выделенных больших блоков и связный список свободных элементов (размера δ), находящихся в памяти этих блоков. Процедура выделения памяти из TFreeList возвращает первый элемент из списка свободных элементов, если этот список не пуст, иначе происходит выделение нового большого блока, при этом все его элементы добавляются в список свободных. Процедура освобождения элемента состоит во включении этого элемента в голову списка свободных элементов.

Описанная технология для 16-байтных записей, представляющих узлы ROBDD, позволяет сэкономить от 25 % памяти (в зависимости от вида выравнивания, используемого системным менеджером памяти). Помимо сокращения накладных расходов на выделение одного элемента, использование TFreeList позволяет перечислять все выделенные элементы, быстро освобождать память (не поэлементно, а по списку больших блоков) и т. д. Кроме того, появляется возможность легко обеспечить выравнивание адресов элементов на любую границу b, размер которой кратен размеру блока, за счет добавления в большой блок «страховочных» ($b - 4$) байт (позволяющих при необходимости сдвинуть таблицу элементов блока на адрес, кратный b).

Многократное использование областей памяти для представления различных узлов ROBDD. Далее приведено описание традиционного представления в памяти узла ROBDD.

Структура записи TBDDNode

Поле	Тип	Размер	Описание
VN	uint	2	Номер переменной
RefCnt	uint	2	Число ссылок на запись
$Next_0$	Pointer	4	Переход по [VN] = 0
$Next_1$	Pointer	4	Переход по [VN] = 1
HashNext	Pointer	4	Следующий узел в хэш-таблице узлов BDD

Поле VN содержит номер связанной с узлом логической переменной. Для представления констант можно использовать номер переменной, равный –1. Поле RefCnt используется для реализации разделяемых структур данных, позволяющих использовать узел в различных частях одной или нескольких ROBDD. В RefCnt содержится счетчик ссылок на данную запись, при обнулении которого узел освобождается. Счетчик ссылок работает с насыщением: при достижении максимального значения он фиксируется и перестает изменяться. Таким образом, запись, число ссылок на которую достигает максимального значения, никогда не будет освобождена (предполагается, что это и не потребуется делать, раз уж данный узел настолько «популярен»). Поля $Next_0$ и $Next_1$ содержат ссылки на детей рассматриваемого узла (low и high соответственно). Эти поля рассматриваются в качестве элементов массива Next.

При реализации алгоритмов работы с ROBDD, эксплуатирующих идеологию «динамического программирования», для быстрой проверки наличия некоторой записи используются

хэш-таблицы. При использовании открытого хеширования необходимо включать каждый узел в список, связанный с соответствующей ячейкой хэш-таблицы. Для организации таких списков предназначено поле HashNext.

Все сказанное означает, что при использовании 32-разрядной архитектуры общий размер одной записи с информацией об узле может составлять 16 байт. Данный факт благоприятен с точки зрения эффективности обращения к этим записям и их размещения в памяти, поэтому нежелательно как увеличивать, так и уменьшать этот размер. Данное требование приводит к ограничению на использование в задаче не более 2^{16} переменных (это ограничение согласуется с большинством задач, в которых применение ROBDD представляется реалистичным). При необходимости работы с бóльшим количество м переменных можно «передать» в поле VN часть битов поля RefCnt.

Результатом использования системного менеджера памяти является выравнивание всех адресов выделяемых им блоков на некоторую границу, которая, как правило, кратна 4 байтам. При использовании TFreeList данный параметр можно варьировать, например, осуществляя выравнивание на границу в 8 байт.

Таким образом, при использовании выравнивания адресов записей узлов несколько младших битов (не менее 2-х) в адресе каждого узла ROBDD будут всегда равны 0, что позволяет использовать эти биты для хранения любой дополнительной информации без изменения основных структур данных. Далее этот факт используется в основе технологии представления различных узлов ROBDD посредством многократного использования одной записи.

На рис. 6 приведено схематичное описание адреса записи, в которой содержится информация об узле V . Запись разбита на два фрагмента. В первом фрагменте записана информация об адресе структуры TBDDNode (см. табл. выше). Данную часть адреса назовем *информационной*. Второй фрагмент образован битами, которые в результате выравнивания адреса являются нулевыми. Эту часть адреса записи назовем *избыточной*. Такая организация памяти в принципе позволяет использовать один адрес для хранения информации о 2^d сходных узлах ROBDD (посредством многократных ссылок на этот адрес).

