УЧРЕЖДЕНИЕ РОССИЙСКОЙ АКАДЕМИИ НАУК
ИНСТИТУТ ДИНАМИКИ СИСТЕМ И ТЕОРИИ УПРАВЛЕНИЯ
СИБИРСКОГО ОТДЕЛЕНИЯ РАН

На правах рукописи

Игнатьев Алексей Сергеевич

МЕТОДЫ ОБРАЩЕНИЯ ДИСКРЕТНЫХ ФУНКЦИЙ С ПРИМЕНЕНИЕМ ДВОИЧНЫХ РЕШАЮЩИХ ДИАГРАММ

05.13.18 – Математическое моделирование, численные методы
и комплексы программ

ДИССЕРТАЦИЯ
на соискание ученой степени
кандидата физико-математических наук

Научный руководитель
к. т. н., доцент
Семенов Александр Анатольевич

Иркутск – 2010

Содержание

Введение

Актуальность работы. В последние годы неуклонно растет интерес к дискретным моделям в различных областях информатики и кибернетики. Данный класс моделей чрезвычайно широк и включает модели безопасности компьютерных систем, модели процессов передачи и защиты информации, а также различные автоматные модели. К последним можно отнести автоматные сети, спектр применения которых варьируется от теории принятия решений до компьютерной биологии. Для численного исследования дискретных моделей далеко не всегда успешны «традиционные» методы, оперирующие с действительными числами. Во многих случаях получаемые такими методами результаты являются весьма грубыми приближениями и могут не удовлетворять требуемым критериям точности. Обширный класс дискретных моделей, тем не менее, допускает точные алгоритмы поиска решений. К данному классу относятся, в частности, модели, поведение которых может быть описано алгоритмически вычислимыми дискретными функциями, то есть функциями, преобразующими двоичные слова в двоичные слова. В задачах компьютерной безопасности и при исследовании автоматных моделей одной из наиболее часто возникающих является проблема обращения дискретной функции, вычислимой детерминированным образом за полиномиальное от длины входа время (то есть по известному алгоритму вычисления функции и известному образу требуется найти некоторый прообраз). В контексте данной постановки можно, например, рассматривать подавляющее большинство задач криптоанализа. Используя идеи С. А. Кука (изложенные им еще в 1971г.), можно строго доказать эффективную сводимость проблем обращения полиномиально вычислимых дискретных функций к задачам поиска решений логических (булевых) уравнений. Причем, в конечном счете, возможен эффективный переход к одному уравнению вида КНФ=1 (КНФ — конъюнктивная нормальная форма).

Задачи поиска решений логических уравнений дают пример проблем, чья аргументированная вычислительная сложность (в общей постановке

они NP-трудны) не является препятствием для появления новых методов и алгоритмов. Об актуальности данной проблематики свидетельствует хотя бы факт издания в Нидерландах специализированного журнала «JSAT» (см. `http://jsat.ewi.tudelft.nl/`), подавляющее большинство статей в котором посвящено SAT-задачам (SAT-задачами называются задачи поиска решений логических уравнений вида КНФ=1).

На данный момент можно выделить (в качестве наиболее успешных) два общих подхода к поиску решений логических уравнений, высокая эффективность которых делает их широко используемыми. В первую очередь речь идет о SAT-подходе, в основе которого лежат процедуры приведения разнородных систем логических уравнений к уравнениям вида «КНФ=1». Второй класс методов базируется на использовании двоичных решающих диаграмм (BDD), а точнее сокращенных упорядоченных BDD или «ROBDD» — формата представления булевых функций в виде направленных помеченных графов специального вида. О популярности «ROBDD-подхода» в задачах синтеза и верификации дискретных систем говорит тот факт, что на протяжении ряда лет ключевая статья по алгоритмике двоичных решающих диаграмм (R. Bryant, [1]) лидировала в рейтинге цитируемости международной системы мониторинга научных публикаций «Citeseer» (см. `http://citeseer.ist.psu.edu/source.html`).

Преимуществом SAT-подхода является простота базовых структур данных и возможность их эффективного представления и оперирования с ними в памяти ЭВМ. Один из главных недостатков SAT-подхода состоит в фактической неполноте современных SAT-решателей, проявляющейся на аргументированно трудных тестах (например, на задачах криптоанализа ряда систем шифрования). Следует отметить, что наиболее быстрые (по результатам специализированных конкурсов) алгоритмы решения SAT-задач эксплуатируют идеологию «накопления ограничений» (подобно методам отсечений в целочисленном линейном программировании). Для хранения ограничений и быстрой работы с ними используется только оперативная память компьютера. При переполнении памяти возникает необходимость «чистки» баз накопленных ограничений, то есть удаления некоторых ограни-

чений (дизъюнктов), признаваемых нерелевантными. Все известные практические оценки релевантности имеют характер эвристик. Данный факт не дает гарантии того, что алгоритм в дальнейшем не породит уже отброшенные ограничения. В такого рода ситуациях возможно зацикливание алгоритма (потеря полноты).

Основной недостаток ROBDD-подхода состоит в том, что даже в отношении простых в контексте SAT-подхода логических уравнений можно строго показать общую неэффективность в применении к ним ROBDD-подхода. Главное преимущество ROBDD-подхода состоит в том, что произвольная ROBDD единственным образом (с точностью до изоморфизма графов) представляет соответствующую булеву функцию. Тем самым, ROBDD можно рассматривать как некоторую компактную форму представления булевых функций в специальном классе графов.

Исходя из всего вышесказанного, актуальными представляются проблемы разработки, обоснования эффективности и программной реализации методов решения логических уравнений, в которых сочетались бы преимущества скорости обработки данных, присущей SAT-подходу, и компактности представления данных, присущей ROBDD-подходу. Решатели, базирующиеся на таких методах, могут использоваться при исследовании широкого класса дискретных моделей, поведение которых допускает описание полиномиально вычислимыми дискретными функциями.

Цель и задачи исследования. Целью диссертационной работы является разработка и практическая реализация гибридного (SAT+ROBDD)-метода решения систем логических уравнений, кодирующих проблемы обращения полиномиально вычислимых дискретных функций.

Для достижения указанной цели ставятся и решаются перечисленные ниже задачи.

1. Разработать стратегию логического вывода, в которой SAT-подход сочетается с ROBDD-подходом в следующем смысле: DPLL (как базовый алгоритм решения SAT) порождает массивы ограничений-дизъюнктов, которые в дальнейшем хранятся и обрабатываются в форме

ROBDD-представлений соответствующих булевых функций (данный шаг предполагает сокращение объема оперативной памяти, используемой для хранения накопленных ограничений); разработать и реализовать ROBDD-аналоги основных механизмов вывода, используемых в современных SAT-решателях; строго обосновать корректность и эффективность работы соответствующих процедур.

2. Разработать и программно реализовать ROBDD-решатель систем логических уравнений, использующий новые эвристические алгоритмы.

3. Программно реализовать гибридный (SAT+ROBDD)-решатель логических уравнений, ориентированный на задачи обращения полиномиально вычислимых дискретных функций; разработать параллельные гибридные (SAT+ROBDD)-алгоритмы решения логических уравнений и обращения дискретных функций; программно реализовать разработанные алгоритмы с использованием стандарта MPI (Message Passing Interface); интегрировать все разработанные алгоритмы в программный комплекс.

4. Протестировать построенный программный комплекс на задачах обращения криптографических функций и задачах исследования некоторых автоматных моделей.

Методы и инструменты исследования. Теоретическая часть исследования использует аппарат теории множеств, дискретной математики, теории вычислительной сложности, теории булевых функций, теории параллельных вычислений, а экспериментальная — современные средства разработки программного обеспечения, а также многопроцессорные вычислительные системы.

Научная новизна. Новыми являются все основные результаты, полученные в диссертации, в том числе:

- эвристические алгоритмы решения систем логических уравнений при помощи двоичных решающих диаграмм, использующие декомпозиции исходной системы;

- ROBDD-аналоги базовых алгоритмов и процедур, используемых в современных DPLL-решателях, а также новый алгоритм модификации порядка означивания переменных в ROBDD;

- программный комплекс, включающий ROBDD-решатель систем логических уравнений, а также параллельную и последовательную версии гибридного (SAT+ROBDD)-решателя;

- вычислительные эксперименты, включающие решение задач обращения некоторых криптографических функций и исследование автоматных моделей генных сетей (построенный программный комплекс на тестовых задачах превзошел по эффективности сторонние разработки).

Основные результаты, выносимые на защиту.

1. Метод обращения полиномиально вычислимых дискретных функций, в основе которого лежит гибридный (SAT+ROBDD) логический вывод; строгое обоснование (в форме теорем) математических свойств ROBDD, рассматриваемых в роли баз булевых ограничений; ROBDD-аналоги основных процедур, используемых в нехронологическом DPLL-выводе (правило единичного дизъюнкта, процедура «Clause Learning», процедуры работы с дизъюнктами, использующие идеологию «отсроченных вычислений»).

2. Новые эвристические алгоритмы решения систем логических уравнений, использующие двоичные решающие диаграммы (ROBDD).

3. Программный комплекс, представляющий собой новый гибридный (SAT+ROBDD)-решатель, ориентированный на решение задач обращения дискретных функций и функционирующий в распределенных вычислительных средах (РВС).

4. Результаты численных экспериментов, включающие исследование дискретно-автоматных моделей из компьютерной биологии, а также ре-

шение задач обращения некоторых криптографических функций в РВС.

Достоверность результатов. Достоверность полученных в работе теоретических результатов обеспечивается строгостью производимых математических построений. Корректность алгоритмов и эффективность их практической реализации подтверждаются результатами вычислительных экспериментов.

Соответствие специальности. В диссертации разработан новый вычислительный метод, использующий алгоритмы обработки дискретных данных и применимый к широкому спектру практических задач (тестирование и верификация дискретных управляющих систем, исследование различных дискретно-автоматных моделей, обращение дискретных функций, криптоанализ). Разработанный метод реализован в виде программного комплекса, функционирующего в распределенных вычислительных средах. Комплекс протестирован на аргументированно трудных задачах обращения некоторых криптографических функций.

Теоретическая и практическая значимость работы. Теоретическая значимость работы заключена в возможности применения предложенных методов к исследованию алгоритмических аспектов задач обращения дискретных функций. Практическая значимость состоит в возможности использовать предложенные методы и применяемые в них вычислительные алгоритмы в исследовании широкого класса моделей дискретных систем, поведение которых описывается полиномиально вычислимыми дискретными функциями.

Содержательная часть работы включает введение и три главы.

Первая глава является обзорной и содержит теоретическую базу для последующего материала. В данной главе приведены необходимые сведения из теории дискретных функций, кратко описаны основные алгоритмы решения SAT-задач. Заключительный раздел первой главы посвящен основам теории двоичных решающих диаграмм и их применению к логическим уравнениям и задачам обращения дискретных функций.

Во **второй главе** развивается гибридный (SAT+ROBDD)-подход к решению задач обращения полиномиально вычислимых дискретных функций. Теоретические результаты данной главы дают основу для разработки и программной реализации гибридного (SAT+ROBDD)-решателя логических уравнений.

В **разделе 2.1** детально описана ROBDD-алгоритмика работы с системами логических уравнений произвольного вида. Здесь же приведено семейство новых эвристик и механизмов, позволяющих повысить эффективность процедур построения ROBDD-представлений характеристических функций систем логических уравнений. Перечисленные компоненты в дальнейшем составляют основу архитектуры ROBDD-решателя логических уравнений, используемого в гибридном (SAT+ROBDD)-подходе.

В **разделе 2.2** рассмотрена проблема модификации ROBDD в соответствии с новым порядком означивания переменных. Исследован описанный в литературе подход к этой проблеме, использующий идеологию сортировки «методом пузырька». Предложен новый алгоритм решения данной проблемы, приведена оценка его трудоемкости и обоснование его большей эффективности в сравнении с известными подходами.

В **разделе 2.3** описывается гибридный (SAT+ROBDD)-подход к решению задач обращения полиномиально вычислимых дискретных функций. В основе данного подхода лежит понятие ядра DPLL-вывода и возможности декомпозиционного разбиения решаемой задачи обращения на SAT- и ROBDD- части. Последующий процесс решения — это сочетание нехронологического DPLL-вывода на SAT-части с выводом на ROBDD.

Раздел 2.4 посвящен разработке новых алгоритмов работы с ROBDD как с модифицируемой базой булевых ограничений, накапливаемых в процессе нехронологического DPLL-вывода. Полученные алгоритмы можно рассматривать как ROBDD-аналоги известных из теории решения SAT-задач процедур: правила единичного дизъюнкта, процедуры «Clause Learning», процедуры работы с дизъюнктами, использующие идеи «отсроченных вычислений» (watched literals, head-tail literals). Свойства алгоритмов (в первую очередь, их корректность) обоснованы в форме теорем. Для всех

построенных алгоритмов приведены оценки их трудоемкости.

Третья глава посвящена программной реализации гибридного (SAT+ +ROBDD)-подхода к обращению полиномиально вычислимых дискретных функций.

В **разделе 3.1** описывается архитектура ROBDD-решателя систем логических уравнений, в котором используются предложенные во второй главе эвристические алгоритмы «реорганизации» рассматриваемых систем, а также специальный менеджер памяти, значительно повышающий эффективность процедур работы с оперативной памятью ЭВМ при построении ROBDD. Реализованный в соответствии с описанными принципами ROBDD-решатель был протестирован на задачах исследования дискретно-автоматных моделей генных сетей, используемых в компьютерной биологии.

В **разделе 3.2** описывается программный комплекс, основанный на концепции гибридного (SAT+ROBDD)-подхода к обращению полиномиально вычислимых дискретных функций. В данном комплексе были реализованы все алгоритмы описанные во второй главе. Основной предпосылкой эффективности гибридного (SAT+ROBDD)-подхода является наблюдаемый в экспериментах эффект «ROBDD-сжатия» баз ограничений-дизъюнктов, накапливаемых в процессе нехронологического DPLL-вывода. Разработанный комплекс был реализован с использованием стандарта MPI и функционирует в распределенных вычислительных средах. Его принципиальным отличием от современных решателей SAT-задач, использующих межпроцессорные взаимодействия, является технология обмена независимо накапливаемыми булевыми ограничениями, представляемыми в виде ROBDD. Заключительный пункт данного раздела содержит результаты численных экспериментов, в которых разработанный программный комплекс был протестирован на задачах обращения некоторых криптографических функций.

Апробация работы. Результаты диссертации докладывались и обсуждались на 3-ей Международной научной конференции «Параллельные вычислительные технологии» (Нижний Новгород, 2009 г.); на VII Всерос-

сийской конференции с международным участием «Новые информационные технологии в исследовании сложных структур» (Томск, 2008 г.); на Всероссийской школе-семинаре с международным участием Sibecrypt-09 (Омск, 2009 г.); на VIII и на IX школах-семинарах «Математическое моделирование и информационные технологии» (Иркутск, 2006, 2007 гг.), на ежегодных конференциях из серии «Ляпуновские чтения» (Иркутск, 2006, 2007, 2008, 2009 гг.), а также на научных семинарах Института динамики систем и теории управления СО РАН, научных семинарах кафедры математической информатики Восточно-Сибирской государственной академии образования; научном семинаре лаборатории дискретного анализа Института математики им. С. Л. Соболева СО РАН; научном семинаре кафедры защиты информации и криптографии Томского государственного университета.

Результаты диссертации были получены в процессе исследований по следующим проектам:

- проект СО РАН «Интеллектные методы и инструментальные средства создания и анализа интегрированных распределенных информационно-аналитических и вычислительных систем для междисциплинарных исследований с применением ГИС, GRID и Веб-технологий» 2007–2009 гг.;
- грант РФФИ №07-01-00400-а «Характеризация сложности обращения дискретных функций в задачах криптографии и интервального анализа»;
- грант Президента РФ НШ-1676.2008.1.

Публикации и личный вклад автора. По теме диссертации опубликовано 14 работ. Наиболее значимые результаты представлены в 7 работах. В число указанных работ входят 2 статьи из Перечня ведущих рецензируемых журналов и изданий ВАК РФ (2010 г.), 3 статьи в научных журналах, 2 полных текста докладов в материалах международных конференций.

Результаты, относящиеся к разделу 2.3, получены совместно с научным руководителем Семеновым А. А. и являются неделимыми. Из совместных работ с Беспаловым Д. В., Заикиным О. С., Хмельновым А. Е. в диссертацию включены результаты, принадлежащие лично автору.

Структура работы. Диссертация состоит из введения, трех глав, заключения и списка литературы, из 111 наименований. Объем диссертации — 109 страниц, включая 27 рисунков и 7 таблиц.

Глава 1

Дискретные функции, логические уравнения и двоичные диаграммы решений

1.1. Общие сведения из теории булевых функций. Логические уравнения

К важнейшим элементарным объектам математической логики и дискретной математики (см. [2, 3]) относятся логические (или булевы) переменные и булевы функции. Булевыми переменными называются переменные, принимающие значения из множества {*ложь, истина*}. Далее в соответствии с принятой системой обозначений обозначаем ложь нулем, а истину единицей. Через $\{0,1\}^n$ обозначим множество всевозможных двоичных векторов (слов) длины n. Произвольные функции вида

$$f : \{0,1\}^n \to \{0,1\}, n \in N,$$

называются булевыми функциями или функциями алгебры логики (см. [3–5]).

Очевидно, что произвольная формула исчисления высказываний (ИВ) от n переменных задает некоторую булеву функцию $f : \{0,1\}^n \to \{0,1\}$, определенную всюду на $\{0,1\}^n$. Пусть $L(x_1, \ldots, x_n)$ — произвольная формула ИВ от переменных $x_1, \ldots, x_n$. Выражения вида

$$L(x_1, \ldots, x_n) = \beta, \beta \in \{0,1\}, \tag{1.1}$$

называются логическими (см. [6]) или булевыми (см. [7]) уравнениями. Решение логического уравнения вида (1.1) — это такой набор $(\alpha_1, \ldots, \alpha_n)$, $\alpha_i \in \{0,1\}$, что формула $L(x_1, \ldots, x_n)$ при подстановке $x_i = \alpha_i$, $i \in \{1, \ldots, n\}$, принимает значение β (этот факт обозначается как $L(x_1, \ldots, x_n)|_{(\alpha_1,\ldots,\alpha_n)} = \beta$ или $L(\alpha_1, \ldots, \alpha_n) = \beta$). Если такого набора не существует, то говорят, что уравнение вида (1.1) решений не имеет. Системой логических уравне-

ний называется выражение вида

$$\begin{cases} L_1(x_1,\ldots,x_n) &= \beta_1 \\ & \ldots \\ L_m(x_1,\ldots,x_n) &= \beta_m \end{cases}, \tag{1.2}$$

где $\beta_1 \in \{0,1\},\ldots,\beta_m \in \{0,1\}$, а $L_1(x_1,\ldots,x_n)=\beta_1,\ldots,L_m(x_1,\ldots,x_n)=$ $=\beta_m$ — логические уравнения. Если существует набор значений истинности $(\alpha_1,\ldots,\alpha_n)$ переменных $x_1,\ldots,x_n$, который является решением каждого из уравнений системы (1.2), то система уравнений (1.2) называется совместной, а набор $(\alpha_1,\ldots,\alpha_n)$ называется решением данной системы. Если такого набора не существует, то система логических уравнений называется несовместной.

Пусть $X=\{x_1,\ldots,x_n\}$ — множество булевых переменных. Термы x_i и $\overline{x}_i$, $i \in \{1,\ldots,n\}$, называются литералами над X (через $\overline{x}$ обозначается логическое отрицание x). Литералы x и $\overline{x}$ называются контрарными. Дизъюнкция различных литералов над X, среди которых нет контрарных, называется дизъюнктом над X. Конъюнктивной нормальной формой (далее КНФ) над X называется конъюнкция различных дизъюнктов над X. Пусть $C(x_1,\ldots,x_n)$ — КНФ над множеством булевых переменных $X=\{x_1,\ldots,x_n\}$. Рассмотрим логическое уравнение вида

$$C\,(x_1,\ldots,x_n)=1. \tag{1.3}$$

Если уравнение (1.3) имеет хотя бы одно решение, то КНФ $C(x_1,\ldots,x_n)$ (кратко «C») называется выполнимой, а соответствующие решения (1.3) называются наборами, выполняющими C. В противном случае C называется невыполнимой. Логические уравнения вида «КНФ=1» образуют очень важный в практическом отношении подкласс. Задачи поиска их решений называют также SAT-задачами.

Дискретными функциями называются булевы вектор-функции, то есть функции вида $f_n:\{0,1\}^n \to \{0,1\}^*, n \in N$, где $\{0,1\}^*=\bigcup_{n\in N}\{0,1\}^n$. Через $dom\ f_n \subseteq \{0,1\}^n$ обозначается область определения функции f_n, а через $range\ f_n \subseteq \{0,1\}^*$ — ее область значений. Дискретную функцию f_n назовем всюду определенной, если $dom\ f_n=\{0,1\}^n$.

Рассматривая в качестве вычислительной модели детерминированную машину Тьюринга (или ДМТ) с входным алфавитом $\Sigma = \{0, 1\}$, определим понятие алгоритмической вычислимости дискретной функции (семейства дискретных функций): если для некоторого семейства $\{f_n\}_{n \in N}$ всюду определенных дискретных функций существует программа для детерминированной машины Тьюринга M (ДМТ-программа), вычисляющая любую функцию f_n, то в соответствии с тезисом Черча (см., например, [8]) данное семейство образовано алгоритмически вычислимыми функциями. В этом случае для ДМТ-программы M стандартным образом (см. [9–11]) можно определить вычислительную сложность как функцию от n (длины входа). Далее в обозначении функций f_n индекс n опускаем.

Обозначим через $\Im$ (см. [12]) класс, который образован всеми семействами всюду определенных дискретных функций, вычислимых детерминированным образом за полиномиальное время. Для любой дискретной функции $f \in \Im$ от n переменных можно поставить задачу ее обращения следующим образом: дано двоичное слово $y \in range\ f$, требуется найти такое слово $x \in \{0, 1\}^n$, что $f(x) = y$.

Можно привести многочисленные примеры практически важных задач, которые формулируются в контексте проблемы обращения полиномиально вычислимых дискретных функций ([13], [14], [15], [16]).

Используя идеи С. А. Кука, изложенные им в 1971 году в его знаменитой статье [17] (см. также [18], можно поставить проблему обращения произвольной функции из класса $\Im$ как задачу поиска решений некоторой системы логических уравнений. Именно этот факт утверждается в следующей теореме.

Теорема 1.1 (см. [19]) Существует алгоритм с полиномиально от n ограниченной сложностью, который, получив на входе описание $f \in \Im$ в виде пары (M, n), выдает систему логических уравнений $S\left(x_1, \ldots, x_{q(n)}\right)$ над множеством булевых переменных $X^* = \{x_1, \ldots, x_{q(n)}\}$, $q(n)$ — некоторый полином. Подстановка в $S\left(x_1, \ldots, x_{q(n)}\right)$ вектора $y \in range\ f$ дает совместную систему, из произвольного решения которой можно за линейное время

выделить вектор $x \in \{0,1\}^n : f(x) = y$.

Для решения логических уравнений можно использовать различные подходы. Далее перечислены наиболее разработанные и дающие хорошие результаты на обширных классах тестов.

- Сведение произвольных систем логических уравнений к системам полиномиальных уравнений над полем $GF(2)$ с последующим их решением при помощи разнообразных модификаций алгоритма Бухбергера (см. [20, 21]) или методов, использующих идеи линеаризации (к последним относится, например, метод линеаризационного множества, предложенный и развитый в работах [22–24]);
- Подходы, использующие булеву унификацию (см. [7, 25, 26]);
- SAT-подход, в основе которого лежит техника приведения, вообще говоря, «разнородных» по своей структуре систем логических уравнений к одному уравнению вида «КНФ=1» (см. [12, 15, 19]);
- Подход, использующий в своей основе двоичные решающие диаграммы (Binary Decision Diagrams, BDD, см. [25, 27]).

В настоящей работе, главным образом, рассматривается SAT-подход и подход, использующий двоичные диаграммы решений.

1.2. SAT-подход к задачам обращения дискретных функций (краткое описание)

1.2.1. Основа SAT-подхода

Ядром SAT-подхода, помимо теоремы 1.1, является техника приведения систем логических уравнений (вообще говоря, произвольной природы) к одному уравнению вида «КНФ=1». В основе такой техники лежат т. н. «преобразования Цейтина», введенные Г. С. Цейтиным в 1968 году в [28, 29]. Первоначально преобразования Цейтина определялись и использовались в

отношении формул исчисления высказываний (ИВ) для доказательства их противоречивости. Несложно перенести действие данных преобразований на логические уравнения (см. [30]).

Рассмотрим логическое уравнение

$$F\left(h_1\left(x_1^1,\ldots,x_{r_1}^1\right),\ldots,h_s\left(x_1^s,\ldots,x_{r_s}^s\right)\right). \tag{1.4}$$

Здесь $h_1,\ldots,h_s$ — некоторые (в общем случае сложные) булевы функции.

Положим

$$X=\{x_1,\ldots,x_n\}=\bigcup_{i=1}^{s}\{x_1^i,\ldots,x_{r_i}^i\},$$

таким образом, $F:\{0,1\}^n\to\{0,1\}$. Введем в рассмотрение булеву функцию

$$g_{h_1}\left(x_1^1,\ldots,x_{r_1}^1,u_1\right):g_{h_1}:\{0,1\}^{r_1+1}\to\{0,1\},$$

которая задается формулой ИВ:

$$h_1\left(x_1^1,\ldots,x_{r_1}^1\right)\equiv u_1.$$

Рассмотрим логическое уравнение

$$C\left(g_{h_1}\left(x_1^1,\ldots,x_{r_1}^1,u_1\right)\right)\cdot F\left(u_1,\ldots,h_s\left(x_1^s,\ldots,x_{r_s}^s\right)\right)=1, \tag{1.5}$$

здесь $C\left(g_{h_1}\left(x_1^1,\ldots,x_{r_1}^1,u_1\right)\right)$ — КНФ-представление булевой функции g_{h_1} над $\{x_1^1,\ldots,x_{r_1}^1\}$. Переход от уравнения (1.4) к уравнению (1.5) — это одна итерация преобразований Цейтина применительно к логическим уравнениям.

Теорема 1.2 (см. [30]) Множество решений (1.4) пусто тогда и только тогда, когда пусто множество решений (1.5). Если данные множества не пусты, то существует взаимно однозначное соответствие между ними. Сложность перехода от произвольного решения уравнения (1.5) к соответствующему решению уравнения (1.4) линейна.

При доказательстве теоремы 1.2 строится явное отображение множества решений уравнения (1.5) на множество решений уравнения (1.4). Затем показывается, что данное отображение является сюръекцией и инъекцией одновременно, то есть биективно.

Следующий факт, вытекающий из теорем 1.1 и 1.2, по сути, определяет содержание SAT-подхода к обращению функций из класса $\Im$.

Следствие 1.1 (теорем 1.1 и 1.2, [19]) Проблема обращения произвольной дискретной функции $f \in \Im$ в общем случае за полиномиальное от n время преобразуется в проблему поиска решений уравнения вида «КНФ=1».

В соответствии с данным утверждением общая концепция SAT-подхода к обращению дискретных функций из класса $\Im$ принимает следующий вид.

- Алгоритм M, вычисляющий произвольную функцию f из $\Im$, при помощи преобразований, фигурирующих в доказательствах теорем 1.1 и 1.2, за полиномиальное от n (длины слов на входе M) время преобразуется в КНФ $C\left(x_1, \ldots, x_{q(n)}\right)$, $q(n)$ — некоторый полином.

- В уравнение $C\left(x_1, \ldots, x_{q(n)}\right) = 1$ подставляется известный вектор $y \in range\ f$. Из теорем 1.1 и 1.2 следует, что полученная в результате этого КНФ $C|_y$ является выполнимой, а из выполняющего ее набора можно эффективно выделить компоненты вектора $x \in \{0,1\}^n$ такого, что $f(x) = y$.

- Таким образом, задача обращения произвольной функции $f \in \Im$ в точке $y \in range\ f$ оказывается сведенной к задаче поиска выполняющего набора некоторой выполнимой КНФ.

1.2.2. Основные алгоритмы решения SAT-задач

Следует отметить, что SAT-задачи в общих постановках являются NP-трудными (см. [9, 31]). Данный факт означает, что навряд ли существуют полиномиальные алгоритмы поиска решений SAT-задач в общем случаи. Однако широта спектра их практической применимости делает принципиальной проблему разработки алгоритмов, эффективных на важнейших подклассах SAT-задач. В последние годы наблюдается интенсивный процесс

разработки новых методик решения SAT-задач с последующим применением получаемых алгоритмов к «индустриальным тестам» (главным образом, из области верификации микроэлектронных устройств и программного обеспечения, см. [13, 32–36]).

Программные комплексы, разрабатываемые для решения SAT-задач, называются SAT-решателями. Отметим, что SAT-решатели — это, как правило, довольно многослойные программы, в которых основной (базовый) алгоритм дополняется множеством различных техник, вспомогательных алгоритмов и эвристик. Среди наиболее успешных SAT-решателей можно выделить следующие (по типу базового алгоритма).

1. Решатели, базирующиеся на методе резолюций (см. [37–41]). Данные решатели используются, главным образом, в системах автоматического доказательства теорем логики предикатов первого порядка, а также в логическом программировании (см. [42–44]).

2. Оптимизационные SAT-решатели. В таких решателях SAT-задачи рассматриваются как задачи дискретной или даже непрерывной оптимизации (MAXSAT-алгоритмы, алгоритмы, использующие локальный и глобальный поиск, и др., см. [45–48]). Основной их недостаток в нахождении, как правило, лишь приближенных решений.

3. Решатели, основанные на алгоритме DPLL и его последующих модификациях (см. [49–53]).

В настоящий момент существует достаточно много различных SAT-решателей. Наиболее известны *zChaff*, *Berkmin*, *Jerusat*, *MiniSat* (MiniSat является неоднократным победителем в ежегодном соревновании «SAT-competition»). Подавляющее большинство SAT-решателей, эффективных на обширных базах индустриальных тестов, базируются на алгоритме DPLL. Поскольку настоящая работа ориентирована на активное использование DPLL (в том числе реализованы и различные его модификации), остановимся на этом алгоритме более подробно.

В развитии эффективных алгоритмов, использующих в своей основе DPLL, принято выделять следующие три этапа:

- 1962 г. собственно алгоритм DPLL (бэктрекинг, BCP-стратегия) (см. [49]);
- 1994 г. первые идеи по распараллеливанию алгоритма DPLL (см. [54]);
- 1999 г. алгоритм GRASP (бэкджампинг, CL-процедура, см. [50]);
- 2001–2009 гг. дальнейшие усовершенствования алгоритмов на основе DPLL (эвристики выбора значений угадываемых переменных, процедуры чистки базы конфликтных дизъюнктов, рестарты, быстрые структуры данных и т. д., см. [51–53]); первые параллельные программные реализации (см. [55–62]).

Алгоритм DPLL. В 1962 г. М. Девисом (M. Davis), Дж. Лоджманом (G. Logemann) и Д. Лавлендом (D. W. Loveland) в работе [49] был предложен алгоритм DPLL (в некоторых источниках используется сокращение DLL). Данный алгоритм основан на более ранней процедуре, разработанной (в 1960 г.) М. Дэвисом и Х. Патнемом в [63]. Ниже приведено подробное описание DPLL.

Рассмотрим произвольную КНФ вида

$$C = D_1 \cdot \ldots \cdot D_m,$$

в которой $D_j, j \in \{1, \ldots, m\}$ — это дизъюнкты над множеством булевых переменных $X = \{x_1, \ldots, x_n\}$. Требуется решить SAT-задачу для КНФ C.

Дизъюнкты, входящие в C рассматриваем как логические ограничения, а КНФ — как систему (базу) ограничений-дизъюнктов. В процессе поиска выполняющего набора задействуется процедура угадывания значений переменных из X. Угадыванием (decision assignment) называется последовательность элементарных действий над некоторой переменной $x_i \in \in \{x_1, \ldots, x_n\}$, включающая в себя выбор этой переменной, присвоение ей произвольного значения и подстановку данного значения в КНФ C. После

этого переменную x_i называем назначенной переменной. Все остальные переменные называются неназначенными. Угадывание может осуществляться только из множества неназначенных переменных. Возможна ситуация, в которой ограничения, представляющие собой некоторые дизъюнкты КНФ C, могут не удовлетворять значению истинности назначенных переменных. В таком случае эти дизъюнкты принимают значение «0», и данный факт обозначается как конфликт. Так, дизъюнкт $(x_1 \vee x_4)$ принимает значение «0» в результате присвоений $x_1 = 0, x_4 = 0$, а дизъюнкт $(\overline{x}_2 \vee x_3 \vee \overline{x}_4)$ — в результате присвоений $x_2 = 1, x_3 = 0, x_4 = 1$.

Последовательность угадываний организуется в виде нумерованного списка уровней решения (decision level). Переменная, значение которой угадывается на данном уровне решения, называется переменной решения (decision variable) этого уровня. В случае конфликта происходит так называемый бэктрекинг (backtracking). Процедура бэктрекинга включает в себя откат на один шаг назад и «переключение» значения переменной решения последнего уровня на противоположное. В том случае, если оба варианта присвоения последней назначенной переменной (кроме первой), скажем, $x_{i+1}, i \in \{2, \ldots, n-1\}$, приводят к конфликту, то при бэктрекинге происходит откат еще на один уровень назад, то есть возвращение к моменту выбора значения для переменной $x_i, i \in \{1, \ldots, n-1\}$, с последующим изменением значения x_i на противоположное.

Описанную выше процедуру логического вывода, состоящую из угадываний значений переменных и откатов в случае конфликта, удобно представлять в виде бинарного дерева.

Рассмотрим дизъюнкт $D = (z_{i_1} \ldots, z_{i_k})$, $k \geq 2$, где $z_{i_j}, j \in \{1, \ldots, k\}$ — некоторые литералы. Если произвольным $k-1$ литералам дизъюнкта D присвоить нулевые значения, то D выполним тогда и только тогда, когда оставшийся литерал принимает значение «1». В этом состоит правило единичного дизъюнкта (unite clause). В некоторых ситуациях удачное угадывание значений нескольких переменных может привести к серии последовательных срабатываний правила единичного дизъюнкта. Итеративное применение правила единичного дизъюнкта называется распространением

булевых ограничений (Boolean constraint propagation, далее «BCP-стратегия»). Переменные, выведенные по правилу единичного дизъюнкта, также добавляются в список назначенных.

Алгоритм называется полным, если он всегда (на любом входе) завершается за конечное число шагов, и неполным, если конечность его работы в общем случае не гарантируется. Поскольку результатом работы алгоритма DPLL является либо вывод набора, выполняющего исходную КНФ, либо ситуация, когда оба варианта значений переменной решения первого уровня приводят к конфликтам (данный факт означает, что исходная КНФ невыполнима), то алгоритм DPLL является полным.

Следует отметить, что алгоритм DPLL может быть эффективен на выполнимых КНФ с большим числом выполняющих ее наборов. На невыполнимых КНФ его эффективность в большинстве случаев сопоставима с полным перебором.

Алгоритм GRASP. Этот алгоритм, являющийся усовершенствованием DPLL, был предложен в 1999 г. Дж. П. Маркесом-Сильвой и К. А. Сакалла в работе [50]. Алгоритм GRASP стал родоначальником нового поколения решателей, базирующихся на DPLL. По скорости работы и широте спектра применения эти решатели значительно превзошли «чистый» DPLL. Данный эффект объясняется основным конструктивным отличием GRASP от DPLL — возможностью использования так называемых «глубоких откатов», то есть откатов с уровня решения, на котором произошел конфликт, не на предыдущий уровень, а на более ранний. Эта процедура введена в алгоритме GRASP в качестве альтернативы бэктрекингу и названа нехронологическим бэктрекингом (бэкджампингом, backjumping).

Возможность бэкджампинга в ходе работы алгоритма GRASP обеспечивается CL-процедурой (от англ. «Clause-Learning» см. [50]), которая сохраняет информацию о конфликтах в форме новых ограничений, называемых конфликтными дизъюнктами. Конфликтные дизъюнкты конъюнктивно приписывается к общей базе ограничений. Каждый конфликтный дизъюнкт при этом является логическим следствием исходной КНФ C, рассматриваемой как система (база) ограничений-дизъюнктов.

Построению конфликтного дизъюнкта предшествует анализ специального графа, называемого графом вывода (implication graph). Граф вывода — это ориентированный ациклический помеченный граф, который может быть описан следующим образом:

- вершинам графа вывода приписываются выражения вида $v(x = \alpha)$ — тем самым вершины графа отображают присвоения значений истинности переменным;
- дугам графа вывода приписаны номера дизъюнктов, из которых выводятся соответствующие присвоения;
- уровни решения отделяются друг от друга пунктирной линией;
- вершина имеет входную степень 0, если присвоение было угадано, и ≥ 1, если это присвоение выведено по правилу единичного дизъюнкта;
- конфликт обозначается темной областью.

Работу алгоритма GRASP проиллюстрируем на примере 1.1.

Пример 1.1. Дана КНФ над $\{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}\}$

$$\begin{aligned} C = (\overline{x}_1 \vee x_2) \cdot (\overline{x}_1 \vee x_3 \vee \overline{x}_4) \cdot (\overline{x}_2 \vee x_3 \vee x_5) \cdot (x_4 \vee x_8 \vee \overline{x}_9) \cdot \\ \cdot (\overline{x}_5 \vee x_8 \vee x_9) \cdot (\overline{x}_6 \vee \overline{x}_5 \vee x_7) \cdot (\overline{x}_1 \vee \overline{x}_8 \vee x_{10}) \cdot (x_4 \vee \overline{x}_8 \vee \overline{x}_{10}). \end{aligned} \tag{1.6}$$

Требуется определить выполнимость данной КНФ.

Предположим, что на некотором этапе выполнения алгоритма GRASP граф вывода имеет вид, представленный на рисунке 1.1. Как видно на рисунке, на четвертом уровне решения произошел конфликт. Несложно понять, что к конфликту приводит одновременное присвоение $x_4 = 0$, $x_5 = 1$, $x_8 = 0$, иными словами, если истинна формула $\overline{x}_4 \cdot x_5 \cdot \overline{x}_8$. Отрицание этой конъюнкции дает конфликтный дизъюнкт $(x_4 \vee \overline{x}_5 \vee x_8)$, конъюнктивно приписываемый к исходной базе ограничений. Это действие запрещает в дальнейшем производить указанное выше конфликтное присвоение.

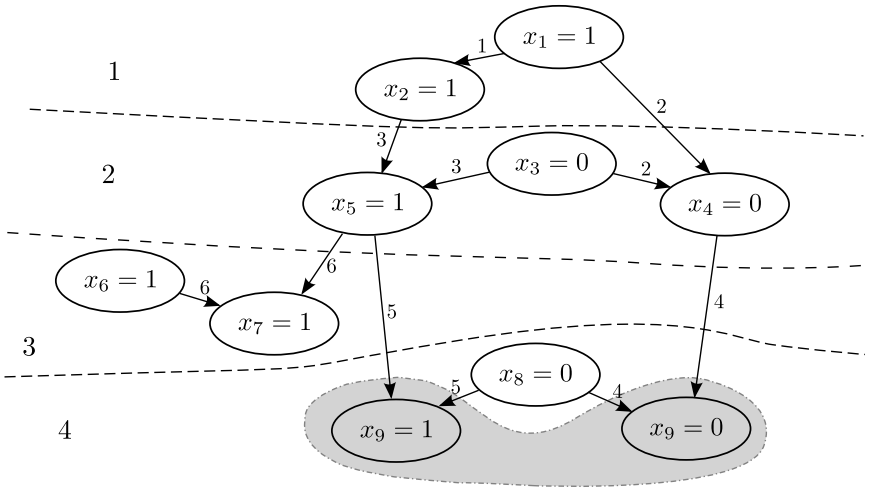


Рис. 1.1. Граф DPLL-вывода для КНФ (1.6)

Несложно убедиться, что КНФ $C' = C \cdot \cdot (x_4 \vee \overline{x}_5 \vee x_8)$ выполнима на тех и только тех наборах значений истинности переменных из X, на которых выполнима исходная КНФ C. Переходим к рассмотрению проблемы выполнимости КНФ C' (соответствующий граф вывода представлен на рисунке 1.2). После того как дизъюнкт $(x_4 \vee \overline{x}_5 \vee x_8)$ конъюнктивно приписан к общей базе ограничений, из него, в силу присвоений $x_4 = 0$, $x_5 = 1$, выводится присвоение $x_8 = 1$, противоположное угаданному на предыдущем этапе на четвертом уровне решения ($x_8 = 0$). Таким образом, произошло «переключение» последнего угадывания и откат на второй уровень решения. Далее снова применяем BCP-стратегию к полученной КНФ.

$$\begin{aligned} C = {} & (\overline{x}_1 \vee x_2) \cdot (\overline{x}_1 \vee x_3 \vee \overline{x}_4) \cdot (\overline{x}_2 \vee x_3 \vee x_5) \cdot \\ & \cdot (x_4 \vee x_8 \vee \overline{x}_9) \cdot (\overline{x}_5 \vee x_8 \vee x_9) \cdot (\overline{x}_6 \vee \overline{x}_5 \vee x_7) \cdot \\ & \cdot (\overline{x}_1 \vee \overline{x}_8 \vee x_{10}) \cdot (x_4 \vee \overline{x}_8 \vee \overline{x}_{10}) \cdot (x_4 \vee \overline{x}_5 \vee x_8). \end{aligned} \tag{1.7}$$

Как видно из рисунка 1.2, очередной конфликт происходит на втором уровне и, следовательно, нет никакого смысла анализировать присвоение, сделанное на третьем уровне. Таким образом, следует откатиться с четвертого уровня решения на второй, минуя третий. Символически данный факт иллюстрирует рисунок 1.3.

Подчеркнем основное отличие GRASP от DPLL. Если DPLL можно рассматривать как направленный обход бинарного дерева поиска, то в

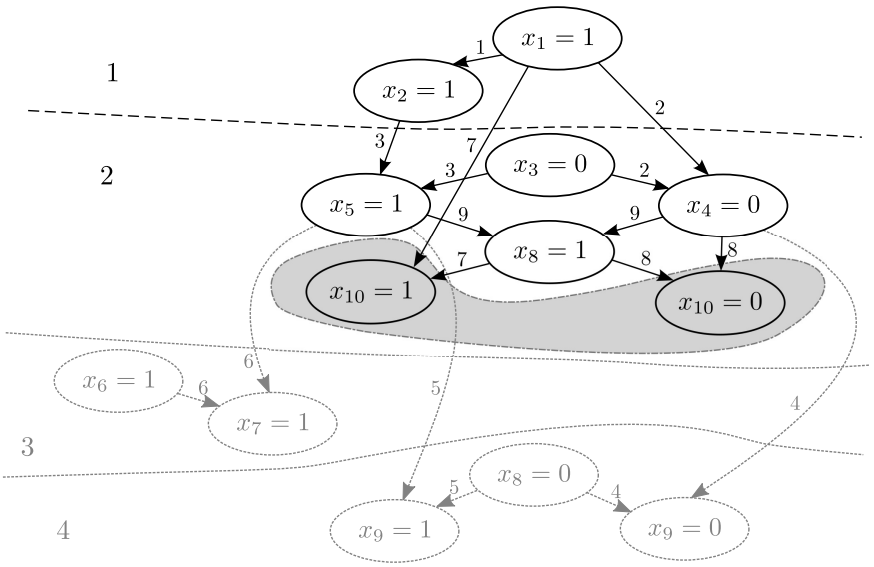


Рис. 1.2. Граф DPLL-вывода для КНФ (1.7)

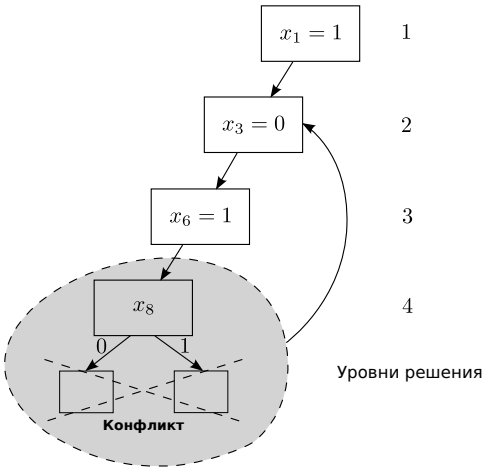


Рис. 1.3. Схема работы бэкджампинга применительно к КНФ (1.6)

GRASP-е информация о пройденных ветвях дерева (соответствующих попаданиям в конфликты) хранится в виде новых ограничений-дизъюнктов. В этом смысле GRASP похож на методы отсечений, широко используемые в целочисленном линейном программировании. Плюс такого подхода состоит в возможности нехронологического бэктрекинга и, соответственно, в более быстром (в некоторых случаях) отсечении тупиковых ветвей дерева поиска. Минус состоит в необходимости задействовать память ЭВМ для хранения новых ограничений. Несложно понять, что GRASP является полным алгоритмом. При этом можно указать семейства противоречий в форме КНФ, на которых GRASP-у для их опровержения потребуется по-

рождать экспоненциальное число конфликтных дизъюнктов (см. [64–67]), таким образом, в общем случае GRASP (как, впрочем, и DPLL) неэффективен. Однако он и его дальнейшие усовершенствования показывают очень хорошие экспериментальные результаты на обширных классах тестов, проистекающих из различных практических приложений.