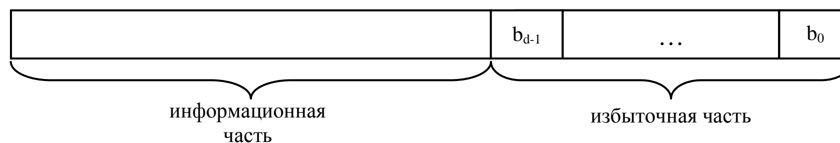
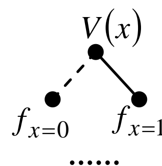


Рис. 6. Адрес записи, содержащей информацию об узле V

Далее мы используем избыточные биты для кодирования информации о так называемых «дополнительных ребрах» и для последующего развития этой идеи. В описании концепции дополнительных ребер мы следуем [8].

Инвертирование ребер. Отметим, что BDD можно рассматривать как графическую интерпретацию рекурсивной формы разложения Шеннона булевой функции. В этом контексте произвольную вершину BDD вида



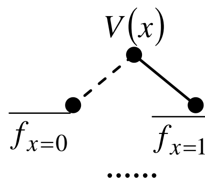
можно рассматривать как корень BDD-представления некоторой булевой функции f , первая итерация разложения Шеннона для которой имеет вид

$$f = \overline{x} \cdot f_{x=0} \vee x \cdot f_{x=1}.$$

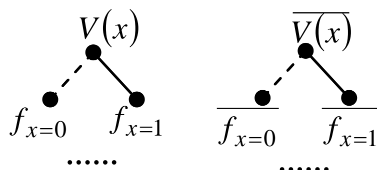
Здесь через $f_{x=\alpha}, \alpha \in \{0,1\}$, обозначена булева функция, которая задается формулой, являющейся результатом подстановки $x = \alpha$ в формулу, задающую f. Используя стандартные операции на булевской решетке, можно показать, что

$$\overline{f} = \overline{x} \cdot \overline{f_{x=0}} \vee x \cdot \overline{f_{x=1}}.$$

Тем самым функцию $\overline{f}$ можно представить в виде следующей BDD:



Данный факт позволяет использовать фрагмент ROBDD, представляющий f, также и для представления $\overline{f}$ – для этого достаточно к записи, представляющий узел $V(x)$, добавить информацию об отрицаниях его детей. Соответствующая реализация достигается при помощи специального флага NegRef, установка которого (т. е. ситуация «NegRef = 1») означает, что вместо выражения, представленного записью, на которую указывает ссылка, следует рассматривать его отрицание. Однако данного приема самого по себе недостаточно, поскольку, как легко понять из вышесказанного, BDD следующего вида



представляют одну и ту же функцию. Данные варианты представления назовем соответственно первым и вторым. Для сохранения однозначности представления в [8] предлагается использовать соглашение о том, что ссылка из Next_1 не должна быть инвертированной. Мы также будем в дальнейшем придерживаться этого соглашения. Таким образом, если в рассмотренном примере в описании вершины, соответствующей Next_1, не установлен флаг NegRef, то выбирается первый вариант представления. В контексте вышесказанного естественным представляется хранить значение флага NegRef в одном из избыточных битов адреса записи информации об узле. Далее полагаем, что для хранения значения NegRef используется младший (нулевой) бит адреса.

Инвертирование low-ребра. Следующий шаг состоит в использовании еще одного избыточного бита. Данный бит можно использовать для хранения информации об инвертировании вершины в Next_0 (соответствующий флаг имеет название «NegNot»). В результате, с одной стороны, поля Next_1 и Next_0 не содержат флагов отрицания, а с другой стороны, все возможные варианты расстановки флагов отрицания у детей рассматриваемой вершины допускают следующее естественное описание при помощи двух флагов инвертирования

Вариант	NegRef	NegNot
(V, A, B)	0	0
(V, A, ¬B)	1	1
(V, ¬A, B)	0	1
(V, ¬A, ¬B)	1	0

Здесь через $(V, ?A, ?B)$ обозначен узел V, детьми которого (соответственно low и high) являются узлы $?A$ и $?B$ («?» означает наличие или отсутствие отрицания). Значение NegNot хранится в первом бите адреса записи информации об узле.