Дальнейшие усовершенствования алгоритмов на основе DPLL. Ниже приведен краткий обзор технологий, применение которых позволяет существенно ускорять DPLL-вывод.

Одним из важнейших факторов эффективности DPLL-вывода является способ выбора присвоений, ответственных за конфликт, и, соответственно, способ формирования конфликтных дизъюнктов. Для этой цели используются различные подходы. Как показано в работе [51], значимость некоторых ограничений-дизъюнктов может быть обоснована с помощью анализа графа вывода.

Вершину v_1 графа вывода будем называть доминирующей над вершиной v_2, если любой путь из вершины, соответствующей переменной решения текущего уровня, в вершину v_2 проходит через вершину v_1. Вершина графа вывода, доминирующая над обеими конфликтными вершинами, называется UIP (от англ. «Unique Implication Point»). Ближайшая к конфликту UIP называется FUIP (First UIP). В работах [50, 51] были исследованы различные способы построения конфликтных дизъюнктов, базирующихся на анализе UIP-вершин графа вывода. Наиболее популярна по результатам экспериментов FUIP-схема, описание которой приведено ниже.

Вершина, соответствующая переменной текущего уровня решения, обозначена прямоугольником. Овалы за границей текущего уровня обозначают присвоения значений истинности некоторым переменным, имеющиеся до момента означивания переменной решения текущего уровня. Кружки и жирный овал внутри границы текущего уровня обозначают вывод присвоений по правилу единичного дизъюнкта на текущем уровне.

Очевидно, что переменная решения текущего уровня является наиболее удаленной от конфликта UIP. Ближайшей к конфликту UIP, то есть FUIP, является вершина, соответствующая присвоению «$u = \lambda$». В [51]

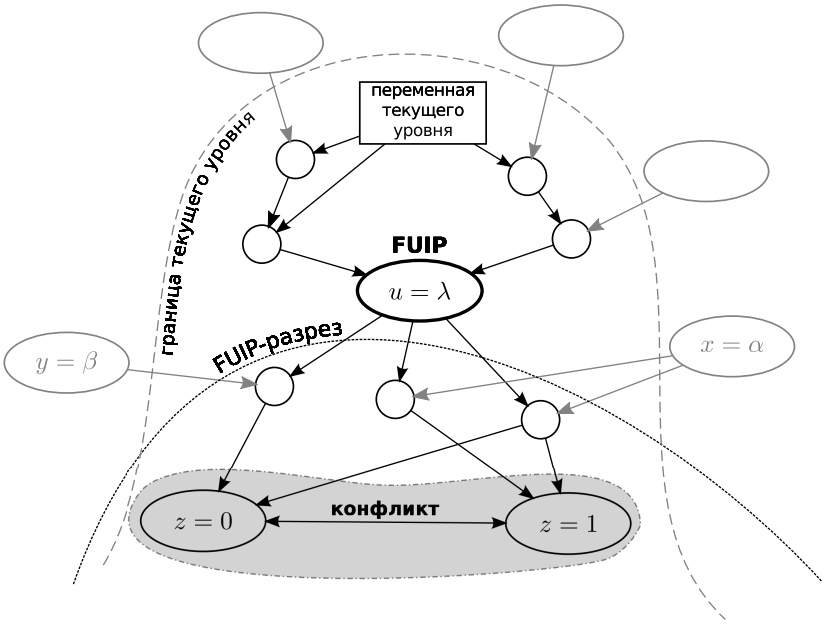


Рис. 1.4. Схема анализа графа вывода

были приведены примеры, когда анализ некоторых UIP позволял синтезировать более сильные ограничения-дизъюнкты, чем те, которые порождала схема, изначально использованная в GRASP (см. [50]). В соответствии с FUIP в ситуации, представленной на рисунке 1.4, будет порожден конфликтный дизъюнкт $u^{\overline{\lambda}} \vee x^{\overline{\alpha}} \vee y^{\overline{\beta}}$.

Следующим важным этапом DPLL-вывода является «удачный» выбор порядка угадываний переменных уровней решения. В общем случае процедуры угадывания носят характер эвристик (далее «эвристики выбора переменных»). Существуют так называемые «статические» и «динамические» эвристики выбора переменных. Статические эвристики использовались в ранних SAT-решателях (см., например, [68]). Отличительная их черта состоит в том, что приоритет угадывания переменных не зависит от получаемой в процессе поиска информации.

На практике лучшие результаты показывают динамические эвристики, которые используют для выбора угадываемых переменных информацию, получаемую в ходе поиска. Примером динамической эвристики является эвристика VSIDS (Variable State Independent Decaying Sum), впервые использованная в SAT-решателе zchaff (см. [69]). Ключевым моментом VSIDS является техника накопления статистики конфликтности перемен-

ных/литералов. В процессе анализа каждого конфликта выделяются литералы, находящиеся в графе вывода в отношении достижимости с конфликтными вершинами. Относительно таких литералов говорят, что они «принимают участие» в конфликте. В итоге каждому литералу ставится в соответствие мера конфликтности — число конфликтов, в которых данный литерал принимает участие. При выборе (угадывании) на очередном уровне решения соответствующего литерала можно обратиться к имеющейся статистике конфликтности и выбрать тот литерал, который обладает определенным приоритетом по конфликтности. Так, если требуется как можно скорее породить новый конфликт, следует выбирать в качестве угадываемого литерал с наибольшей конфликтностью. Статистика конфликтности играет весьма существенную роль, позволяя сохранять и использовать предысторию поиска.

В течение продолжительного времени работы DPLL на некоторой КНФ C база конфликтных дизъюнктов, как правило, достигает весьма больших размеров. В этих случаях принимается решение о рестарте. Процедура рестарта заключается в прекращении поиска, «чистке» базы конфликтных дизъюнктов (некоторые из них, признанные нерелевантными, могут быть удалены) и запуске поиска с новыми начальными данными. При этом накопленная статистика по конфликтности переменных сохраняется и используется в дальнейшем для формирования иерархии уровней решения. Таким образом, переменные новых уровней решения выбираются в соответствии с конфликтностью, показанной ими на предыдущих этапах (учет предыстории поиска).

Сами по себе рестарты с сохранением накопленных ограничений не влияют на полноту алгоритма (см. [70]). Однако на практике, как было сказано выше, после каждого рестарта происходит удаление ограничений, признаваемых нерелевантными. Решение о релевантности того или иного ограничения принимается на основе неких эвристических соображений и, вообще говоря, может привести к потере алгоритмом полноты: как правило, нет никаких четких гарантий того, что алгоритм не пойдет по уже пройденному пути, информация о котором была утрачена в результате

удаления соответствующего ограничения. Такие ситуации характерны для некоторых сложных задач обращения криптографических функций (см. [71]). И тем не менее, во всех быстрых SAT-решателях используются эвристические процедуры чистки баз ограничений, поскольку в случае отказа от них алгоритм довольно быстро заполняет всю доступную память ЭВМ, что ведет к резкой потере производительности.

Еще один механизм ускорения процесса решения SAT-задач заключается в использовании специальных структур данных для представления КНФ в памяти ЭВМ. В процессе работы BCP-стратегии угадывание на очередном уровне решения значения одной единственной переменной требует просмотра всей КНФ в поисках дизъюнктов, на которых срабатывает правило единичного дизъюнкта. Результаты таких срабатываний должны снова подставляться в формулу. По некоторым оценкам до 80% времени работы решателя может тратиться именно на реализацию BCP-стратегии. Поэтому алгоритм DPLL требует быстрой модификации больших массивов данных. В последних поколениях SAT-решателей это достигается в результате использования специальных приемов организации данных. Ускорение достигается за счет применения BCP не ко всем дизъюнктам, а лишь к тем из них, которые демонстрируют определенного рода «готовность» к срабатыванию на них правила единичного дизъюнкта. Структуры данных, основанные на этой идее, называются ленивыми (lazy). В качестве примеров ленивых структур данных, используемых во всех современных самых быстрых SAT-решателях, можно привести «head-tail literals» и «watched literals», предложенные в работах [72] и [73].

1.3. Двоичные решающие диаграммы

1.3.1. Базовые понятия

Двоичные решающие диаграммы (или двоичные диаграммы решений) как математический объект появляются в конце 50-х начале 60-х годов прошлого века. Впервые это понятие ввел в своей работе (см. [74]) C. Y.

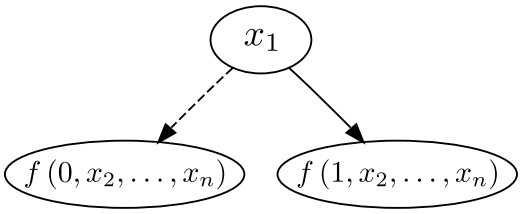
Lee в 1959 г. В основе BDD лежит разложение Шеннона булевых функций (Shannon's expansion).

Пусть дана произвольная булева функция f от n булевых переменных. Тогда для любой булевой переменной x_i,$i \in \{1, \ldots, n\}$, справедливо равенство

$$f(x_1, \ldots, x_i, \ldots, x_n) = \overline{x}_i \cdot f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \vee \\ \vee x_i \cdot f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n), \tag{1.8}$$

называемое разложением Шеннона булевой функции f по переменной x_i. При этом $f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$ и $f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$ называются коэффициентами разложения Шеннона функции $f(x_1, \ldots, x_n)$ по переменной x_i при соответствующих литералах. Разложение Шеннона имеет множество применений в теории булевых функций, теории проектирования цифровых логических схем и во многих других разделах дискретной математики.

Формулу (1.8) можно естественным образом проинтерпретировать, используя язык ориентированных помеченных графов (здесь $i = 1$):



Именно такого рода конструкции приводят к понятию двоичных диаграмм решений.

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

Стандартно BDD определяется как направленный ациклический помеченный граф, в котором выделена одна вершина с входной степенью 0, называемая корнем, и две вершины с выходной степенью 0, называемые терминальными. Терминальные вершины помечаются константами 0 и 1. Все остальные вершины помечаются переменными из множества

$X = \{x_1, \ldots, x_n\}$. Из любой вершины, за исключением терминальных, выходят в точности 2 ребра. Одно ребро, как правило, рисуют пунктирной, а другое — сплошной линией. Ребро, обозначенное пунктиром, называется low-ребром, а ребро, обозначенное сплошной линией, называется high-ребром.

Простейшие примеры BDD можно строить на основе двоичных деревьев решений. Двоичные деревья решений широко используется в различных разделах дискретной математики. В частности, с их помощью очень удобно представлять процесс означивания переменных при вычислении значений булевой функции. Остановимся на этом подробнее.

Произвольной всюду определенной булевой функции $f : \{0,1\}^n \to \to \{0,1\}$ можно поставить в соответствие ее дерево решений. Вершины (узлы) дерева соответствуют булевым переменным. Пунктирное ребро, исходящее из узла, помеченного переменной x_i, $x_i \in \{1, \ldots, n\}$, означает, что данная переменная принимает значение 0, сплошное 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.

Пример 1.2. Дана булева функция $f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$. Один из возможных вариантов дерева решений для данной функции представлен на рисунке 1.5.

Если все листья дерева решений произвольной булевой функции, помеченные 0, склеить в одну вершину и то же самое проделать с листьями, помеченными 1, получится BDD.

Прохождение произвольного пути в BDD из корня в лист индуцирует перечисление переменных из $X = \{x_1, \ldots, x_n\}$ (возможно не всех) в некотором порядке. Будем называть данный порядок порядком означивания переменных из X.

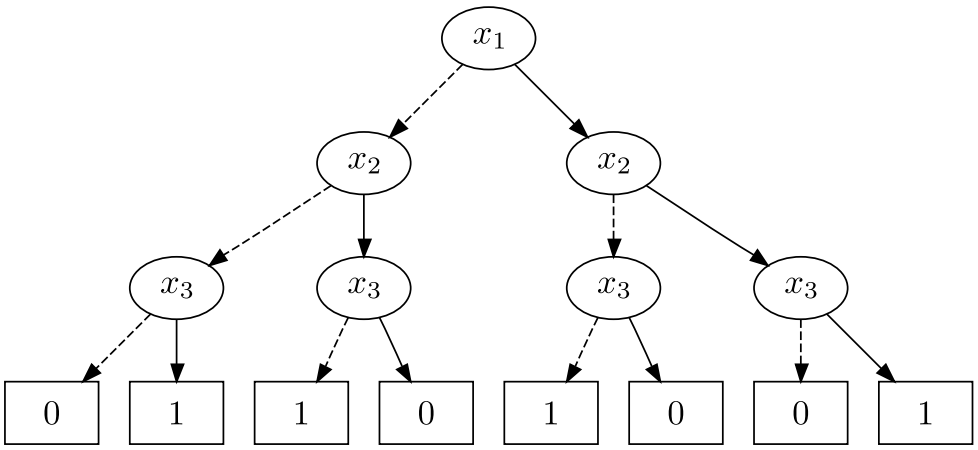


Рис. 1.5. Дерево решений функции $f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$

Если в BDD каждый путь из корня в терминальную вершину не содержат вершин, помеченных одинаковыми переменными, и его прохождение подчинено некоторому общему для всех путей порядку (например, $x_1 \prec x_2 \prec \ldots \prec x_{n-1} \prec x_n$), то такая BDD называется упорядоченной (ordered binary decision diagram, OBDD). В записи «$x_1 \prec \ldots \prec x_n$» здесь и далее подразумевается, что корень рассматриваемой OBDD помечен переменной x_1.

При использовании BDD в роли структур данных, представляющих булевы функции, возможны ситуации, когда разные вершины BDD помечены одной и той же переменной. Для того чтобы отличать такие вершины друг от друга, далее используем для вершин обозначения типа «$v_1(x), v_2(x), \ldots$». Дети произвольной нетерминальной вершины $v(x)$ обозначаются соответственно через $low\,(v(x))$ и $high\,(v(x))$. Также используем обозначение «$var\,(v(x)) = x$» или более краткое «$var(v) = x$».

В произвольной OBDD можно выделять фрагменты (подграфы), которые сами являются OBDD. Для этой цели достаточно объявить соответствующую нетерминальную вершину корнем OBDD. Идея сокращенной OBDD (reduced ordered binary decision diagram, ROBDD) заключается в склейке повторяющихся фрагментов: ROBDD-граф не должен содержать одинаковых OBDD-подграфов меньших размерностей. Таким образом, в некотором смысле ROBDD можно рассматривать как «наиболее сжатое» представление булевой функции в рассматриваемом классе графов.

Сказанное означает, что ROBDD — это OBDD, которая удовлетворяет двум условиям:

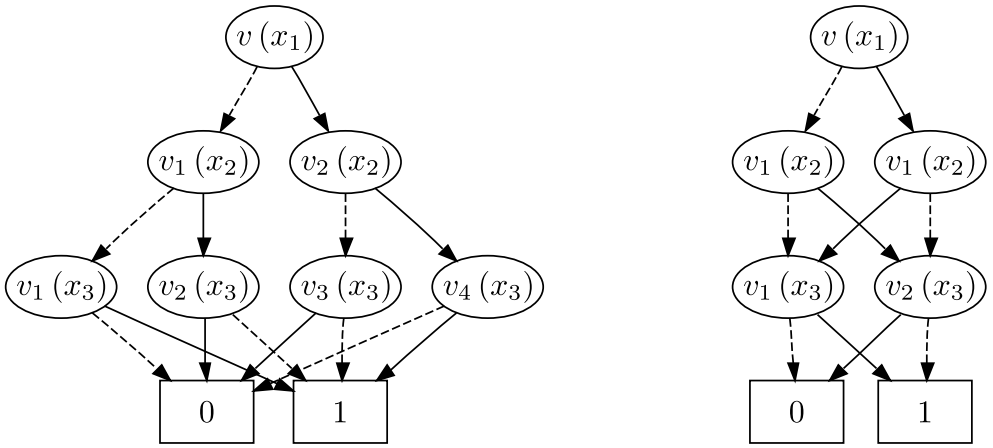
1. не существует двух различных вершин, помеченных одной и той же переменной, «одноименные» дети которых совпадают. Иными словами, равенства

$$var(v) = var(u), high(v) = high(u), low(v) = low(u)$$

означают, что $v = u$;

2. для любой нетерминальной вершины v потомок по high-ребру не совпадает с потомком по low-ребру, то есть $high(v) \neq low(v)$.

Пример 1.3. Построим ROBDD-представление рассмотренной выше булевой функции $f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$. Ее дерево решений приведено выше. Выберем порядок означивания переменных $x_1 \prec x_2 \prec x_3$. Тогда OBDD данной функции имеет следующий вид, представленный на рисунке 1.6, *а*.



(*а*). OBDD (*б*). ROBDD

Рис. 1.6. OBDD- и ROBDD-представление функции $f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$

Несложно видеть, что в рассматриваемом примере можно склеить вершины $v_2(x_3)$ и $v_3(x_3)$, а также $v_1(x_3)$ и $v_4(x_3)$. Проделав это, будем иметь граф, изображенный на рисунке 1.6, *б*. В данной OBDD все вершины уникальны (в том смысле, что нет вершин, которые можно было бы склеить).

Тем самым имеем (в соответствии с определением) ROBDD-представление рассматриваемой функции.

1.3.2. Основные алгоритмы работы с BDD

В 1986 году в работе [1] Р. Брайант показал, что любая булева функция при фиксированном порядке означивания переменных имеет единственное (с точностью до изоморфизма соответствующих графов) ROBDD-представление. Следует также отметить, что именно после этой работы пришло осознание фундаментального значения BDD для теории булевых функций. В [1] впервые были описаны алгоритмы работы с BDD, благодаря которым эта структура данных приобрела большую популярность в различных областях математической и прикладной кибернетики. Обзором по практической применимости BDD является более поздняя работа Р. Брайанта [75]. В последние 15 лет BDD стали активно использоваться в задачах верификации дискретных автоматов и программных логик (см. [76–78]).

Далее кратко остановимся на основных алгоритмах «манипулирования» булевыми функциями при помощи BDD.

Одним из важнейших в этом плане является алгоритм *Apply* (см. [1]). Данный алгоритм по паре ROBDD $B(f_1)$ и $B(f_2)$, представляющих булевы функции f_1 и f_2 над множеством булевых переменных $X = \{x_1, \ldots, x_n\}$, строит ROBDD-представление булевой функции $f_3 = f_1 * f_2$, где «$*$» — произвольная бинарная логическая связка. При этом означивание переменных в $B(f_1)$ и $B(f_2)$ должно быть подчинено одному порядку. Сложность алгоритма *Apply* построения $B(f_3)$ оценивается сверху величиной $O\left(|B(f_1)| \cdot |B(f_2)|\right)$. Здесь и далее через $|B|$ обозначено число вершин в ROBDD B.

Основа алгоритма *Apply* чрезвычайно проста и заключается в одновременном прохождении обеих ROBDD в соответствии с выбранным порядком означивания переменных. Такому обходу $B(f_1)$ и $B(f_2)$ ставится в соответствие дерево $T(f_3)$, представляющее функцию f_3. Если порядок означивания переменных в $B(f_1)$ и $B(f_2)$ совпадает, то при построении

$T(f_3)$ не происходит возвратов, поэтому число вершин в нем не превосходит величины $|B(f_1)|\cdot|B(f_2)|$. После построения дерева $T(f_3)$ оно усекается до ROBDD. Процедура усечения линейна от размерности $T(f_3)$.

Пример 1.4. Рассмотрим булевы функции $f_1(x_1, x_2) = x_1 \oplus x_2$ и $f_2(x_1, x_3) = = x_1 \vee x_3$. Выберем порядок означивания переменных: $x_1 \prec x_2 \prec x_3$. Соответственно, ROBDD $B(f_1)$ и $B(f_2)$ имеют следующий вид:

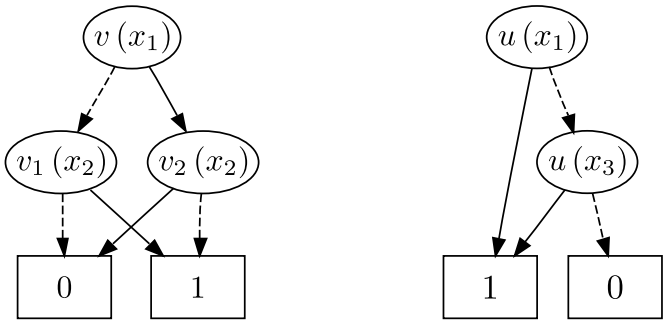


Рис. 1.7. ROBDD функций $x_1 \oplus x_2$ и $x_1 \vee x_3$

Иллюстрация работы алгоритма *Apply* на построении ROBDD функции $f_3(x_1, x_2, x_3) = f_1(x_1, x_2) \cdot f(x_1, x_3)$ приведена на рисунке 1.8. Работа *Apply* может быть представлена в виде двоичного дерева. Вершины дерева при этом помечены именами вершин в $B(f_1)$ и $B(f_2)$, в которых алгоритм *Apply* находится на данном шаге. Ребра дерева дополнительно помечены номерами переменных в заданном порядке, означивание которых происходит на текущий момент. Справа приведена ROBDD $B(f_3)$, которая является результатом работы алгоритма *Apply*.

Следует отметить, что при использовании алгоритма *Apply* в таком виде некоторые вершины дерева $T(f_3)$ могут быть пройдены более одного раза. Поэтому представляется целесообразным использование специальных механизмов «обучения» алгоритма, путем учета уже пройденных вершин. В [1] предлагается использовать имена вершин в $B(f_1)$ и $B(f_2)$ как координаты узлов в дереве $T(f_3)$. При программной реализации построенные узлы дерева заносятся в динамически заполняемую таблицу и при создании очередного узла проверяется наличие его дубликата в текущей таблице. Данный прием позволяет сразу переходить к ROBDD итоговой булевой функции, фактически минуя этап построения дерева $T(f_3)$.

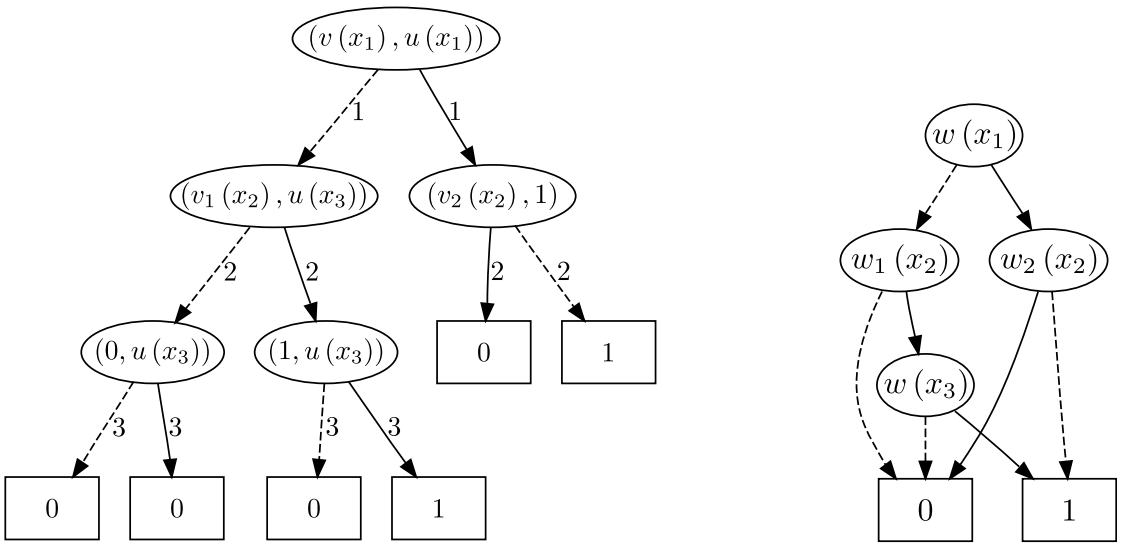


Рис. 1.8. Построение ROBDD функции $(x_1 \oplus x_2) \cdot (x_1 \vee x_3)$

В работе [1] был предложен алгоритм *SAT-count*, который, так же, как и *Apply*, широко применяется при работе с ROBDD на практике. Алгоритм *SAT-count* по данной ROBDD подсчитывает число векторов значений булевых переменных, на которых представляемая этой ROBDD булева функция принимает значение 1 (двойственно для значения 0). Сложность алгоритма *SAT-count* линейна от числа вершин в ROBDD.

Рассматриваем ROBDD $B(f)$, представляющую булеву функцию f от n переменных. Для определенности зафиксируем порядок означивания переменных $x_1 \prec x_2 \prec \ldots \prec x_{n-1} \prec x_n$. Исходную ROBDD $B(f)$ с корнем в вершине $v(x_1)$ обозначим через $B_{v(x_1)}$. И пусть также $B_{high(v(x_1))}$ — это ROBDD, которая есть подграф исходной ROBDD $B_{v(x_1)}$ с корнем в вершине, являющейся high-ребенком вершины $v(x_1)$; аналогичный смысл имеет обозначение $B_{low(v(x_1))}$. Через $ind(v(x))$ обозначим номер переменной x (помечающей вершину $v(x)$) в заданном порядке означивания переменных, а через $\#(B)$ — число наборов, на которых булева функция, представленная ROBDD B (над соответствующим множеством булевых переменных), принимает значение 1.

Любой путь из корня $B_{v(x_1)}$ в терминальную вершину «1» определяет некоторый набор или семейство наборов, доставляющих функции f значение 1. Пусть π — произвольный такой путь и вершина $v(x_i)$ входит в π,

например, вместе с вершиной $v(x_j) = high\,(v(x_i))$, причем $j > i+1$. Данный факт означает, что π определяет такое семейство наборов, доставляющих f значение 1, в котором переменные $x_{i+1}, \ldots, x_{j-1}$ принимают любые значения. Учитывая все сказанное, имеем

$$\#\left(B_{v(x_1)}\right) = 2^{ind(high(v(x_1)))-ind(v(x_1))-1} \cdot \#\left(B_{high(v(x_1))}\right) + \\ +2^{ind(low(v(x_1)))-ind(v(x_1))-1} \cdot \#\left(B_{low(v(x_1))}\right).$$

Для эффективного алгоритмического подсчета числа $\#\left(B_{v(x_1)}\right)$ следует использовать принцип «динамического программирования», подсчитывая число выполняющих наборов в направлении «от терминальной вершины «1» к корню». В этом и состоит алгоритм *SAT-count*, сложность которого ведет себя как $O\left(\left|B_{v(x_1)}\right|\right)$.

Еще одним важным алгоритмом для работы с ROBDD является алгоритм *Restrict*, также приведенный в [1].

Пусть $L(f)$ — произвольная формула, выражающая булеву функцию f от n переменных $x_1, \ldots, x_n$. Обозначим через $f|_{x_i=\alpha_i}$, $i \in \{1, \ldots, n\}$, $\alpha_i \in$ $\in \{0, 1\}$, булеву функцию от $n-1$ переменной, которая выражается формулой $L(f)|_{x_i=\alpha_i}$.

Алгоритм *Restrict* модифицирует подаваемое ему на вход ROBDD-представление $B(f)$ булевой функции f, выдавая на выходе ROBDD-представление $B\left(f|_{x_i=\alpha_i}\right)$ булевой функции $f|_{x_i=\alpha_i}$. Процедура, реализующая алгоритм *Restrict*, использует полный обход ROBDD $B(f)$. Каждая вершина, помеченная переменной x_i, удаляется, и на ее место ставится ее ребенок, соответствующий значению «$x_i = \alpha_i$». Несложно заметить, что в результате описанных преобразований структура ROBDD сохраняется (то есть после удаления некоторой вершины и передаче одного из ее детей родителям полученный граф по-прежнему является ROBDD). Все сказанное означает, что итоговая сложность алгоритма *Restrict* есть $O\left(|B(f)|\right)$.

Несложно понять, что общая задача построения ROBDD-представления произвольной булевой функций является NP-трудной. Действительно, таковой является уже задача построения ROBDD-представления константы 0 (к этой задаче сводится по Тьюрингу задача проверки невыполнимо-

сти произвольной КНФ).

В силу сказанного при решении практических задач, предполагающих построение ROBDD, оправдано применение разнообразных эвристик и «разумных соображений». Очень важным в этом контексте зачастую оказывается удачный выбор порядка означивания переменных в ROBDD. Различные приемы выбора «хорошего» порядка означивания описаны в книге [27].

На практике возможны ситуации, когда на некотором этапе построения ROBDD приходится сделать вывод о том, что выбранный порядок оказался неудачным (например, если наблюдается существенный рост количества вершин строящейся ROBDD). Для корректировки выбранного порядка можно предложить различные схемы его изменения. В [27] описана схема динамического изменения порядка (dynamic reordering) переменных в ROBDD, в основе которой лежит идеология сортировки последовательности чисел методом «пузырька». В результате получается ROBDD, представляющая ту же булеву функцию, что и исходная, но в соответствии с новым порядком означивания переменных. В [27] также показано, что данная процедура имеет очень высокую трудоемкость, накладывающую существенные ограничения на ее практическое использование.

В главе 2 настоящей работы предлагается новый алгоритм изменения некоторого фиксированного порядка означивания переменных в уже построенной ROBDD на известный новый порядок. Данный алгоритм предполагает не более чем n-кратное (n — число булевых переменных) применение процедуры, сложность которой ограничена полиномом от числа вершин в ROBDD, подаваемой ей на вход.

Глава 2

Алгоритмы решения логических уравнений и обращения дискретных функций, использующие BDD

Как уже отмечалось выше, непосредственное применение ROBDD-подхода к задачам обращения криптографических функций оправдано на ряде генераторов ключевого потока (генераторы Геффе и Вольфрама) и проигрывает SAT-подходу на более сложных криптографических тестах.

С другой стороны, негативной стороной SAT-решателей является их принципиальная неполнота, причинной которой являются эвристические процедуры чистки баз накопленных ограничений.

Выход из сложившейся ситуации видится в разработке гибридных стратегий (SAT+ROBDD) обращения дискретных функций из класса $\mathfrak{F}$. Настоящая глава содержит основы данного подхода, а также описание необходимых в его реализации алгоритмических конструкций с оценками их трудоемкости. Ниже приведен краткий план главы.

В первом разделе описана алгоритмика ROBDD-подхода в применении к задачам поиска решений логических уравнений.

Во втором разделе приведено описание новой процедуры изменения порядка означивания переменных в ROBDD.

В третьем разделе описаны основы гибридного подхода (SAT+ROBDD) к поиску решений логических уравнений, кодирующих задачи обращения дискретных функций из класса $\mathfrak{F}$.

В четвертом разделе приводятся ROBDD-аналоги основных алгоритмических конструкций, используемых в современных SAT-решателях, и строятся оценки трудоемкости соответствующих алгоритмов.

2.1. Алгоритмика ROBDD-подхода к решению систем логических уравнений

Пусть дана система логических уравнений S следующего вида

$$\begin{cases} L_1(x_1, \ldots, x_n) & = 1 \\ \ldots & \\ L_m(x_1, \ldots, x_n) & = 1 \end{cases} \tag{2.1}$$

Очевидно, что система (2.1) эквивалентна одному логическому уравнению вида

$$L_1(x_1, \ldots, x_n) \cdot \ldots \cdot L_m(x_1, \ldots, x_n).$$

Введем в рассмотрение булеву функцию $\delta_S : \{0,1\}^n \to \{0,1\}$, заданную формулой $L_1(x_1, \ldots, x_n) \cdot \ldots \cdot L_m(x_1, \ldots, x_n)$. Будем называть δ_S характеристической функцией системы (2.1). Пусть $B(\delta_S)$ — ROBDD-представление функции δ_S. Очевидно, что по известной $B(\delta_S)$ за линейное от $|B(\delta_S)|$ время можно найти некоторое решение системы (2.1) либо убедиться в ее несовместности. Тем самым задача поиска решения системы (2.1) сводится к задаче построения $B(\delta_S)$. Известны подходы к решению данной задачи, различающиеся в деталях (эвристиках). Но все они используют базовые алгоритмы работы с BDD: *Build*, *Apply*, *Restrict* и т. д. (см. [1]).

Достаточно естественным представляется подход, который заключается в построении ROBDD характеристических функций отдельных подсистем (или уравнений) системы (2.1) и последующим «объединением» их в итоговую ROBDD при помощи алгоритма *Apply*. Данная схема может быть представлена, например, следующим образом:

$$\begin{aligned} B_1 &= Apply(B(\delta_1) \cdot B(\delta_2)), \\ B_2 &= Apply(B_1 \cdot B(\delta_3)), \\ &\ldots \\ B_{m-1} &= Apply(B_{m-2} \cdot B(\delta_m)), \end{aligned}$$

где $\delta_1, \delta_2, \ldots, \delta_m$ — характеристические функции отдельных подсистем системы (2.1). Отметим, что для корректной и эффективной работы алгоритма *Apply* требуется, чтобы порядок означивания переменных для всех

рассматриваемых фрагментов системы (2.1) был одинаков. В результате решение системы (2.1) может быть найдено по $B_{m-1} = B(\delta_S)$ в общем случае за $O(|B_{m-1}|)$ шагов.

Несложно понять, что в общем случае описанный подход неэффективен. Действительно, даже если построены относительно компактные ROBDD-представления характеристических функций всех уравнений исходной системы логических уравнений, размер итоговой ROBDD может быть экспоненциальным от объема двоичной кодировки системы, например, если каждое очередное применение *Apply* дает ROBDD, число вершин в которой примерно в 2 раза больше, чем в предыдущей.

Однако на практике во многих важных задачах удается добиться хороших результатов, благодаря использованию в архитектуре ROBDD-решателей возможностей подключения разного рода эвристик, ориентированных на особенности решаемых задач.

Выбор порядка означивания переменных зачастую оказывает весьма существенное (а иногда решающее) влияние на размер итоговой ROBDD. В качестве примера приведем ROBDD для функции $(x_1 \oplus x_2) \cdot (x_3 \vee x_4)$ с порядком переменных $x_1 \prec x_3 \prec x_2 \prec x_4$ (слева) и с порядком переменных $x_1 \prec x_2 \prec x_3 \prec x_4$ (справа):

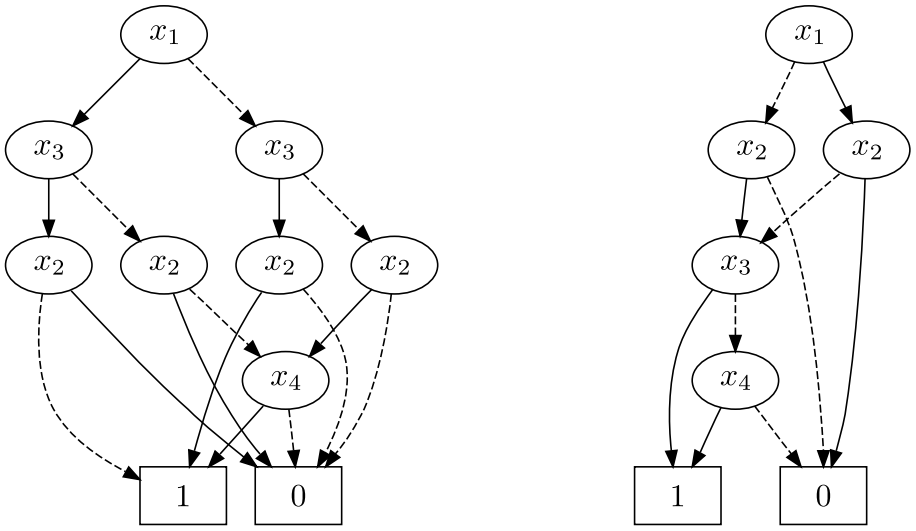


Рис. 2.1. ROBDD-представление функции $(x_1 \oplus x_2) \cdot (x_3 \vee x_4)$ с порядком $x_1 \prec x_3 \prec x_2 \prec \prec x_4$ (слева) и с порядком $x_1 \prec x_2 \prec x_3 \prec x_4$ (справа)

Рассматриваем задачу построения ROBDD-представлений всюду опре-

деленных на $\{0,1\}^n$ булевых функций. Пусть $X = \{x_1, \ldots, x_n\}$ соответствующее множество булевых переменных. Далее везде будем полагать, что из каких-либо соображений на множестве X зафиксирован некоторый порядок, обозначаемый обычно через

$$\tau : x_1 \prec x_2 \prec \ldots \prec x_n$$

и называемый начальным порядком (это может быть и порядок означивания переменных в некоторой ROBDD, с которой далее производятся те или иные действия). Если в результате некоторых действий переменные оказались переставлены относительно исходного порядка, то используются обозначения типа

$$\tau' : x_{i_1} \prec x_{i_2} \prec \ldots \prec x_{i_n}.$$

Данная запись означает, что первой рассматривается (происходит ее означивание) переменная, имеющая в исходном порядке номер $i_1 \in \{1, \ldots, n\}$, и т. д.

2.1.1. Специальные представления формул ИВ

Используемые в современных ROBDD-решателях эвристики выбора «удачного» порядка учитывают преимущества различных структур данных, представляющих уравнения системы. Описываемая далее эвристика использует в своей основе анализ «степени воздействия» переменных исходной системы друг на друга, основываясь на структуре специального формата представления формул в левых частях логических уравнений системы. А именно, в качестве представления уравнений системы предлагается использовать бинарные деревья (похожие подходы к представлению формул ИВ были использованы, например, в [44, 79, 80]). На рисунке 2.2 приведено древовидное представление формулы, записанной в левой части уравнения $(x_1 \oplus (x_2 \vee x_3)) \cdot x_4 = 1$.

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*).

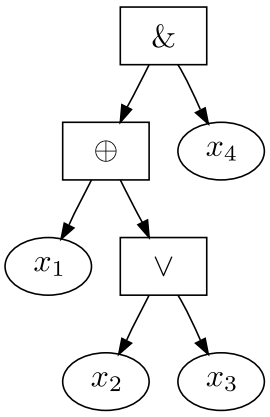


Рис. 2.2. Древовидное представление формулы $(x_1 \oplus (x_2 \vee x_3)) \cdot x_4$

Общая схема эвристики формирования порядка означивания переменных, выстраивающей «иерархию влияния» одних переменных на другие и использующей для этого древовидные представления формул, может быть представлена следующим образом:

1. производится обход всех деревьев, представляющих формулы в левых частях уравнений системы, с присвоением переменным весов (натуральных чисел), учитывающих степень взаимного влияния переменных (см. пример 2.1);

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

Пример 2.1. Приведем пример реализации описанной эвристики формирования порядка. 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:

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

Дерево T_1, «представляющее» первое уравнение, приведено на рисунке 2.2. 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, приведено на следующем рисунке.

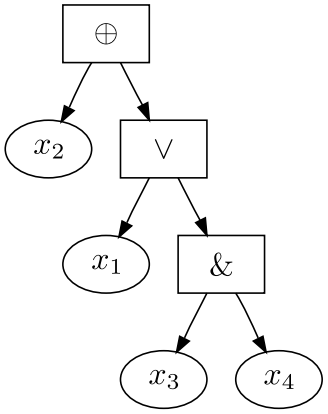


Рис. 2.3. Древовидное представление формулы $x_2 \oplus (x_1 \vee (x_3 \cdot x_4))$

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 рассматривае-

мой системы в соответствии с описанным подходом должен быть следующим: $x_2 \prec x_4 \prec x_1 \prec x_3$.

2.1.2. Разбиение системы на слои и использование шаблонов

Описанная выше техника формирования порядка переменных в ROBDD может быть усилена следующим образом (см. [81–83]). Предположим, что выбран некоторый порядок означивания переменных (например, в соответствии с представленной выше эвристикой) — $x_{i_1} \prec x_{i_2} \prec \ldots \prec x_{i_{n-1}} \prec \prec x_{i_n}$. Число $r \in \{1, \ldots, n\}$ называем индексом переменной x_{i_r} относительно выбранного порядка. Используя полученный порядок, можно разбить рассматриваемую систему логических уравнений 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 уравнениям, не входящим в первый слой. И так далее. Пусть R — число определяемых описанным образом слоев системы. Очевидно, что $1 \leq R \leq n$. ROBDD каждого слоя $B_j, j \in \{1, \ldots, R\}$, строится по следующей рекурсивной схеме:

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

где $B_{j_1}, \ldots, B_{j_k}$ — ROBDD булевых функций, выраженных формулами в левых частях уравнений системы, образующих j-тый слой. Итоговая ROBDD характеристической функции системы строится по аналогичной рекурсивной схеме:

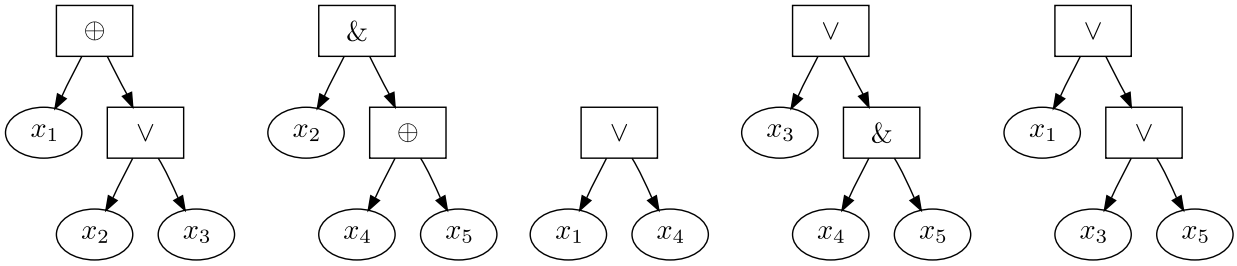
$$B = Apply\left(B_1 \cdot Apply\left(B_2 \cdot Apply\left(B_3 \cdot \ldots \cdot Apply\left(B_{R-1} \cdot B_R\right)\right)\right)\right).$$

Проиллюстрируем работу эвристики слоев на следующем примере.

Пример 2.2. Рассмотрим систему логических уравнений

$$\begin{cases} x_1 \oplus (x_2 \vee x_3) & = 1 \\ x_2 \wedge (x_4 \oplus x_5) & = 1 \\ x_1 \vee x_4 & = 1 \\ x_3 \vee x_4 \wedge x_5 & = 1 \\ x_1 \vee x_3 \vee x_5 & = 1 \end{cases} .$$

Уравнения представленной системы имеют следующие древовидные представления соответственно:



Предположим, что в контексте описанной выше эвристики выбора порядка означивания переменных имеем следующие веса переменных по уравнениям:

1. $wt_{T_1}(x_1) = 600, wt_{T_1}(x_2) = 597, wt_{T_1}(x_3) = 597, wt_{T_1}(x_4) = 0, wt_{T_1}(x_5) = 0$
2. $wt_{T_2}(x_1) = 0, wt_{T_2}(x_2) = 600, wt_{T_2}(x_3) = 0, wt_{T_2}(x_4) = 597, wt_{T_2}(x_5) = 597$
3. $wt_{T_3}(x_1) = 600, wt_{T_3}(x_2) = 0, wt_{T_3}(x_3) = 0, wt_{T_3}(x_4) = 600, wt_{T_3}(x_5) = 0$
4. $wt_{T_4}(x_1) = 0, wt_{T_4}(x_2) = 0, wt_{T_4}(x_3) = 600, wt_{T_4}(x_4) = 597, wt_{T_4}(x_5) = 597$
5. $wt_{T_5}(x_1) = 600, wt_{T_5}(x_2) = 0, wt_{T_5}(x_3) = 597, wt_{T_5}(x_4) = 0, wt_{T_5}(x_5) = 597$

Тогда суммарные веса переменных по всем уравнениям будут иметь значения: $wt(x_1) = 1800, wt(x_2) = 1197, wt(x_3) = 1794, wt(x_4) = 1794, wt(x_5) = = 1791$. Таким образом, используя описанную выше схему формирования порядка, выбираем порядок означивания переменных $x_1 \prec x_3 \prec x_4 \prec x_5 \prec \prec x_2$. То есть слой B_1 будет образован всеми уравнениями, которые содержат переменную x_1: первое, третье и пятое уравнения системы. Далее

рассматриваем оставшиеся уравнения системы (второе и четвертое) и аналогичным образом находим значения весов переменных в них (см. выше). Пусть суммарные значения весов по переменным будут следующими:

$$wt(x_2) = 600, wt(x_3) = 600, wt(x_4) = 1194, wt(x_5) = 1194.$$

Формируя на основании этих данных порядок переменных (в нашем случае это $x_4 \prec x_5 \prec x_2 \prec x_3$), заключаем, что наименьшим индексом в полученном порядке обладает переменная x_4. То есть слой должен быть образован оставшимися уравнениями, которые содержат переменную x_4. Поскольку данная переменная содержится во всех оставшихся уравнениях, слой B_2 должен быть образован уравнениями 2 и 4. Таким образом, схема построения итоговой ROBDD характеристической функции рассматриваемой системы с заданным на ней порядком означивания переменных $x_1 \prec x_3 \prec x_4 \prec x_5 \prec x_2$ может быть представлена последовательностью действий

$$B_1 = Apply\left(b_1, Apply\left(b_3, b_5\right)\right), B_2 = Apply\left(b_2, b_4\right), B = Apply\left(B_1, B_2\right),$$

где $b_i, i \in \{1, \ldots, 5\}$ — ROBDD-представления функций, выражаемых формулами в левых частях соответствующих уравнений системы.