Инвертирование значения переменной. Ввод в рассмотрение флага NegV дает возможность использовать один адрес для описания 8 различных узлов. Флаг NegV позволяет поменять порядок узлов, на которые ссылаются поля Next_0 и Next_1. Таким образом, один и тот же

адрес может использоваться для записи узлов (V, A, B) и (V, B, A). «Наличие флага» NegV (т. е. ситуацию NegV=1) можно рассматривать как признак изменения значения переменной, используемой для выбора следующего узла (вместо $Next_0$ выбираем $Next_1$, а вместо $Next_1$ – $Next_0$). Для обеспечения однозначности представления необходимо определиться с тем, какой из вариантов – (V, A, B) или (V, B, A) – будет подразумеваться при «отсутствии флага», т. е. при NegV = 0. В разработанном пакете для этих целей используется условие «int(A) < int(B)», т.е. low-ребенком V должен быть узел, имеющий меньший адрес.

Для правильной совместной работы с флагами NegNot и NegV необходимо определиться с порядком их применения. В реализованном пакете сначала применяется NegV, а затем к той из ссылок, которая оказывается на месте $Next_0$, может применяться инвертирование в случае установки NegNot.

Значение флага NegV хранится во 2-м бите адреса записи узла. Еще раз обратим внимание, что возможность задействовать описанные три флага достигается посредством выравнивания адресов узлов на 8 байт при помощи TFreeList.

Пример 6 (пример кода). Продемонстрируем работу со всеми тремя флагами инвертирования на примере функции Eval, предназначенной для вычисления значения выражения, представленного рассматриваемой ROBDD, на конкретном векторе значений переменных.

```
function Eval(Root: PBDDNodeF; Vals: TBits): boolean;
var
  N: PBDDNodeF; //Адрес с флагами
  NN: PBDDNode; //Адрес без флагов
  V: Boolean; NegF: integer;
begin
  N := Root;
  repeat //Цикл до достижения константного узла
    NN := PBDDNode(N and not (NegRef or NegNot or NegV)); //Снять флаги, получить адрес
    if NN^.VN<0 then
      break; //Константа найдена
    V := Vals[NN^.VN]; //Значение переменной VN
    if not V and(N and NegNot<>0) then //Если идем по ссылке 0 и установлен признак NegNot,
      N := N xor NegRef; //то инвертируем флаг отрицания
    if N and NegV<>0 then //Установлен флаг инвертирования значения переменной =>
      V := not V; //Обмен Next[0]<->Next[1]
    N := NN^.Next[V] xor (N and NegRef); //Переходим к выбранному потомку,
                                          //переносим отрицание
  until false;
  Result := N and NegRef=0; //Значение константы определяется по флагу NegRef
end ;
```

Экспериментальные результаты

Описанная выше ROBDD-технология решения логических уравнений и обращения дискретных функций тестировалась на задачах, находящих применение в современной криптографии. Криптографическая природа тестов аргументированно обосновывает их сложность, позволяя тем самым адекватно оценивать эффективность разрабатываемых алгоритмов.

Все рассматриваемые далее задачи являются частными случаями общей проблемы обращения дискретных функций и решаются в соответствии с идеологией, иллюстрируемой примером 2. Все вычисления были выполнены на компьютере Intel Core 2 Duo 2,5 GHz с 4 Гб оперативной памяти.

Первый класс тестов образован задачами факторизации натуральных чисел. В данных задачах использовался алгоритм умножения «столбиком» двух беззнаковых целых n-битных чисел. Для решения задачи строился вектор ROBDD-представлений булевых функций, соответствующих битам произведения. Для выбора способа инвертирования флагов использова-

лась условная компиляция, что позволило избежать расхода ресурсов процессора на лишние проверки для неиспользуемых флагов.

Таблица 1

n	Без флагов		NegRef			NegRef + NegNot			NegRef + NegNot + NegV		
	N	*T*	*N*	*T*	%	*N*	*T*	%	*N*	*T*	%
10	171 590	00:01	159 278	00:01	7,18	144 036	00:01	9,57	141 525	00:01	1,74
11	533 404	00:03	492 741	00:03	7,62	442 734	00:03	10,15	437 057	00:03	1,28
12	1 651 206	00:12	1 513 070	00:11	8,37	1 351 071	00:11	10,71	1 338 272	00:11	0,95
13	5 098 916	00:47	4 640 960	00:39	8,98	4 134 307	00:39	10,92	4 106 359	00:39	0,68
14	15 716 731	03:31	14 235 636	02:35	9,42	12 612 490	02:32	11,40	12 550 383	02:30	0,49
15	48 367 364	21:55	43 546 586	12:23	9,97	38 515 423	11:54	11,55	38 376 241	11:34	0,36

В табл. 1 представлены следующие столбцы:

- n – число бит в каждом сомножителе;
- N – суммарное количество узлов во всех выражениях (узлы могут разделяться между разными выражениями);
- T – общее время решения задачи (мин:сек);
- % – сокращение числа узлов N (в % по отношению к предыдущему набору флагов).