Одним из серьезных препятствий к применению ROBDD в решении систем логических уравнений является проблема чрезмерного роста используемой ROBDD-решателем памяти. Вообще говоря, объем используемой ROBDD-решателем памяти растет как некоторая экспонента от числа применения алгоритма *Apply*. Использование разного рода эвристик (в частности описанной выше эвристики слоев) позволяет лишь уменьшить основание экспоненты, но не решить проблему в принципе. Задачи обращения криптографических функций из класса $\Im$ интересны тем, что кодирующие их системы логических уравнений часто выполнимы на единственном наборе. На практике это означает, что при построении ROBDD характеристической функции такой системы до некоторого момента будет наблюдаться экспоненциальный рост числа вершин в текущей ROBDD, а затем

начнется «экспоненциальное сокращение» ее объема (в результате должна получиться ROBDD, представляющая элементарную конъюнкцию n литералов). Качество эвристик формирования порядка и разбиения на слои можно оценивать на основе наблюдаемых коэффициентов роста (предполагая примерно те же коэффициенты убывания).

Еще одна возможность повышения эффективности ROBDD-подхода в решении систем логических уравнений состоит в использовании некоторых фрагментов рассматриваемой системы в качестве «шаблонов» для построении на их основе более сложных конструкций. Проиллюстрируем сказанное следующим примером.

Пример 2.3. Пусть дана функция $f(x_1, x_2, x_3, x_4, x_5) = x_1 \cdot (x_2 \oplus x_3 \oplus x_4) \cdot x_5$. И пусть необходимо построить ROBDD данной функции, используя порядок означивания переменных $x_1 \prec x_2 \prec x_3 \prec x_4 \prec x_5$. Применяя для построения ROBDD алгоритм *Build* (см. [84]) или *Apply*, получим следующую ROBDD.

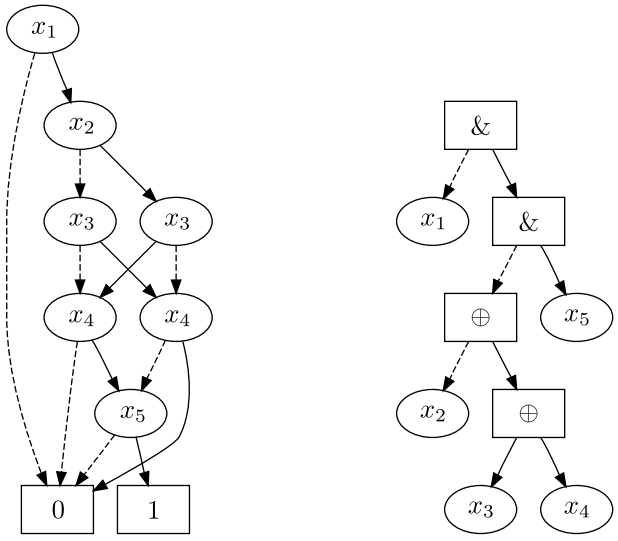


Рис. 2.4. ROBDD функции, выраженной формулой $x_1 \cdot (x_2 \oplus x_3 \oplus x_4) \cdot x_5$, и древовидное представление этой формулы

Алгоритм *Build* при этом производит построение и последующее усечение полного дерева решений рассматриваемой функции, состоящего из 63 вершин. Алгоритм *Apply*, конечно же, более эффективен. Однако можно

построить данную ROBDD еще эффективнее. Бинарное дерево, построенное по исходной формуле, изображено на рисунке 2.4 справа. Анализируя это дерево, приходим к выводу, что, зная «типовую» структуру ROBDD для конъюнкции и сложения по модулю 2, можно построить ROBDD рассматриваемой функции, не прибегая к помощи алгоритмов *Build* или *Apply*. Действительно, ROBDD функции $y(x_2, x_3, x_4) = x_2 \oplus x_3 \oplus x_4$ с порядком означивания переменных $x_2 \prec x_3 \prec x_4$ и функции $z(x_1, y, x_5) = x_1 \cdot y \cdot x_5$ с порядком означивания переменных $x_1 \prec y \prec x_5$ имеют вид (соответственно), представленный на рисунке 2.5.

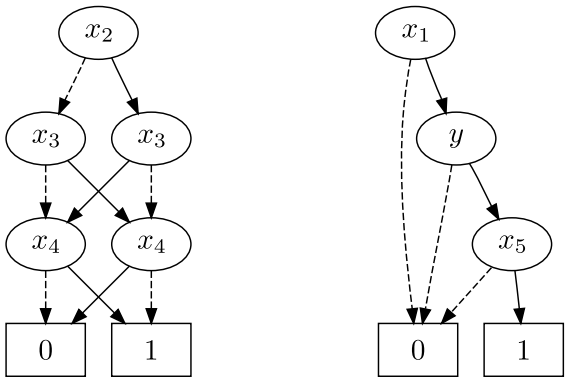


Рис. 2.5. ROBDD функций $x_2 \oplus x_3 \oplus x_4$ и $x_1 \cdot y \cdot x_5$

Подставляя в последней ROBDD вместо переменной y ROBDD для функции $y(x_2, x_3, x_4) = x_2 \oplus x_3 \oplus x_4$, получим ROBDD рассматриваемой функции.

Описанные в данном пункте техники легли в основу программно реализованного ROBDD-решателя систем логических уравнений, который оказался эффективным на задачах исследования некоторых классов дискретных автоматов ([16]). Результаты численных экспериментов, в которых использовался построенный ROBDD-решатель, приведены в главе 3.

Следует особо отметить, что использование ROBDD «в чистом виде» в задачах обращения аргументированно трудных дискретных функций, по-видимому, малоперспективно. Более того, используя алгоритм *SAT-count* и результаты работ [30, 85, 86] можно строго показать, что для некоторых классов булевых функций, заданных в КНФ, SAT-подход в отношении ко-

торых гарантированно эффективен, невозможно эффективное (полиномиальное по сложности) построение ROBDD-представлений этих функций в предположении, что $P \neq NP$ (см. [87]).

Все сказанное означает необходимость разработки в применении к проблеме обращения дискретных функций «гибридного подхода», в котором преимущества скорости обработки данных, присущей SAT-подходу, сочетаются с преимуществами ROBDD, рассматриваемыми в роли компактных форм представления булевых ограничений. Этих вопросам посвящены пункты 2.3 – 2.4 настоящей главы.

2.2. Процедуры изменения порядка означивания переменных в ROBDD

Как уже отмечалось выше, во многих задачах, возникающих на практике, выбор порядка означивания переменных булевой функции существенным образом влияет на число вершин в ROBDD-представлении данной функции.

В [27] описан подход к изменению порядка означивания переменных в ROBDD, использующий свойства разложения Шеннона. Эта техника позволяет поменять в произвольной ROBDD B две соседние в смысле заданного («текущего») на B порядка переменные.

Пусть дана ROBDD $B(f)$, представляющая булеву функцию f над множеством переменных $X = \{x_1, \ldots, x_n\}$ в соответствии с некоторым порядком τ на X. Без ограничения общности полагаем, что

$$\tau : x_1 \prec x_2 \prec \ldots \prec x_i \prec x_j \prec \ldots \prec x_n.$$

Ставится задача построения ROBDD $B'(f)$, представляющей ту же самую булеву функцию, но в соответствии с порядком

$$\tau : x_1 \prec x_2 \prec \ldots \prec x_j \prec x_i \prec \ldots \prec x_n.$$

Для решения данной задачи в [27] предлагается использовать некоторые свойства разложения Шеннона. В частности, можно вместо разложе-

ния по одной переменной использовать разложение по двум переменным: x_i и x_j, которое имеет следующий вид

$$f = x_i \cdot x_j \cdot f|_{x_i=x_j=1} \vee x_i \cdot \overline{x}_j \cdot f|_{x_i=1,x_j=0} \vee \overline{x}_i \cdot x_j \cdot f|_{x_i=0,x_j=1} \vee \overline{x}_i \cdot \overline{x}_j \cdot f|_{x_i=x_j=0}.$$

Очевидно, что это соотношение можно переписать следующим образом:

$$f = x_j \cdot x_i \cdot f|_{x_j=x_i=1} \vee x_j \cdot \overline{x}_i \cdot f|_{x_j=1,x_i=0} \vee \overline{x}_j \cdot x_i \cdot f|_{x_j=0,x_i=1} \vee \overline{x}_j \cdot \overline{x}_i \cdot f|_{x_j=x_i=0}.$$

Применительно к ROBDD-представлению f имеем возможность эффективной перестановки местами переменных x_i, x_j в порядке означивания (но только тогда, когда x_i и x_j являются в этом порядке соседними).

Проиллюстрируем возможность перестановки местами (в англ. варианте variable swap) переменных x_i и x_j, соседних в порядке означивания.

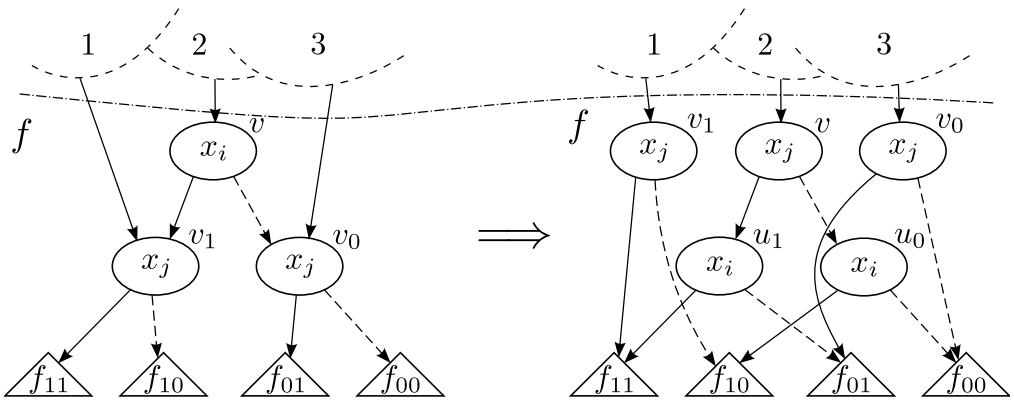


Рис. 2.6. Перестановка местами переменных x_i и x_j (соседних в исходном порядке)

Левая часть рисунка отображает часть ROBDD до изменений, правая — результат перестановки местами переменных x_i и x_j. Как видно из рисунка, после модификации ROBDD вершина v помечена переменной x_j и имеет в качестве детей новые вершины u_0 и u_1, помеченные переменной x_i. При этом пути из областей ROBDD, отмеченных цифрами 1, 2 и 3, в вершины v, v_0 и v_1 сохранены.

Можно заметить, что описанная процедура последовательных перестановок местами соседних (в порядке на ROBDD) переменных имеет очевидное сходство с известной процедурой сортировки упорядоченных массивов, за которой закрепился термин «сортировка пузырьком» (bubble sort, [88]).

В ряде ситуаций данная процедура действительно позволяет улучшать имеющийся порядок (в смысле сокращения числа вершин в итоговой ROBDD), однако в общем случае нельзя гарантировать даже незначительного улучшения. С другой стороны, возможны ситуации, когда использование этой техники приводит к весьма значительному росту числа вершин ROBDD в процессе ее промежуточной модификации, что является существенным недостатком данного подхода (этот момент подробно освещается в [27]). В [27] также рассматриваются различные способы повышения эффективности описанной процедуры.

Развитие идеи «пузырьковой сортировки» применительно к модификации порядка в ROBDD предложено в статье [89]. В этой работе также используется разложение Шеннона булевой функции по двум переменным, однако перестановка соседних переменных осуществляется не непосредственно (как в [27]), а при помощи линейных преобразований над переменными. Основным результатом [89] является алгоритм, которые выдает в общем случае приближенное решение задачи минимизации булевой функции в классе ROBDD. Сложность такого алгоритма ограничена полиномом от размера таблицы истинности, задающей булеву функцию.

Отметим, что в перечисленных выше ситуациях изменение порядка означивания переменных в ROBDD использовалось либо при *динамическом переупорядочении* (dynamic reordering) ROBDD, то есть непосредственно в процессе ее построения, либо при минимизации в классе ROBDD некоторой булевой функции, заданной таблицей истинности.

В некоторых случаях (в частности, в рассматриваемом далее гибридном (SAT+ROBDD)-выводе) есть веские основания считать некоторый вполне конкретный порядок означивания переменных лучше текущего порядка в построенной ROBDD. Тем самым возникает следующая задача.

Определение 2.1. *Дана ROBDD $B(f)$, представляющая булеву функцию $f : \{0,1\}^n \to \{0,1\}$, построенная в соответствии с заданным порядком τ означивания переменных из множества $X = \{x_1, \ldots, x_n\}$. Требуется построить ROBDD $B'(f)$, представляющую ту же самую функцию,*

в которой порядок означивания переменных из X есть τ', отличный от τ. Назовем данную проблему проблемой модификации ROBDD в соответствии с новым порядком.

Для решения данной задачи далее предлагается подход, в корне отличающийся от методов, использующих «пузырьковую сортировку». В его основе лежит возможность гарантированного решения за полиномиальное от числа вершин в ROBDD время задачи установки произвольной переменной из X на заданную позицию в новом порядке означивания переменных.

Пусть дана произвольная ROBDD $B(f)$, представляющая булеву функцию f от n переменных, и построенная в соответствии с порядком их означивания

$$\tau : x_1 \prec x_2 \prec \ldots \prec x_n. \tag{2.2}$$

Рассмотрим произвольную подстановку (см. [90]) на множестве $\{1, \ldots, n\}$

$$\sigma\left(\tau \to \tau'\right) = \begin{pmatrix} 1 & 2 & \ldots & n \\ \alpha_1 & \alpha_2 & \ldots & \alpha_n \end{pmatrix},$$

$\{\alpha_1, \ldots, \alpha_n\} = \{1, \ldots, n\}$. Будем говорить, что данная подстановка задает изменение исходного порядка τ (2.2) на порядок τ', подразумевая под этим следующее. Столбец подстановки с номером $i, i \in \{1, \ldots, n\}$, имеющий вид $\begin{pmatrix} i \\ j \end{pmatrix}, j \in \{1, \ldots, n\}$, интерпретирует тот факт, что в новом порядке τ' переменная x_j будет находиться на позиции с номером i. Например, исходный порядок $\tau : x_1 \prec x_2 \prec x_3$ подстановкой $\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}$ изменяется на порядок $\tau' : x_2 \prec x_3 \prec x_1$. Исходному порядку в этом смысле соответствует тождественная подстановка $E_\tau = \begin{pmatrix} 1 & \ldots & n \\ 1 & \ldots & n \end{pmatrix}$. Также будем говорить, что порядок τ' определяется подстановкой $\sigma\left(\tau \to \tau'\right)$ относительно порядка τ.

Определение 2.2. *Дана ROBDD $B(f)$, построенная в соответствии с заданным порядком τ означивания переменных множества X. Требуется построить ROBDD $B'(f)$, в которой порядок означивания переменных*

определяется относительно исходного порядка τ произвольной подстановкой следующего вида:

$$\begin{pmatrix} 1 & \dots & i & \dots & n \\ \alpha_1 & \dots & j & \dots & \alpha_n \end{pmatrix}.$$

Данную проблему называем проблемой установки переменной x_j на позицию с номером i.

Теорема 2.1. Пусть $B(f)$ — произвольная ROBDD с порядком τ (2.2). Для произвольных $i, j \in \{1, \dots, n\}$ проблема установки переменной x_j на позицию с номером i решается детерминированным образом за время, ограниченное сверху величиной $O\left(|B(f)|^2\right)$.

Доказательство. Пусть дана ROBDD $B(f)$, представляющая булеву функцию от n переменных, образующих множество X. Считаем, что $B(f)$ построена в соответствии с порядком τ (2.2). Рассмотрим ROBDD B_j^0 и B_j^1, представляющие булевы функции $f^0 = f|_{x_j=0}, f^1 = f|_{x_j=1}$. Данные ROBDD являются результатом применения процедуры *Restrict* к $B(f)$, что требует времени, ограниченного сверху величиной $O\left(|B(f)|\right)$. Заметим, что B_j^0 и B_j^1 — ROBDD над множеством булевых переменных $X \setminus \{x_j\}$ с заданным на нем порядком

$$\tau^* : x_1 \prec \dots \prec x_{j-1} \prec x_{j+1} \prec \dots \prec x_n.$$

Построим две ROBDD $B(x_j), B(\overline{x}_j)$, изображенные на рисунке 2.7.

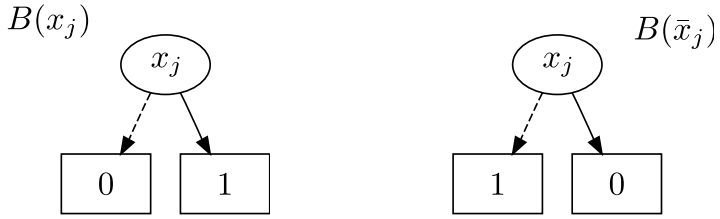


Рис. 2.7. ROBDD $B(x_j)$ (слева) и $B(\overline{x}_j)$ (справа)

При помощи алгоритма *Apply* построим в соответствии с порядком означивания переменных

$$\tau' : x_1 \prec \dots \prec x_{i-1} \prec x_j \prec x_i \prec \dots \prec x_{j-1} \prec x_{j+1} \prec \dots \prec x_n \qquad (2.3)$$

следующие ROBDD:

$$B^0 = Apply\left(B(\overline{x}_j)\cdot B_j^0\right), B^1 = Apply\left(B(x_j)\cdot B_j^1\right).$$

Рассмотрим ROBDD

$$B'(f) = Apply\left(B^0 \vee B^1\right), \tag{2.4}$$

построенную в соответствии с порядком τ' (2.3).

Заметим, что B^0 — ROBDD-представление функции $\overline{x}_j \cdot f|_{x_j=0}$, а B^1 — ROBDD-представление функции $x_j \cdot f|_{x_j=1}$. Таким образом, (2.4) можно рассматривать как специальный вид разложения Шеннона булевой функции f по переменной x_j. Тем самым ROBDD $B'(f)$ представляет булеву функцию f в соответствии с порядком (2.3) означивания переменных множества X.

Поскольку $|B(\overline{x}_j)| = |B(x_j)| = 3$, то сложность построения ROBDD B^0 и B^1 заведомо ограничена сверху величиной $O(|B(f)|)$. Это есть следствие оценки трудоемкости алгоритма *Apply* и того факта, что порядок τ^* подчинен порядку τ' (то есть из $x' \prec_{\tau^*} x''$ следует, что $x' \prec_{\tau'} x''$). В силу сказанного, число вершин в ROBDD B^0 и B^1 также ограничивается сверху величиной $O(|B(f)|)$. Откуда (снова используя оценку сложности алгоритма *Apply*) имеем: сложность построения ROBDD $B'(f)$ ограничивается сверху величиной $O\left(|B(f)|^2\right)$. Теорема 2.1 доказана. ■

Данная теорема (см. [91]) позволяет предложить следующий алгоритм изменения фиксированного начального порядка τ на новый порядок τ'.

Следствие 2.1 (теоремы 2.1) Пусть $B(f)$ — ROBDD, представляющая булеву функцию f от n переменных в соответствии с порядком означивания переменных τ. Проблема модификации $B(f)$ в соответствии с произвольным порядком τ' сводится к l-кратному ($l \leq n$) решению проблемы установки переменной на позицию с заданным номером.

Доказательство. Докажем данный факт, предъявив в явном виде соответствующий алгоритм. Пусть имеется ROBDD $B(f)$, представляющая булеву функцию f от n переменных в соответствии с некоторым порядком означивания переменных τ, и рассматривается некоторый порядок τ',

отличный от τ. Тем самым, τ' определяется относительно τ некоторой подстановкой, отличной от E_τ. Обозначим вторую строку данной подстановки через $\alpha_{\tau'} = (\alpha_1, \ldots, \alpha_n)$. Описываемый ниже алгоритм получает на входе ROBDD $B(f)$, а также порядки τ и τ'.

0. Пусть $m = 1$.

1. Рассматриваем переменную, которой в векторе $\alpha_{\tau'}$ соответствует компонента α_m.

2. Если $\alpha_m = m$, то полагаем, что $m = m + 1$ и переходим к шагу 1; в противном случае переходим к шагу 3.

3. Решаем проблему установки рассматриваемой на данном шаге переменной на позицию с номером m в текущей ROBDD.

4. Если $m < n$, то полагаем $m = m + 1$ и переходим к шагу 1; в противном случае завершаем выполнение алгоритма.

Теперь покажем, что описанный алгоритм решает проблему модификации ROBDD в соответствии с новым порядком τ'. Самое первое применение алгоритма дает установку в ROBDD некоторой переменной x_{j_1} на позицию с номером m_1. При этом, как следует из доказательства теоремы 2.1, все переменные, соответствующие компонентам вектора $\alpha_{\tau'}$, номера которых меньше m_1, своего порядка в текущей ROBDD не изменяют. Аналогично, при установке x_{j_2} на позицию с номером m_2, $m_2 > m_1$, все переменные, отвечающие компонентам $\alpha_{\tau'}$ с номерами $< m_2$, не изменяют своего порядка в ROBDD, то есть переменная x_{j_1} сохраняет позицию с номером m_1. Обобщая сказанное, заключаем, что для корректного решения проблемы модификации ROBDD в соответствии с новым порядком τ' достаточно не более n раз решить проблему установки переменной на заданную позицию, что и делает описанный алгоритм. Следствие доказано. ■

Обратим внимание на то, что данный алгоритм, вообще говоря, не является полиномиальным (от размера исходной ROBDD). Однако фактически он разбит на l, $l \leq n$, процедур, каждая из которых устанавливает некоторую переменную на заданную позицию и сама по себе работает за полиномиальное от числа вершин в подаваемой ей на вход ROBDD время. На

практике алгоритм показывает хорошие результаты в задачах модификации порядка ROBDD, возникающих в гибридном (SAT+ROBDD)-выводе. Его эффективность особенно заметна при незначительных отличиях в новом и старом порядках (когда большая часть переменных не меняет своего местоположения). Следует подчеркнуть, что для процедур, использующих пузырьковую сортировку, в общем случае невозможны полиномиальные (от числа вершин в исходной ROBDD) оценки даже в отношении задачи установки переменной на заданную позицию.

2.3. Основы гибридного (SAT+ROBDD) подхода к задачам обращения дискретных функций

Как уже отмечалось выше, использование ROBDD «в чистом виде» оправдано лишь на обращении сравнительно простых функций (криптоанализ простейших генераторов типа Геффе, см. [92]). Однако ROBDD привлекательны как структуры данных, наиболее экономным образом представляющие булевы функции в специальном классе графов. Этот факт приводит к идее использования ROBDD в качестве структур, представляющих булевы ограничения, накапливаемые в процессе нехронологического DPLL-вывода. Данная идея представляется перспективной именно в отношении задач обращения полиномиально вычислимых функций. И этому есть целый ряд причин.

Одна из главных причин состоит в том, что если рассматривать задачу обращения некоторой полиномиально вычислимой дискретной функции $f : \{0,1\}^n \to \{0,1\}^*$, $f \in \Im$, как SAT-задачу, то в соответствующей КНФ $C(f)$ можно выделить подмножество булевых переменных, от которых в некотором смысле «функционально зависят» все остальные переменные, фигурирующие в данной КНФ. Это множество, обозначаемое далее через X, образовано булевыми переменными, кодирующими входное слово из $\{0,1\}^n$. Как правило, число n существенно меньше общего числа переменных в $C(f)$. Однако можно показать, что для решения соответствующей SAT-задачи достаточно оперировать (в указанном ниже смысле) только с

переменными множества X. Кратко остановимся на перечисленных моментах (результаты, представленные в данном пункте, подробно изложены в [87] и [82]).

Пусть $C = C(x_1, \ldots, x_k)$ — произвольная КНФ над множеством булевых переменных $\tilde{X} = \{x_1, \ldots, x_k\}$. Рассмотрим некоторое множество $X' = (x'_1, \ldots, x'_r)$, являющееся подмножеством $\tilde{X}$. Пусть $(\alpha_1, \ldots, \alpha_r)$ — произвольный вектор, образованный значениями истинности переменных из X'. Осуществим подстановку в КНФ C значения $x'_1 = \alpha_1$. Данная подстановка заключается в вычеркивании из C некоторых литералов и дизъюнктов. При этом отслеживаются возможности срабатывания правила единичного дизъюнкта с последующими подстановками в C соответствующих индуцированных значений. Если в результате не выведен конфликт или выполняющий C набор, то в КНФ $C|_{x'_1=\alpha_1}$ осуществляется подстановка $x'_2 = \alpha_2$. И так далее.

Описанная процедура определяет последовательную подстановку в C вектора $(\alpha_1, \ldots, \alpha_r)$ относительно порядка $x'_1 \prec \ldots \prec x'_r$. Очевидно, что последовательная подстановка в общем случае реализуется эффективно. Возможны различные исходы этой процедуры. Во-первых, может оказаться, что данная подстановка выводит конфликт. Во-вторых, что ее результатом является нахождение некоторого выполняющего C набора. Наконец, возможен переход к некоторой КНФ, относительно которой нельзя сказать ничего. Если имеет место первая или вторая ситуация, то будем говорить, что данная подстановка индуцирует детерминированный DPLL-вывод соответственно конфликта или выполняющего набора.

Предположим, что относительно некоторой КНФ C над $\tilde{X}$ и некоторого множества $X' : X' \subset \tilde{X}$, можно показать, что результатом последовательной подстановки любого вектора значений переменных из X' в C относительно некоторого порядка могут быть только первая или вторая ситуации. Несложно видеть, что в этом случае достаточно перебрать $2^{|X'|}$ всевозможных значений переменных из X', чтобы решить рассматриваемую SAT-задачу в отношении КНФ C.

Определение 2.3. *Пусть* $\tilde{X} = \{x_1, \ldots, x_k\}$ *— множество булевых переменных и* $X' \subseteq \tilde{X}$*. Проекцией произвольного вектора* $\alpha = (\alpha_1, \ldots, \alpha_k)$ *значений переменных из* $\tilde{X}$ *на множество* X' *называется вектор, образованный теми компонентами* α*, которые являются значениями переменных из* X'*. Проекцию вектора* α *на множество* X' *обозначим через* $\alpha_{X'}$*.*

Определение 2.4. *Ядром DPLL-вывода для КНФ* $C = C(x_1, \ldots, x_k)$ *над множеством булевых переменных* $\tilde{X} = \{x_1, \ldots, x_k\}$ *называется такое множество* $X^{ker}(C) \subseteq \tilde{X}$ *с введенным на нем порядком* τ*, что имеют место следующие свойства:*

1. *для любого вектора* $\alpha = (\alpha_1, \ldots, \alpha_k)$*, выполняющего* C*, последовательная подстановка в* C *вектора* $\alpha_{X^{ker}(C)}$ *относительно* τ *индуцирует детерминированный DPLL-вывод* α*;*

2. *для любого вектора* $\beta = (\beta_1, \ldots, \beta_k) : C|_\beta = 0$*, последовательная подстановка в* C *вектора* $\beta_{X^{ker}(C)}$ *относительно* τ *индуцирует детерминированный DPLL-вывод конфликта.*

Ядро $X^{ker}(C) : X^{ker}(C) = \tilde{X}$*, называется тривиальным. Ядро наименьшей мощности называется минимальным и обозначается через* $X^{ker}_*(C)$*.*

Рассмотрим произвольную функцию $f : \{0,1\}^n \to \{0,1\}^*$, $f \in \Im$. Назовем множество $X = \{x_1, \ldots, x_n\}$ множеством переменных входа функции f. Рассматриваем задачу обращения f в произвольной точке $y \in range\ f$. При помощи преобразований Цейтина (см. [28]) и техники, описанной, например, в [93], сводим за полиномиальное время данную проблему к задаче поиска выполняющего набора выполнимой КНФ $C(f)$ от булевых переменных, образующих множество $\tilde{X} = \{x_1, \ldots, x_{q(n)}\}$, $q(\cdot)$ — некоторый полином. Справедлива следующая теорема.

Теорема 2.2 (см. [87]) Рассмотрим произвольную функцию $f : \{0,1\}^n \to$ $\to \{0,1\}^*$ из класса $\Im$. Обозначим через X_τ множество переменных входа

f с зафиксированным на нем порядком τ (вообще говоря, произвольным). Пусть $C(x_1, \ldots, x_{q(n)})$ — КНФ, кодирующая задачу обращения функции f в произвольной точке $y \in range\ f$. Тогда $X_*^{ker}(C(x_1, \ldots, x_{q(n)})) \subseteq X_\tau$.

Фактически данная теорема означает следующее. Будем рассматривать задачу обращения функции f в произвольной точке $y \in range\ f$ как SAT-задачу в отношении КНФ $C(x_1, \ldots, x_{q(n)})$, решаемую при помощи любого алгоритма, основанного на DPLL. Тогда в качестве переменных уровней решения достаточно выбирать переменные входа рассматриваемой функции. Очень важен тот факт, что порядок выбора может быть произвольным. Это означает, что стратегия выбора переменных только из X гарантирует решение задачи обращения f в произвольной точке $y \in range\ f$ алгоритмом DPLL, использующим CL-процедуру и рестарты [70].

Рассмотрим систему логических уравнений следующего вида

$$\begin{cases} W_1(y_1, \ldots, y_s) & = 1 \\ \ldots & \\ W_k(y_1, \ldots, y_s) & = 1 \\ C(z_1, \ldots, z_t) & = 1 \end{cases} \tag{2.5}$$

над множеством булевых переменных $U = \{y_1, \ldots, y_s\} \cup \{z_1, \ldots, z_t\}$. Предположим, что $C(z_1, \ldots, z_t)$ — КНФ, для которой известно некоторое нетривиальное ядро DPLL-вывода $Z^{ker}(C)$, причем $Z^{ker}(C) \subseteq \{y_1, \ldots, y_s\}$. Через $B(W)$ обозначим ROBDD-представление характеристической функции системы

$$\begin{cases} W_1(y_1, \ldots, y_s) & = 1 \\ \ldots & \\ W_k(y_1, \ldots, y_s) & = 1 \end{cases} \tag{2.6}$$

Определение 2.5. *Пусть $B(W)$ — ROBDD-представление булевой функции $W : \{0,1\}^s \to \{0,1\}$ от s переменных $y_1, \ldots, y_s$. Назовем путь в $B(W)$ от корня к терминальной вершине «1» полным относительно множества Y', $Y' \subseteq Y$, если прохождение этого пути задает присвоение соответствующих значений всем переменным из Y'.*

Справедлива следующая теорема.

Теорема 2.3 (см. [82]) Прохождение любого пути в ROBDD $B(W)$ из корня в терминальную вершину «1», который полон относительно множества $Z^{ker}(C)$, индуцирует подстановку в КНФ $C(z_1, \ldots, z_t)$ значений переменных из $Z^{ker}(C)$. Результатом этой подстановки является либо вывод по правилу единичного дизъюнкта конфликта, либо вывод набора, выполняющего $C(z_1, \ldots, z_t)$. В последнем случае имеем некоторое решение исходной системы.

Пусть уравнение $C(x_1, \ldots, x_{q(n)}) = 1$ кодирует задачу обращения функции f из класса $\Im$ в некоторой точке $y \in range\ f$. Через $X^{ker}(C)$ обозначено некоторое нетривиальное ядро DPLL-вывода КНФ C (для рассматриваемой задачи в качестве $X^{ker}(C)$ всегда можно взять X — множество переменных входа функции f). Организуем решение задачи поиска набора, выполняющего C, при помощи алгоритма DPLL, дополненного CL-процедурой (см. главу 1). При этом, руководствуясь теоремой 2.2, будем выбирать в качестве переменных уровней решения только те переменные, которые находятся в $X^{ker}(C)$. Допустим, что осуществлено Q итераций такого выбора с последующим распространением булевых ограничений и реализацией CL-процедуры, но выполняющий набор при этом не найден. Обозначим через

$$D_1\left(x_1^1, \ldots, x_{r_1}^1\right), \ldots, D_Q\left(x_1^Q, \ldots, x_{r_Q}^Q\right)$$

конфликтные дизъюнкты, выведенные в данных итерациях (очевидно, что $\bigcup_{i=1}^{Q}\{x_1^i, \ldots, x_{r_i}^i\} \subseteq X^{ker}(C)$). Рассмотрим следующую систему логических уравнений:

$$\begin{cases} D_1(x_1^1, \ldots, x_{r_1}^1) & = 1 \\ \ldots & \\ D_Q(x_1^Q, \ldots, x_{r_Q}^Q) & = 1 \end{cases} \tag{2.7}$$

Определение 2.6. *Пусть S — произвольная совместная система логических уравнений, $B(S)$ — ROBDD-представление ее характеристической функции, и π — произвольный путь из корня $B(S)$ в терминальную вершину «1». Данный путь определяет некоторое множество решений S,*

обозначаемое через $A(\pi)$, $|A(\pi)| \geq 1$. *Про любое* $\alpha \in A(\pi)$ *говорим также, что путь* π *содержит* α.

Теорема 2.4 (см. [87]) Обозначим через $\alpha = (\alpha_1, \ldots, \alpha_n)$ произвольное решение задачи обращения функции f из класса $\Im$ в некоторой точке $y \in range\ f$. Пусть $B(S)$ — ROBDD-представление характеристической функции системы (2.7) в контексте рассматриваемой задачи. Тогда существует такой путь π из корня $B(S)$ в терминальную вершину «1» , что $\alpha \in A(\pi)$.

Отметим, что данная теорема определяет конкретный вид систем (2.5)––(2.6) в контексте задачи обращения функции f в точке $y \in range\ f$. Роль $C(z_1, \ldots, z_t)$ в этом случае играет КНФ $C(f) = C(x_1, \ldots, x_{q(n)})$, а роль системы (2.6) — система (2.7), образованная уравнениями вида $D_i = 1$, $i \in \{1, \ldots, Q\}$, где D_i — конфликтные дизъюнкты, полученные в результате DPLL-вывода, примененного к КНФ $C(x_1, \ldots, x_{q(n)})$, с выбором переменных уровней решения из $X^{ker}(C)$.

Теоремы 2.2–2.4 дают теоретическую базу для нового подхода к решению задач обращения полиномиально вычислимых дискретных функций, основная идея которого состоит в совместном использовании SAT и ROBDD именно в тех «частях» задачи обращения, где они могут дать ощутимый выигрыш. Как уже говорилось, навряд ли можно за счет использования только ROBDD «обогнать» SAT-подход на задачах обращения криптографических функций. Однако ROBDD могут использоваться для решения другой важной проблемы — потери полноты SAT-решателем в результате чистки баз ограничений. Именно ROBDD представляются оптимальными структурами данных для хранения массивов конфликтных дизъюнктов, накапливаемых SAT-решателем в процессе вывода (и это целиком подтверждается численными экспериментами). Особо отметим, что никакие из синтезированных в процессе вывода ограничений при этом не удаляются (в отличие от практики, принятой в большинстве современных SAT-решателей).

Все сказанное означает необходимость описания и изучения ROBDD-аналогов механизмов логического вывода, используемых в современных SAT-решателях, базирующихся на DPLL. Соответствующие процедуры будут применяться к базам булевых ограничений, представленным не в виде конъюнкций дизъюнктов, а в виде ROBDD.

2.4. Логический вывод на ROBDD (основные алгоритмы)

Рассматривается проблема обращения функции $f : \{0,1\}^n \to \{0,1\}$ из класса $\Im$. Пусть $B(S)$ — ROBDD-представление характеристической функции системы логических уравнений (2.7). Дальнейшая цель состоит в описании процесса логического вывода на ROBDD $B(S)$. Для этого потребуется определить аналоги таких компонент DPLL-вывода, как правило единичного дизъюнкта, CL-процедура, процедуры «отсроченной» работы с данными (head-tail literals и watched literals). Все приводимые далее результаты справедливы в отношении произвольных ROBDD (см. [91]).

Определение 2.7. *Пусть $B(f)$ — ROBDD-представление произвольной булевой функции f от переменных $x_1, \ldots, x_n$. Каждой переменной x_i, $i \in$ $\in \{1, \ldots, n\}$, и терминальным вершинам «0», «1» поставим в соответствие множества значений данной переменной, задаваемых всевозможными путями в $B(f)$ из корня в соответствующую терминальную вершину. Данные множества обозначим через $\Delta^0(x_i)$, $\Delta^1(x_i)$.*

Предположим, что в некоторой ROBDD $B(f)$ выполнены перечисленные ниже условия.

1.) Для некоторой переменной $x_k \in X$, $X = \{x_1, \ldots, x_n\}$, любой путь π из корня $B(f)$ в терминальную «1» обязательно проходит через некоторую вершину, помеченную переменной x_k.

2.) Справедливо $\left|\Delta^1(x_k)\right| = 1$.

Результатом данной ситуации является заключение о том, что в любом наборе значений истинности переменных из множества X, на котором значение функции f, представленной $B(f)$, равно 1, переменная x_k может принимать только одно значение (соответствующее значение в $\Delta^1(x_k)$).

Определение 2.8. *Определяемую условиями 1.)–2.) ситуацию далее называем ROBDD-следствием соответствующего значения для переменной* x_k.

Возникновение в $B(S)$, представляющей базу булевых ограничений-дизъюнктов, ROBDD-следствия для некоторой переменной является аналогом правила единичного дизъюнкта в DPLL-выводе. Покажем, что справедлив следующий факт.

Лемма 2.1. Выполнимость 1.)–2.) относительно некоторой переменной x_k означает, что выполнено в точности одно из следующих условий:

1. для каждой вершины, помеченной x_k, ее high-ребенком является терминальная вершина «0»;

2. для каждой вершины, помеченной x_k, ее low-ребенком является терминальная вершина «0».

Доказательство. Прежде всего отметим, что из любой нетерминальной вершины ROBDD $B(f)$ достижима как «0», так и «1». Предположим теперь, что выполнены условия 1.)–2.). Это означает, что для некоторой переменной x_k любой путь в ROBDD $B(f)$ из корня в терминальную вершину «1» содержит вершину, помеченную данной переменной, и все такие пути в $B(f)$ задают x_k одно и то же значение (поскольку $\left|\Delta^1(x_k)\right| = 1$) — либо 0, либо 1. Не ограничивая общности, будем считать, что все пути из корня в «1» задают x_k значение 1 ($\Delta^1(x_k) = \{1\}$). Если предположить, что ребенком некоторой $v(x_k)$ является терминальный «0», то это может быть только low-ребенок. Действительно, в противном случае из $v(x_k)$ по low-ребенку достижима терминальная «1», и соответствующий путь задает x_k значение 0, что противоречит сделанному предположению.

Предположим, что найдется такая $v'(x_k)$, что ее low-ребенком не является терминальный «0», и обозначим через $v'(x_l)$ вершину, являющуюся low-ребенком $v'(x_k)$. Но из $v'(x_l)$ достижима терминальная «1», поэтому существует путь из корня в «1», задающий x_k значение 0, что противоречит условиям 1.)–2.). Итак, если выполнены условия 1.)–2.) и любой путь в $B(f)$ из корня в «1» задает x_k значение 1, то low-ребенок любой вершины, помеченной x_k, — это терминальный «0». Аналогично, если выполнены 1.)–2.) и любой путь из корня $B(f)$ в «1» задает x_k значение 0, то high-ребенок любой вершины, помеченной x_k, — это терминальный «0». Лемма 2.1 доказана. ■

Лемма 2.2. Процедура проверки возникновения ROBDD-следствий в ROBDD $B(f)$ требует детерминированного времени, ограниченного сверху величиной $O\left(n \cdot |B(f)|\right)$.

Доказательство. Сам по себе данный факт не является трудным. Поэтому представляется целесообразным привести в процессе доказательства полное описание алгоритма отслеживания ROBDD-следствий. Осуществляется это при помощи представленной ниже процедуры *check_ impl()*.

```
check_impl(u          - текущая вершина,
           inf_vector - вектор выведенных значений переменных)
{
    if u == 1:
        // u - терминальная вершина
        // проверить «целостность пути» (проверка условия 1)
        check_path_consistency(u, inf_vector);
        return;

    if not already_in_cache():
        // если такой вершины еще нет в кэше,
        // производим рекурсивный спуск по исходной ROBDD,
        // вызывая check_impl() для потомков данной вершины
        check_impl(low(u),  inf_vector);
        check_impl(high(u), inf_vector);

        // проверить «целостность пути» (проверка условия 1)
```

```
        check_path_consistency(u, inf_vector);

        // проверить наличие нулевого ребенка (проверка условия 2)
        check_for_zero_child(u, inf_vector, 0);

        // добавить в кэш запись о том, что данная вершина пройдена
        put_to_cache(u);

    return;
}
```

Как видно из представленного выше псевдокода, процедура *check_impl()* рекурсивна и осуществляет полный обход $B(f)$. Для каждой вершины u проверяются условия 1.)–2.) относительно переменной, соответствующей u. Это осуществляется посредством следующих двух процедур: *check_path_consistency()* и *check_for_zero_child()*.

Процедура *check_path_consistency()*, находясь в вершине u, помеченной некоторой переменной $x_k, k \in \{2, \ldots, n\}$, просматривает родителей данной вершины. Если среди них имеется вершина, помеченная переменной $x_j, j < k - 1$, то принимается решение о невозможности возникновения ROBDD-следствий в отношении переменных $x_{j+1}, \ldots, x_{k-1}$.

Процедура *check_for_zero_child()* проверяет для вершины u, помеченной переменной x_k, выполняется ли в ее отношении утверждение леммы 2.1.

В представленном псевдокоде использована специальная процедура кэширования, назначение которой в хранении информации о пройденных вершинах (ее использование исключает повторное прохождение вершин).

Результатом работы *check_impl()* является *inf_vector* — вектор длины n с компонентами из множества $\{-1, 0, 1\}$. На начальном шаге все компоненты данного вектора нулевые. Компонента с номером $k, k \in \{1, \ldots, n\}$, принимает значение 1 или -1, если для переменной x_k по ROBDD-следствию выведено значение соответственно $x_k = 1$ или $x_k = 0$.

Оценим сложность процедуры *check_impl()*. Заметим, что данная процедура один раз обходит ROBDD $B(f)$, модифицируя при необходимости

вектор *inf_ vector*, при этом работа с произвольной вершиной u может потребовать (в процедуре *check_ path_ consistency()*) просмотра всего текущего вектора *inf_ vector*. Применение процедуры *check_ for_ zero_ child()* в отношении произвольной вершины требует времени $O(1)$. Таким образом, итоговая сложность процедуры *check_ impl()* ограничена сверху величиной $O\left(n \cdot |B(f)|\right)$. Лемма 2.2 доказана. ■

Для дальнейшей работы потребуется модифицированный алгоритм *Restrict*, который позволяет осуществлять в ROBDD $B(f)$ подстановку набора значений некоторых переменных из X.

Как отмечает Р. Брайант [1, 75], данный алгоритм имеет ту же сложность, что и обычный *Restrict*, то есть $O\left(|B(f)|\right)$. Однако данный факт в упомянутых работах не доказывается. Поэтому далее приводится псевдокод модифицированного алгоритма *Restrict* и доказательство соответствующей сложностной оценки.

```
restrict_m(u             - текущая вершина,
           oldbdd        - исходная ROBDD,
           newbdd        - новая ROBDD,
           assign_vector - вектор подставляемых значений переменных)
{
    вершина res; // создаваемая на данном шаге вершина в newbdd

    if u == 1 || u == 0:
        // u - терминальная вершина
        return u;

    if not already_in_cache():
        // если такой вершины еще нет в кэше
        if not in_vector(var(u), assign_vector):
            // переменную не нужно означивать
            low  = restrict_m(low(u),  newbdd, assign_vector);
            high = restrict_m(high(u), newbdd, assign_vector);
            var = var(u);

            // создаем в newbdd вершину с координатами var, low, high
            res = make_newnode(newbdd, var, low, high);
        else:
```

```
            // данную переменную необходимо означить в соответствии
            // с ее значением в векторе assign_vector
            if value_to_assign(var(u), assign_vector) == 1:
                // присвоить переменной var(u) значение 1
                res = restrict_m(high(u), newbdd, assign_vector);
            else:
                // присвоить переменной var(u) значение 0
                res = restrict_m(low(u),  newbdd, assign_vector);

        // добавить в кэш запись о том, что данная вершина пройдена
        put_to_cache(res);
    else:
        // в кэше уже существует запись о прохождении
        // данной вершины ранее; взять значение из кэша
        res = get_from_cache(u);

    return res;
}
```

Лемма 2.3. Сложность модифицированной процедуры *Restrict* ограничена сверху величиной $O\left(|B(f)|\right)$.