Из данной серии экспериментов можно сделать вывод, что использование инвертирования ребер (NegRef) с ростом размерности задачи начинает давать существенную экономию времени. Так, для $n = 15$ время работы сократилось с 21' 55"до 12' 23" Экономия памяти при этом составила от ~7 до ~9 % и также возрастала по мере усложнения задачи.

Инвертирование low ребра также оказалось полезным для рассматриваемой задачи. Время работы в наиболее сложных случаях ($n = 14$ и $n = 15$) дополнительно сократилось, в последнем случае – на полминуты. Расход памяти сократился дополнительно на ~ 10–11 %.

В отличие от двух предыдущих флагов, инвертирование переменной (флаг NegV) оказалось существенно менее эффективным для рассматриваемой задачи. При этом эффективность (сокращение расхода памяти в %) падает по мере увеличения сложность задачи.

Второй класс тестов представлен задачами криптоанализа генератора Геффе (см. [16]). Данный генератор является криптографически слабым, что подтверждается многочисленными успешными атаками в его отношении (см., например [17]). BDD-подход также оправдал себя в применении к криптоанализу генератора Геффе. Для отслеживания динамики выигрыша, получаемого за счет использования флагов, были построены тесты с несколькими угаданными (подставленными) битами секретного ключа генератора (инициализирующей последовательности).

В табл. 2 представлены следующие столбцы:

- n – число угаданных битов ключа;
- N – пиковое использованием памяти (максимальное количество узлов, которое одновременно выделяется в ходе решения задачи);
- N_App – максимальное количество элементов в хэш-таблице алгоритма Apply;
- T – общее время решения задачи (мин:сек);
- % – сокращение числа узлов N (в % по отношению к предыдущему варианту).

Результаты эксперимента демонстрируют, что использование инвертирования ребер и инвертирования low-ребра почти не приводит к сокращению количества узлов. Однако при этом сокращается нагрузка на хэш-таблицу алгоритма Apply и общее время работы алгоритма. Инвертирование переменной, в отличие от предыдущего примера, позволяет получить наибольшую экономию памяти среди предложенных подходов (около 3 %).

В третьем классе тестов BDD-подход использовался в решении задач криптоанализа генератора ключевого потока системы шифрования A5/1 [18]. Данный генератор является весьма стойким (шифр A5/1 используется в шифровании GSM-трафика) и в полной версии не поддается криптоанализу с применением BDD. Однако подстановка значительного числа битов секретного ключа позволяет получать тесты, которые, с одной стороны, оказываются вполне

Таблица 2

n	Без флагов			NegRef				NegRef + NegNot				NegRef + NegNot + NegV			
	N	N_App	T	N	N_App	T	%	N	N_App	T	%	N	N_App	T	%
6	2 445 264	2 416 128	00:12	2 445 056	944 958	00:10	0,01	2 445 051	944 958	00:10	0,00	2 368 076	1 082 096	00:09	3,15
5	3 171 440	3 183 310	00:16	3 171 223	1 230 284	00:14	0,01	3 171 218	1 230 284	00:13	0,00	3 068 895	1 397 359	00:13	3,23
4	4 027 155	4 047 427	00:23	4 026 929	1 618 072	00:17	0,01	4 026 924	1 618 072	00:18	0,00	3 895 682	1 831 530	00:17	3,26
3	4 802 276	4 780 735	00:30	4 802 041	1 903 579	00:22	0,00	4 802 036	1 903 579	00:22	0,00	4 647 058	2 159 802	00:21	3,23
2	14 151 700	12 876 002	02:57	14 151 455	5 194 380	01:40	0,00	14 151 450	5 194 380	01:41	0,00	13 752 050	5 975 381	01:38	2,82
1	18 275 740	16 123 473	04:58	18 275 489	6 446 381	02:25	0,00	18 275 484	6 446 381	02:24	0,00	17 771 189	7 408 339	02:19	2,76
0	23 563 501	19 973 647	07:40	23 563 244	7 867 643	03:22	0,00	23 563 239	7 867 643	03:24	0,00	22 927 879	9 060 643	03:18	2,70