Доказательство. Заметим, что модифицированная процедура *Restrict*, по аналогии с обычной, предполагает полный обход $B(f)$. При этом для каждой вершины проверяется, входят ли переменные, помечающие ее детей, в список означиваемых, и если «да», то меняются ссылки на соответствующих детей. Все эти действия имеют сложность порядка $O(1)$. Далее отмечаем, что при передаче ссылок на детей каждой удаляемой вершины ее родителям структура ROBDD сохраняется (ситуации возникновения дубликатов и вершин, у которых оба ребенка совпадают, невозможны). Таким образом, модифицированный *Restrict* обходит $B(f)$, затрачивая на обработку каждой вершины время $O(1)$. Итоговая сложность данной процедуры ограничивается сверху величиной $O\left(|B(f)|\right)$. Лемма 2.3 доказана. ■

Теорема 2.5. Пусть в ROBDD $B(f)$ подставляются значения переменных

$$x_{i_1} = \alpha_{i_1}, \ldots, x_{i_m} = \alpha_{i_m}, m \leq n, \alpha_{i_j} \in \{0, 1\}, j \in \{1, \ldots, m\}.$$

Сложность детерминированной процедуры, осуществляющей данную подстановку и проверяющей наличие всевозможных ROBDD-следствий, ограничена сверху величиной $O\left(n \cdot |B(f)|\right)$.

Доказательство. Используя результаты леммы 2.2 и модифицированный *Restrict*, можно записать процесс подстановки набора значений переменных в ROBDD с проверкой возникновения ROBDD-следствий в виде следующей процедуры.

```
assign(oldbdd        - исходная ROBDD,
       newbdd        - новая ROBDD,
       assign_vector - вектор подставляемых значений переменных,
       inf_vector    - вектор выведенных значений переменных)
{
    // подстановка в ROBDD oldbdd значений переменных
    // из вектора assign_vector
    restrict_m(oldbdd, newbdd, assign_vector);

    // проверка наличия ROBDD-следствий в ROBDD newbdd
    check_impl(newbdd, inf_vector);
}
```

Учитывая, что сложность процедуры *check_ impl()* ограничена величиной $O\left(n \cdot |B(f)|\right)$ (лемма 2.2), а сложность процедуры *restrict_ m()*, реализующей модифицированный *Restrict*, ограничена величиной $O(|B(f)|)$ (лемма 2.3), заключаем, что верхняя граница сложности для процедуры *assign()* имеет вид $O\left(n \cdot |B(f)|\right)$. Теорема 2.5 доказана. ■

Теорема 2.6. Если результатом применения процедур *check_ impl()* или *assign()* к $B(f)$ является ROBDD-следствие $x_k = \alpha_k$, $\alpha_k \in \in \{0, 1\}$, для некоторой $x_k \in X$, то подстановка в $B(f)$ «$x_k = \alpha_k$» не может привести к возникновению нового ROBDD-следствия, индуцированного данной подстановкой.

Доказательство. Пусть в результате одной из перечисленных процедур в $B(f)$ возникло ROBDD-следствие $x_k = \alpha_k$. Не ограничивая общности, полагаем, что $\alpha_k = 1$. В силу леммы 2.1, сделанное предположение означает, что low-ребенком всех вершин, помеченных x_k, является терминальный

«0». Подстановка «$x_k = 1$» в $B(f)$ для произвольной вершины $u(x_k)$ означает передачу high-ребенка вершины $u(x_k)$ вершинам, являющимся родителями $u(x_k)$. Но low-ребенок $u(x_k)$, то есть терминальный «0», при этом не передается никаким вершинам. Таким образом, подстановка «$x_k = 1$» в $B(f)$ не может привести к *возникновению* в $B(f)$ вершины, ребенком которой является терминальный «0» (что не исключает наличия таких вершин, находившихся в $B(f)$ до осуществления подстановки «$x_k = 1$»). Аналогичные рассуждения справедливы и в предположении, что $\alpha_k = 0$. Теорема доказана. ■

Данный факт демонстрирует очень привлекательное свойство ROBDD, рассматриваемой в роли базы булевых ограничений (см. рисунок 2.8). Напомним, что подстановка значения некоторой переменной в КНФ может приводить к выводу по правилу единичного дизъюнкта (unit clause) ряда индуцированных присвоений, подстановка которых также не исключает дальнейших срабатываний unit clause и т. д. В этом смысл стратегии распространения булевых ограничений (BCP). В общем случае полная реализация BCP может приводить к многократному обходу КНФ, что сопряжено с существенными вычислительными затратами. Полученное свойство ROBDD означает, что порождаемые произвольной подстановкой ROBDD-следствия сами по себе новых ROBDD-следствий породить не могут и, таким образом, вся информация, индуцируемая данной подстановкой, извлекается в результате однократного обхода ROBDD.

Природа конфликтов в гибридном выводе более разнообразна, чем в DPLL-выводе. Во-первых, это обычные «DPLL-конфликты», возникающие в результате подстановок в КНФ-часть. Во-вторых, это «гибридный конфликт»: ситуация, когда в результате последовательности подстановок в ROBDD-части возникло ROBDD-следствие «$x_k = \alpha$», а в КНФ-части возникло выведенное по правилу единичного дизъюнкта присвоение «$x_k = \overline{\alpha}$». Однако в целом механизмы разбора конфликтов аналогичны механизмам, используемым в современных SAT-решателях (см. [94]), поэтому здесь дополнительное внимание этим процедурам не уделяется.

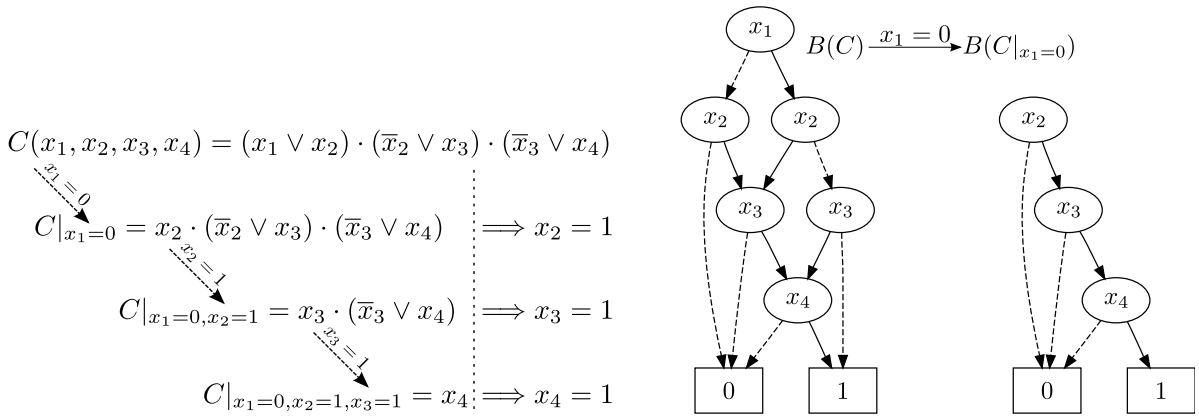


Рис. 2.8. (к теореме 2.6). Слева показана реализация BCP-стратегии применительно к КНФ $(x_1 \vee x_2) \cdot (\overline{x}_2 \vee x_3) \cdot (\overline{x}_3 \vee x_4)$ в результате подстановки $x_1 = 0$; справа показан результат подстановки $x_1 = 0$ в ROBDD, представляющую булеву функцию, которая выражается той же самой КНФ — требуется единственный обход ROBDD

Еще одним полезным свойством гибридного вывода является возможность естественной организации на ROBDD т. н. «отсроченных вычислений». Рассмотрим следующие условия, которые определяют ситуацию, в некотором роде двойственную ситуации возникновения ROBDD-следствия.

i.) Для некоторой переменной $x_q \in X$ в ROBDD $B(f)$ любой путь π из корня в терминальную вершину «0» обязательно проходит через некоторую вершину, помеченную переменной x_q.

ii.) Имеет место $\left|\Delta^0(x_q)\right| = 1$.

Установим справедливость следующей теоремы.

Теорема 2.7. Пусть $B(f)$ — произвольная ROBDD, и относительно некоторой переменной x_q в $B(f)$ справедливы условия i.) и ii.). Тогда в ROBDD $B(f)$ невозможны ROBDD-следствия ни для каких переменных из множества $X \setminus \{x_q\}$. Трудоемкость процедуры проверки условий i.)–ii.) ограничена сверху величиной $O\left(n \cdot |B(f)|\right)$.

Доказательство. Пусть в ROBDD $B(f)$ для некоторой переменной $x_q \in X$ выполняются условия i.) и ii.). Не ограничивая общности, полагаем, что $\Delta^0(x_q) = \{1\}$. Используя рассуждения, полностью аналогичные тем, посредством которых была доказана лемма 2.1, можно показать, что в

этом случае для любой вершины, помеченной переменной x_q, ее low-ребенком является терминальная «1».

Теперь предположим, что существует такая переменная $x_p \in X \setminus \{x_q\}$, для которой выполнены условия 1.)–2.) вывода некоторого ее ROBDD-следствия. Для данной переменной возможны следующие два варианта ее расположения относительно x_q в порядке означивания переменных в ROBDD $B(f)$:

$$1 : x_1 \prec \ldots \prec x_q \prec \ldots \prec x_p \prec \ldots$$

$$2 : x_1 \prec \ldots \prec x_p \prec \ldots \prec x_q \prec \ldots$$

Рассмотрим первый случай. Из вышесказанного следует, что low-ребенком любой вершины, помеченной переменной x_q, является терминальная вершина «1». Данный факт означает, что существует «обходной путь» из корня ROBDD в терминальную вершину «1», не проходящий через вершины, помеченные переменной x_p, то есть для данной переменной вывод ее ROBDD-следствия невозможен.

Рассмотрим второй случай. Как было отмечено выше, условия 1.) и 2.) означают, что одним из детей любой вершины, помеченной переменной x_p, является «0». Но тогда существуют «обходные пути» из вершин, помеченных x_p, в «0», не проходящие через вершины, помеченные x_q, что противоречит предположению о выполнимости условий i.) и ii.). Все проделанные рассуждения переносятся на случай $\Delta^0(x_q) = \{0\}$.

Отметим возможность совмещения процедур подстановки, проверки условий 1.)–2.) и условий i.)–ii.). В самом деле, проверка условий i.)–ii.) гарантируется использованием аналогов процедур *check_path_consistency()* и *check_for_zero_child()*, которые можно также «встроить» в процедуру *check_impl()*. При этом порядок сложности полученной процедуры останется прежним.

Все сказанное позволяет заключить, что сложность процедуры подстановки значений переменных в ROBDD с отслеживанием ситуаций возникновения ROBDD-следствий и выполнения относительно некоторых переменных (не обязательно одной) условий i.)–ii.) ограничена сверху вели-

чиной $O\left(n \cdot |B(f)|\right)$. Теорема 2.7 доказана. ■

Данная теорема позволяет сформировать механизмы отсроченных вычислений при подстановке выведенных в процессе гибридного вывода значений некоторых переменных: если для текущей ROBDD $B(f)$ выполнены i.)–ii.) относительно x_q и из КНФ-части выведено значение некоторой переменной $x_k, k \neq q$, нет смысла на данном этапе подставлять соответствующее значение в $B(f)$ — ничего нового выведено не будет. После присвоения или вывода из КНФ-части некоторого значения для x_q целесообразно осуществить в $B(f)$ подстановку сразу всех накопленных к этому моменту значений переменных, а также вывод всех возможных ROBDD-следствий, используя для этого процедуру *assign()*.

Приведем краткое резюме основных результатов данной главы, касающихся алгоритмики гибридного (SAT+ROBDD) подхода к задачам обращения полиномиально вычислимых дискретных функций. Итак, теоремы 2.2– 2.4 обосновывают гибридный подход в следующей форме. Некоторый алгоритм на основе DPLL действует в отношении КНФ $C(f)$, кодирующей задачу обращения функции f в некоторой точке. После каждого рестарта вместо чистки базы конфликтных дизъюнктов используется процедура построения ROBDD-представления характеристической функции системы вида (2.7). Дальнейший вывод идет как на исходной КНФ, так и на ROBDD, представляющей соответствующую базу накопленных ограничений. При этом на ROBDD действуют аналоги механизмов вывода, используемых в современных SAT-решателях на базе DPLL: аналог подстановки и правила единичного дизъюнкта реализован в виде процедуры *assign()*; аналогом CL-процедуры является применение алгоритма *Apply* к текущей ROBDD и новому ограничению-дизъюнкту; аналог механизма действия структур watched literals определяется условиями i.)–ii.) и теоремой 2.7.

Глава 3

Реализация и тестирование параллельных алгоритмов обращения дискретных функций, использующих BDD

В предыдущей главе была высказана идея использования ROBDD в качестве модифицируемых баз ограничений, порождаемых в процессе DPLL-вывода на КНФ, кодирующих задачи обращения полиномиально вычислимых дискретных функций. Данная идея представляется очень перспективной ввиду того, что ядро DPLL-вывода для таких КНФ содержит, как правило, относительно небольшое число переменных, и это позволяет надеяться на построение компактных ROBDD-представлений соответствующих функций.

В настоящей главе описана программная реализация основных алгоритмических составляющих гибридного (SAT+ROBDD)-подхода к обращению полиномиально вычислимых дискретных функций. Основу «ROBDD-части» данного подхода составляет ROBDD-решатель логических уравнений, алгоритмика которого была подробно описана во второй главе. Далее приводится описание базовых структур данных ROBDD-решателя и задействованные в нем специальные процедуры организации оперативной памяти. Построенный решатель сам по себе оказался полезным при исследовании некоторых математических моделей компьютерной биологии (приводятся результаты соответствующих численных экспериментов).

Весьма позитивным свойством гибридного (SAT+ROBDD)-вывода к обращению дискретных функций является малая размерность ROBDD, представляющих базы конфликтных дизъюнктов, порожденных в процессе нехронологического DPLL-вывода. Этот факт дает основу для принципиально нового подхода к решению задач обращения полиномиально вычислимых дискретных функций в распределенных вычислительных средах (РВС), поскольку делает реальной возможность межпроцессорных обменов

массивами накапливаемых ограничений, представляемыми в виде ROBDD. Заключительный раздел настоящей главы посвящен программной реализации данной идеи и тестированию соответствующего программного комплекса на задачах обращения некоторых криптографических функций.

3.1. Реализация и тестирование ROBDD-решателя логических уравнений

3.1.1. Базовые структуры данных

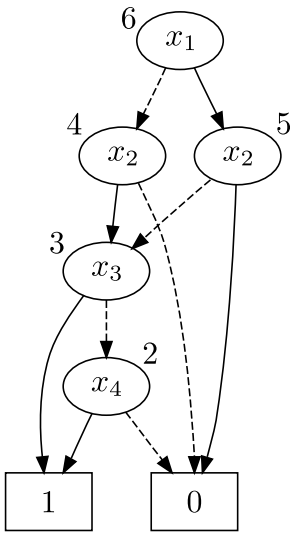
Для представления двоичных диаграмм решений в памяти ЭВМ удобно использовать таблицы. Таблица, представляющая ROBDD, состоит из трех столбцов.

Как было сказано выше, произвольная вершина v ROBDD (за исключением терминальных) определяется тремя «координатами»: переменной, приписанной данной вершине, вершиной, являющейся low-ребенком v, и вершиной, являющейся high-ребенком v.

Строки таблицы пронумерованы. Каждая строка соответствует определенной вершине ROBDD и, таким образом, содержит три позиции: описание переменной, номер строки с описанием low-ребенка и номер строки с описанием high-ребенка. 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 3.1. На рисунке 3.1 приведена ROBDD для функции $(x_1 \oplus x_2)\cdot$ $\cdot(x_3 \vee x_4)$ с порядком переменных $x_1 \prec x_2 \prec x_3 \prec x_4$ и представляющая ее в памяти ЭВМ таблица.

В главе 2 (пункт 2.1.1) описана общая схема представления логических уравнений двоичными деревьями. Интерпретация деревьев в памяти



No	var	low	height
0	—	—	—
1	—	—	—
2	x_4	0	1
3	x_3	2	1
4	x_2	0	3
5	x_2	3	0
6	x_1	4	5

Рис. 3.1. Представление ROBDD в памяти ЭВМ

ЭВМ осуществляется при помощи связанных списков. Каждый компонент списка представляет узел дерева, соответствующий функциональному элементу. Такой список может быть реализован, например, при помощи структуры «*term*», описание которой приведено ниже.

```
typedef struct term {
  operation op;
  int priority;
  struct term *left;
  struct term *right;
  int value;
} term;
```

Таблицу, которая представляет ROBDD в памяти ЭВМ, далее обозначаем через T.

При операциях с ROBDD важна возможность быстрой проверки наличия в ROBDD вершины с конкретными координатами. Данная проблема может быть решена за счет использования техники хеширования (см. [95]). То есть при обработке некоторой вершины v в текущем варианте таблицы просматриваются только те вершины, значение хеш-функции на которых совпадает со значением данной хеш-функции на вершине v.

3.1.2. Организация работы с памятью при построении ROBDD

При программной реализации приложений, использующих ROBDD, основным ограничителем эффективности является объём 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 объём логического выражения, представляемого в виде ROBDD, превышает объём оперативной памяти, проявляется эффект, известный как *cache trashing* (см. [96]). 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).

Основной объём памяти, используемой для представления ROBDD, занимают строки таблицы с информацией о вершинах ROBDD. Простейшая организация памяти может быть осуществлена с использованием менеджера памяти самой операционной системы (далее «системный менеджер»). Основной недостаток данного подхода в том, что при построении ROBDD системный менеджер выполняет многократное выделение и освобождение динамической памяти для представления каждой строки таблицы T. Это приводит, во-первых, к значительному росту времени работы решателя, а во-вторых, к тому, что объем памяти, используемый для представления произвольной строки таблицы T, превышает необходимый (за счет информации, требующейся системному менеджеру).

Повышение эффективности динамического выделения памяти для большого числа однотипных и имеющих одинаковый размер записей может быть достигнуто за счет использования собственного менеджера памяти. Далее описывается техника снижения расхода памяти при представлении отдельных вершин ROBDD, использующая специальный менеджер памя-

ти.

Основная идея предлагаемого подхода состоит в уменьшении числа обращений к системному менеджеру памяти. При этом результатом одного обращения к системному менеджеру является выделение «большого» фрагмента памяти сразу под n строк таблицы T. Далее используемый решателем собственный менеджер памяти самостоятельно распределяет выделенный фрагмент между строками таблицы: доступная решателю память последовательно заполняется создаваемыми в таблице строками с номерами от 0 до $n-1$. Строка с номером n «не помещается» в выделенный фрагмент, поэтому в момент ее создания принимается решение о запросе к системному менеджеру на выделение очередного фрагмента памяти под n строк.

При удалении из таблицы T некоторой строки память, занимаемая ею, помечается как свободная и может быть использована повторно для хранения других строк таблицы T.

Общая экономия памяти при использовании описанного подхода составляет 25% и более в зависимости от вида выравнивания, используемого системным менеджером памяти. Помимо сокращения объема используемой памяти данная техника позволяет организовать эффективные процедуры выделения и освобождения памяти, доступа к произвольной строке таблицы T и перечисления ее строк.

Базовая идея описанного подхода не нова и применяется во многих алгоритмах, активно эксплуатирующих оперативную память (например, в задачах вычислительной геометрии, [97]).

3.1.3. Применение ROBDD-решателя логических уравнений к исследованию дискретно-автоматных моделей генных сетей

В данном пункте приводятся результаты тестирования ROBDD-решателя логических уравнений, построенного в соответствии с принципами, изложенными в главе 2 (пункт 2.1), и использующего техники организации

данных, перечисленные в пунктах 3.1.1 – 3.1.2.

Данный ROBDD-решатель был применен к задачам поиска неподвижных точек одного класса автоматных отображений. По сути речь идет об автоматах, моделирующих поведение генных сетей и, в связи с этим, активно исследуемых в последние годы в компьютерной биологии. (см. [16, 98, 99]).

Рассматривается автоматное отображение, заданное следующей пороговой функцией (см. [98]):

$$A_G(x_i) = x_i^1 = \begin{cases} x_i + 1, & \text{если } \sum\limits_{x_j \in X_i} x_j = 0 \text{ и } x_i < p-1 \\ x_i - 1, & \text{если } \sum\limits_{x_j \in X_i} x_j > 0 \text{ и } x_i > 0 \\ x_i, & \text{иначе} \end{cases} \tag{3.1}$$

Здесь $G = G(U, D)$ — ориентированный связный граф без петель и кратных дуг, $|U| = n$. Каждой вершине $i \in U$ приписан целый вес $x_i \in \{0, \ldots, p-1\}$, $i \in \{1, \ldots, n\}$, задающий концентрацию белка в данной вершине; натуральное число p, $p \geq 2$, при этом определяет верхнюю границу значения веса по всем вершинам G и называется значностью. Через $\tilde{x} = (x_1, \ldots, x_n)$ обозначается начальный набор весов вершин графа G; X_i — множество весов тех вершин из $U \setminus \{i\}$, дуги из которых ведут в вершину i (при $X_i = \emptyset$ полагаем, что $\sum\limits_{x_j \in X_i} x_j = 0$), через x_i^1 обозначено новое значение веса вершины i, $i \in \{1, \ldots, n\}$, тем самым $A_G(\tilde{x}) = \tilde{x}^1$. Неподвижной точкой отображения A_G называется такой вектор $\tilde{x} = (x_1, \ldots, x_n)$, что $A_G(\tilde{x}) = \tilde{x}$ (пример графа A_G см. на рисунке 3.2).