Таблица 3

n	Без флагов			NegRef				NegRef + NegNot					NegRef + NegNot + NegV				
	N	N_App	T	N	N_App	T	%	N	N_App	T	%	%БФ	N	N_App	T	%	% БФ
30	976 720	241 137	00:51	931 742	91 031	00:41	4,61	930 716	91 898	00:42	0,11	4,71	861 494	86 884	00:41	7,44	11,80
29	1 041 960	245 993	01:04	1 005 596	94 653	00:51	3,49	1 004 289	94 717	00:51	0,13	3,62	943 338	84 868	00:50	6,07	9,47
28	2 124 135	598 605	02:03	1 982 993	229 753	01:35	6,64	1 973 164	232 696	01:36	0,50	7,11	1 909 389	196 432	01:35	3,23	10,11
27	3 281 482	942 274	03:05	3 063 779	303 316	02:23	6,63	3 046 203	305 228	02:24	0,57	7,17	2 870 153	296 644	02:25	5,78	12,53
26	4 934 248	1 062 219	04:37	4 985 852	332 118	03:47	-1,05	4 983 733	333 433	03:48	0,04	-1,00	4 798 937	292 885	03:46	3,71	2,74
25	9 885 730	1 978 065	09:44	9 887 713	752 803	07:59	-0,02	9 884 824	1 978 065	08:02	0,03	0,01	9 637 793	953 471	08:03	2,50	2,51
24	16 985 922	3 271 390	17:59	17 365 635	1 653 381	14:35	-2,24	17 278 575	1 653 391	14:42	0,50	-1,72	16 839 943	1 648 750	14:40	2,54	0,86
23	32 377 289	8 017 179	40:34	32 762 253	3 384 168	32:31	-1,19	32 756 756	3 384 275	32:45	0,02	-1,17	31 917 348	3 393 544	32:50	2,56	1,42
22	53 421 927	12 071 588	1:21:06	54 180 872	4 696 790	1:03:07	-1,42	54 169 901	4 702 504	1:03:08	0,02	-1,40	52 639 639	4 046 434	1:02:07	2,82	1,46

по силам BDD-решателю, а с другой стороны, позволяют исследовать эффективность применения различных подходов к оптимизации представления BDD в памяти.

В табл. 3 используются те же наименования столбцов, что и в предыдущем примере с генератором Геффе, но добавлено два столбца «% БФ» – сокращение расхода памяти в процентах по отношению к расходу памяти при отсутствии флагов инвертирования. Для данной задачи наилучшую экономию памяти в большинстве случаев дает инвертирование переменной, а наименее эффективным оказывается инвертирование low-ребра.

Некоторое увеличение расхода памяти в результате применения флагов NegRef и NegNot объясняется использованием при решении данных примеров (криптоанализ ослабленных вариантов A5/1) технологии «упрощения выражения на допустимом множестве» [19]. Грубо говоря, использование флагов NegRef и NegNot может в некоторых случаях «помешать» эффективному упрощению выражения на допустимом множестве.

Тем не менее, следует отметить, что в рассматриваемых задачах криптоанализа A5/1 совместное применение всех трех флагов привело к экономии памяти во всех тестах. Также во всех случаях сократилось время работы алгоритма.

Заключение

В статье исследованы вопросы программной реализации BDD-подхода к решению логических уравнений и задач обращения дискретных функций. Данный подход представляется весьма перспективным в силу возможности рассмотрения ROBDD как наиболее экономной формы представления булевых функций в классе графов специального вида.

Основное внимание в статье уделено проблемам оптимизации представления ROBDD в оперативной памяти компьютера. Предложена технология управления памятью, которая, с одной стороны, позволяет сократить накладные расходы на динамическое выделение памяти, а с другой – дает возможность такой организации памяти, при которой для описания различных узлов ROBDD используются одни и те же записи. Данный факт приводит к существенной экономии памяти при построении ROBDD. Рассмотрены различные детализации этой идеи.

Фигурирующие в последнем пункте тестовые примеры не были подобраны специально и рассматривались в хронологическом порядке выполнения вычислительных экспериментов с использованием библиотеки, реализованной для работы с ROBDD. Тем не менее данные примеры наглядно демонстрируют, что каждый из предложенных в работе подходов к оптимизации памяти может оказаться эффективным при решении некоторой задачи, структура логических выражений которой дает этому подходу возможность «проявить себя».

Список литературы

1. *Lee C. Y.* Representation of Switching Circuits by Binary-Decision Programs // Bell Systems Technical Journal. 1959. Vol. 38. P. 985–999.

2. *Akers S. B.* Binary Decision Diagrams // IEEE Transactions on Computers. 1978. Vol. 27. № 6. P. 509–516.

3. *Bryant R. E.* Graph-Based Algorithms for Boolean Function Manipulation // IEEE Transactions on Computers. 1986. Vol. 35. № 8. P. 677–691.

4. *Кларк Э. М., Грамберг О., Пелед Д.* Верификация моделей программ: Model Checking. М.: МЦНМО, 2002. 416 с.

5. *Ubar R.* Test Generation for Digital Circuits Using Alternative Graphs (in Russian) // Proc. Tallinn Technical University. 1976. № 409. P. 75–81.

6. *Яблонский С. В.* Введение в дискретную математику. М.: Наука, 1986. 384 с.

7. *Семенов А. А., Беспалов Д. В.* Технологии решения многомерных задач логического поиска // Вестн. Томск. гос. ун-та. 2005. Приложение. № 14. С. 61–73.

8. *Meinel Ch., Theobald T.* Algorithms and Data Structures in VLSI-Design: OBDD-Foundations and Applications. Berlin: Springer-Verlag, 1998. 267 p.

9. *Катленд Н.* Вычислимость. Введение в теорию рекурсивных функций. М.: Мир, 1983. 256 с.

10. *Гэри М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982. 416 с.

11. *Cook S. A.* The Complexity of Theorem-Proving Procedures // Proc. 3rd Ann. ACM Symp. on Theory of Computing, ACM. Ohio, 1971. P. 151–159. [Перевод: *Кук С. А.* Сложность процедур вывода теорем. Кибернетический сборник. Новая серия. Вып. 12. М.: Мир, 1975. С. 5–15]

12. *Karp R. M., Lipton R. J.* Some Connections between Nonuniform and Uniform Complexity Classes // Proc. of the 12th ACM Symposium on Theory of Computing. 1980. P. 302–309.

13. *Fortune S. J.* A Sweepline Algorithm for Voronoi Diagrams // Algorithmica. 1987. № 2. P. 153–174.

14. *Игнатьев А. С., Семенов А. А., Хмельнов А. Е.* Решение систем логических уравнений с использованием BDD // Вестн. Томск. гос. ун-та. 2006. Приложение. № 17. С. 25–29.

15. *Menezes A., Van Oorschot P., Vanstone S.* Handbook of Applied Cryptography. CRC Press, 1996. 657 p.

16. *Семенов А. А.* Логико-эвристический подход в криптоанализе генераторов двоичных последовательностей // Тр. междунар. науч. конф. ПАВТ'07. Челябинск: Изд-во ЮУрГУ, 2007. Т. 1. С. 170–180.

17. *Алферов А. П., Зубов А. Ю., Кузьмин А. С., Черемушкин А. В.* Основы криптографии. М.: Гелиос АРВ, 2002. 480 с.

18. *Shiple T. R., Hojati R., Sangiovanni-Vincentelli A. L., Brayton R. K.* Heuristic Minimization of BDDs, Using Don't Cares // University of California, Berkeley. Technical Report No. UCB/ERL M93/58. 1993.

Материал поступил в редколлегию 10.09.2009

A. E. Khmelnov, A. S. Ignatjev, A. A. Semenov

BINARY DECISION DIAGRAMS IN LOGICAL EQUATIONS AND PROBLEMS OF DISCRETE FUNCTIONS INVERSION

The paper addresses software implementation of one approach to the problems of discrete functions inversion. Such an approach is based on the technique representing boolean functions in the form of binary decision diagrams (BDD). We propose new methods of memory usage optimization when working with BDD. The described technology is tested on some cryptanalysis problems.

Keywords: binary decision diagrams, logical equations, discrete functions, cryptanalysis.