В работе [100] задача поиска неподвижных точек автоматных отображений с пороговой функцией вида (3.1) была сведена к задаче поиска решений системы логических уравнений следующего вида:

$$\begin{cases} \left(x_1 \equiv \bigwedge\limits_{x_{j_1} \in X_1} \overline{x}_{j_1} \right) = 1 \\ \cdots \\ \left(x_n \equiv \bigwedge\limits_{x_{j_n} \in X_n} \overline{x}_{j_n} \right) = 1 \end{cases} \tag{3.2}$$

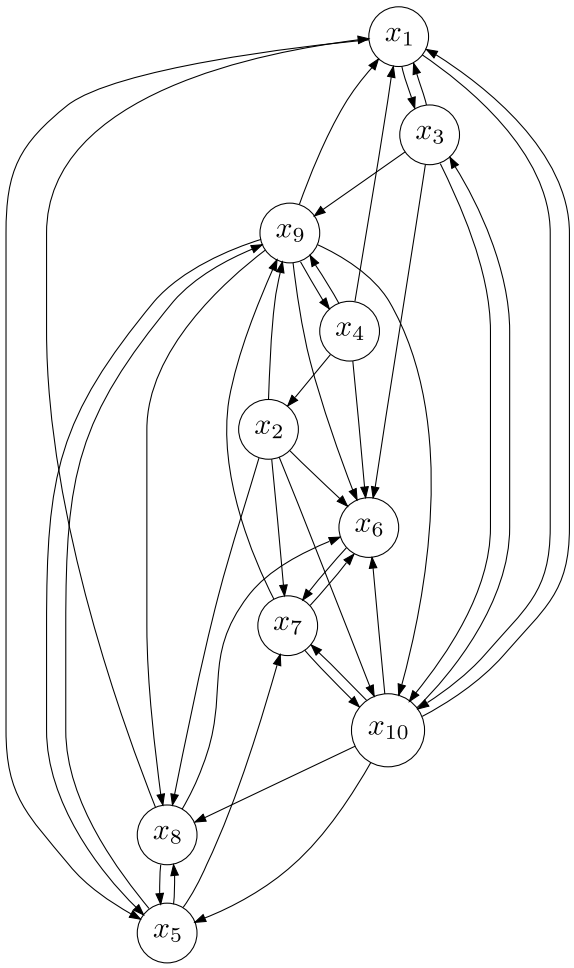


Рис. 3.2. Пример графа, представляющего регуляторный контур генной сети на 10-ти вершинах. Граф генерировался случайным образом в виде булевой матрицы смежности с нулевой главной диагональю; элемент a_{ij}, $i \neq j$, принимает значение 1 с фиксированной вероятностью $P_{ij} = 0{,}5$

Для решения систем вида (3.2) использовался ROBDD-решатель, архитектура и основные механизмы работы которого описаны выше. При этом сравнивались по эффективности следующие два режима работы:

1. в первом режиме порядок означивания переменных в ROBDD формировался в соответствии с эвристикой, приведенной в пункте 2.1.1, кроме этого, использовалась техника разбиения системы на слои, которая описана в пункте 2.1.2;

2. во втором режиме разбиение системы на слои не осуществлялось.

Во всех численных экспериментах использовались функциональные графы, определяющие структуру автоматных отображений (3.1), сгенери-

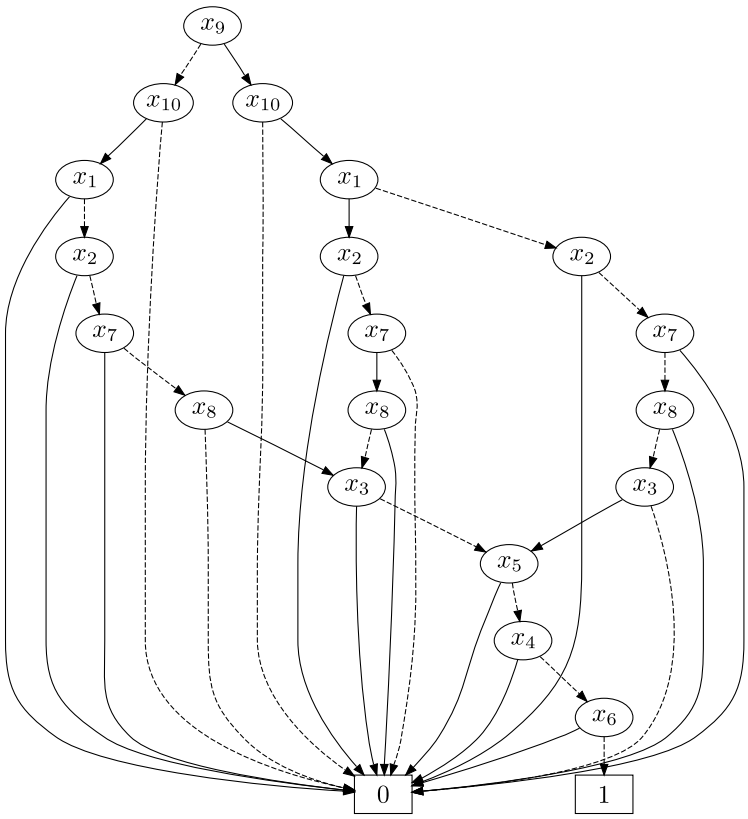


Рис. 3.3. ROBDD-представление характеристической функции системы вида (3.2) для генной сети, приведенной на рисунке 3.2

рованные в форме случайно порождаемых матриц смежности с нулевой главной диагональю и с фиксированной «вероятностью дуги» от i к j (параметр P_{ij}).

Отметим здесь, что использование описанной во второй главе эвристики, строящей разбиение системы логических уравнений на слои, привело к значительному росту эффективности ROBDD-решателя. Как видно из приведенных ниже результатов, на ряде тестов вычисления прерывались из-за переполнения оперативной памяти.

Число вершин в графе A_G	Вероятность P_{ij}							
	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9
50	0	0	1	1	1	0	0	0
60	2	13	17	5	2	1	0	0
70	25	32	48	18	13	5	1	0
80	23	138	129	64	23	10	3	1
90	190	736	960	532	60	38	5	1
100	—	—	—	769	253	67	9	2

Таблица 3.1. Результаты работы ROBDD-решателя в первом режиме (время указано в секундах)

Число вершин в графе A_G	Вероятность P_{ij}							
	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9
50	11	16	10	6	4	2	1	1
60	149	121	72	23	12	6	3	2
70	970	410	206	118	40	20	10	4
80	9328	3261	1371	535	147	47	19	8
90	—	15685	5203	1325	447	106	37	16
100	—	—	—	2826	817	271	66	23

Таблица 3.2. Результаты работы ROBDD-решателя во втором режиме (время указано в секундах)

3.2. Параллельная реализация гибридного (SAT+ROBDD)-подхода к решению задач обращения дискретных функций

Теоретические основы гибридного подхода были описаны во второй главе. Все приведенные выше алгоритмы были программно реализованы и объединены в программный комплекс, который далее называется гибридным (SAT+ROBDD)-решателем (см. [101, 102]).

3.2.1. Реализация и тестирование последовательного гибридного (SAT+ROBDD)-решателя

«SAT-часть» гибридного решателя использует SAT-решатель, в котором нехронологический DPLL-вывод организован на множестве переменных входа функции, задача обращения которой решается (см. теорема 2.2, глава 2). Данный решатель, названный «coresat», был создан на базе широко известного решателя minisat (см. [103]). Особенность анализа конфликтов в coresat состоит в том, что при построении конфликтного дизъюнкта используется только информация о переменных ядра, породивших рассматриваемый конфликт (это достигается при помощи техники характеристических векторов). В результате обратный ход по графу вывода при построении конфликтного дизъюнкта является очень коротким.

ROBDD-часть гибридного решателя фактически представляет собой описанный ранее оригинальный ROBDD-решатель, ориентированный на логические уравнения вида $D = 1$, где D — произвольный дизъюнкт.

При гибридном выводе подстановки осуществляются сначала в ROBDD-часть, а затем в SAT-часть. При этом используются все описанные во второй главе алгоритмы работы с ROBDD, выступающей в роли базы булевых ограничений.

Взаимодействие SAT и ROBDD-компонент гибридного решателя осуществляется в соответствии со схемой, приведенной на рисунке 3.4.

Согласно данной схеме работа последовательного гибридного (SAT++ROBDD)-решателя определяется перечисленными ниже действиями.

1. На начальном этапе работает coresat, результатом чего является накопление некоторого массива конфликтных дизъюнктов, причем каждый состоит из литералов над множеством переменных входа рассматриваемой функции.

2. Работа coresat прерывается и к накопленному массиву конфликтных дизъюнктов применяется ROBDD-решатель, который строит ROBDD-представление характеристической функции соответствующей систе-

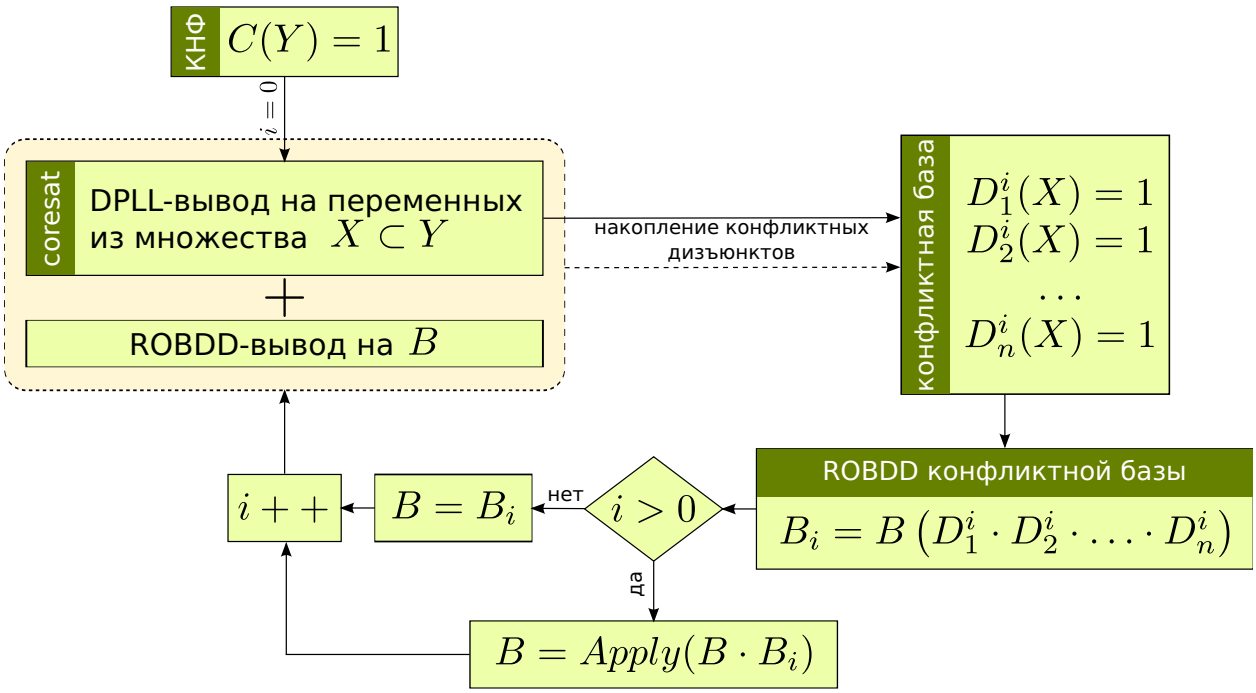


Рис. 3.4. Схема работы гибридного (SAT+ROBDD)-решателя

мы, образованной уравнениями вида $D = 1$.

3. Результатом каждой итерации является новая ROBDD, полученная применением *Apply* к текущей ROBDD и накопленному на данной итерации массиву конфликтных дизъюнктов.

4. Процесс продолжается итеративно и завершается либо нахождением выполняющего набора, либо констатацией невыполнимости КНФ, если решатель используется для обработки параллельного списка SAT-задач, среди которых есть невыполнимые. Такие ситуации возможны, например, при крупноблочном распараллеливании исходной SAT-задачи, кодирующей процедуру обращения рассматриваемой дискретной функции (см. ниже).

Как уже говорилось выше, порядок угадывания переменных выстраивается в соответствии с накапливаемой статистикой конфликтности (см. пункт 1.2.2). Именно статистика конфликтности определяет и порядок означивания переменных при построении ROBDD. Статистика конфликтности меняется от итерации к итерации (впрочем, обычно весьма незначительно).

Изменение порядка в построенной ROBDD осуществляется при помощи описанного во второй главе алгоритма.

Последовательный гибридный (SAT+ROBDD)-решатель, описание которого приведено выше, получил название «hsat»

Тестирование данного решателя, как и его параллельной версии, проводилось на SAT-задачах, являющихся ослабленными вариантами задачи криптоанализа известного генератора ключевого потока A5/1. Напомним, что, в соответствии с [104], генератор A5/1 имеет вид, представленный на рисунке 3.5.

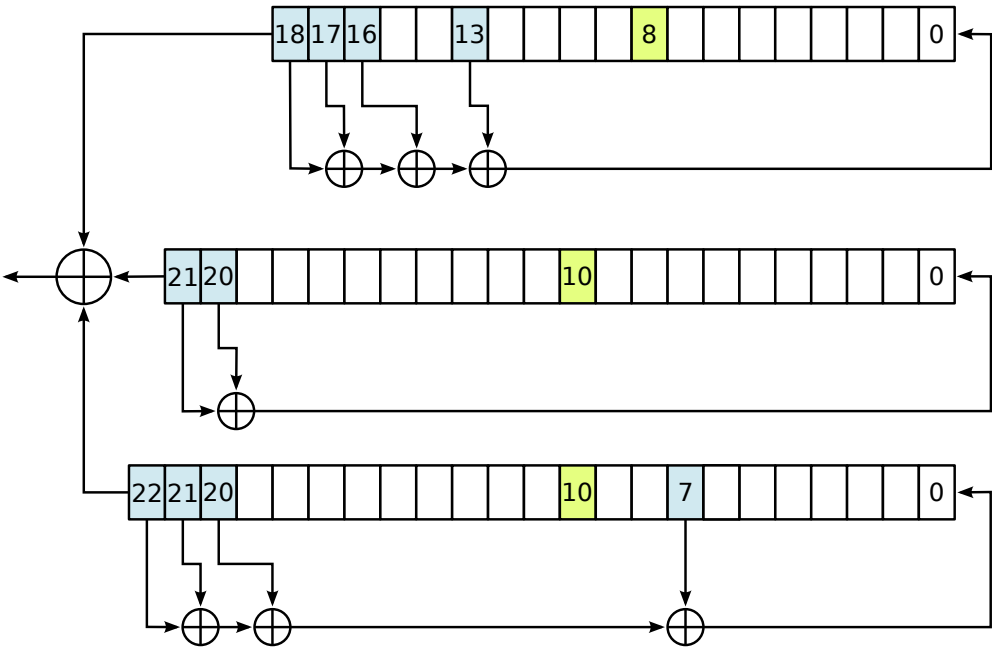


Рис. 3.5. Схема работы генератора A5/1

В генераторе A5/1 используются 3 *регистра сдвига с линейной обратной связью* (РСЛОС или LFSR, см., например, [105]), задаваемые следующими полиномами обратной связи:

$$\text{РСЛОС 1: } X^{19}+X^{18}+X^{17}+X^{14}+1;\ \text{РСЛОС 2: } X^{22}+X^{21}+1;$$
$$\text{РСЛОС 3: } X^{23}+X^{22}+X^{21}+X^{8}+1.$$

В каждом такте могут сдвигаться не все регистры. Сдвиг регистра с номером r, $r \in \{1,2,3\}$, происходит, если значение $\chi_r\left(b_s^1, b_s^2, b_s^3\right)$ равно 1, и не происходит, если значение данной функции равно 0. Через b_s^1, b_s^2, b_s^3

здесь обозначены значения т. н. «серединных битов» текущего шага, то есть битов, находящихся в данный момент в девятой ячейке первого РСЛОС, и в ячейках с номером «11» РСЛОС 2 и РСЛОС 3. Данные ячейки на рисунке отмечены более темной заливкой, нумерация ячеек ведется справа налево. Функция $\chi_r(\cdot)$ определяется следующим образом:

$$\chi_r\left(b_s^1, b_s^2, b_s^3\right) = \begin{cases} 1, & b_s^r = majority\left(b_s^1, b_s^2, b_s^3\right) \\ 0, & b_s^r \neq majority\left(b_s^1, b_s^2, b_s^3\right) \end{cases},$$

где $majority(A, B, C) = A \cdot B \vee A \cdot C \vee B \cdot C$ — функция большинства.

В каждом такте с регистров снимаются старшие биты, складываются по модулю 2, результирующий бит является битом ключевого потока данного такта. Требуется найти начальное заполнений всех трех регистров по некоторой известной последовательности битов ключевого потока и по известному алгоритму их порождения.

В работах [106] был предложен подход к решению задачи криптоанализа данного генератора, использующий идеи крупноблочного параллелизма (см. [71, 107, 108]).

При крупноблочном распараллеливании выделяется некоторое множество $X' \subset X$, X — множество переменных входа рассматриваемой дискретной функции, после чего в КНФ, кодирующую задачу обращения, подставляются векторы, образованные всевозможными значениями истинности переменных из X'. В результате строится т. н. «декомпозиционное семейство» $\Delta\left(X'\right)$, содержащее $2^{|X'|}$ КНФ. Данное семейство обрабатывается как параллельный список несвязанными вычислительными процессами.

Декомпозиционное множество, которое дает наилучшие результаты по итоговой трудоемкости криптоанализа, состоит из 31 переменной и символически отображено на рисунке 3.6 ([106]).

В соответствии с данной схемой предлагается включить в X' переменные, кодирующие начальные состояния ячеек регистров, начиная с первых ячеек, до ячеек, содержащих серединные биты (соответствующие ячейки на рисунке отображены темной заливкой). Иными словами, имеем деком-

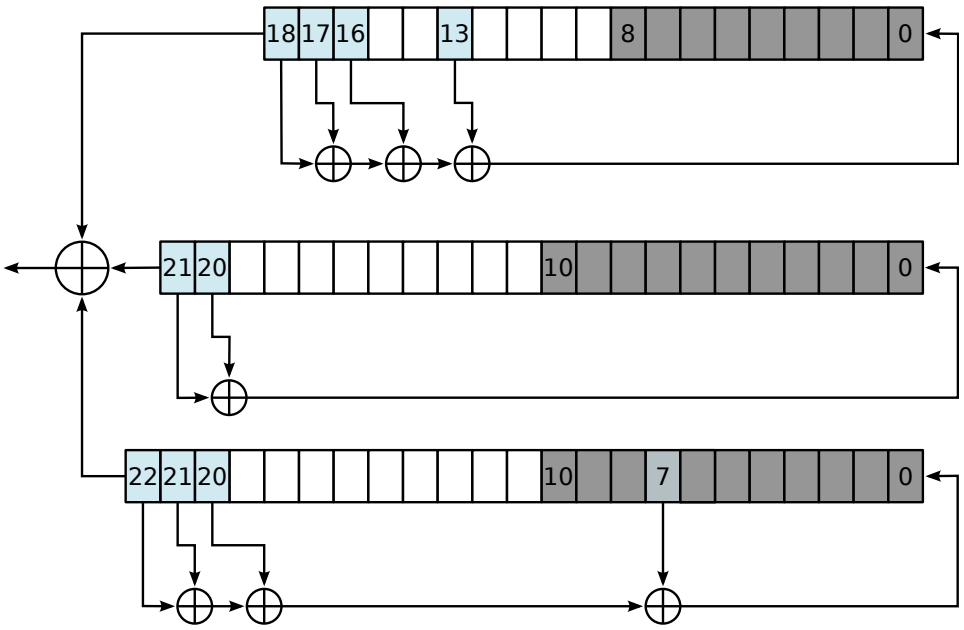


Рис. 3.6. Схема построения декомпозиционного множества из 31 переменной (X')

позиционное множество

$$X' = \{x_1, \ldots, x_9, x_{20}, \ldots, x_{30}, x_{42}, \ldots, x_{52}\}. \tag{3.3}$$

Описанный выше решатель hsat проигрывает на тестах из $\Delta(X')$ (X' построено в соответствии с (3.3)) решателю dminisat (см. [106]). Однако его эффективность существенно превосходит эффективность dminisat на более сложных тестах, а именно на КНФ из декомпозиционного семейства $\Delta\left(\tilde{X}\right)$, где $\tilde{X}$ — декомпозиционное множество, состоящее из 22 переменных и построенное в соответствии со схемой, представленной на рисунке 3.7. Тем самым

$$\tilde{X} = \{x_4, \ldots, x_9, x_{23}, \ldots, x_{30}, x_{45}, \ldots, x_{52}\}. \tag{3.4}$$

Ниже выборочно приведены результаты работы hsat на 50 SAT-задачах, выбранных случайным образом из $\Delta\left(\tilde{X}\right)$ ($\tilde{X}$ построено в соответствии с (3.4)). Было проведено сравнение данного решателя с решателем dminisat.

Как видно из представленной таблицы, на рассматриваемом классе тестов hsat продемонстрировал существенно большую эффективность по сравнению с dminisat. При этом общее время работы решателя hsat соста-

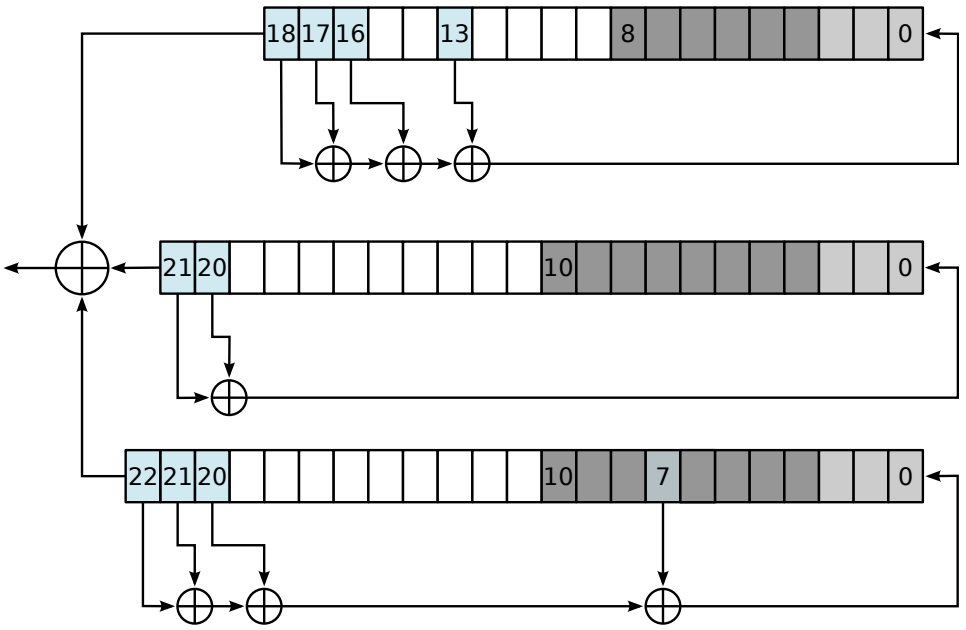


Рис. 3.7. Схема построения декомпозиционного множества из 22 переменной ($\tilde{X}$)

	Номер теста									
	1	2	3	4	5	6	...	48	49	50
hsat	332	259	551	915	289	268	...	1928	477	334
dminisat	3924	1518	1194	1439	1680	1825	...	2841	652	720

Таблица 3.3. Сравнение эффективности работы решателей hsat и dminisat на 50 SAT-задачах, выбранных случайным образом из $\Delta\left(\tilde{X}\right)$

вило 28182 секунды, dminisat — 132203 секунды. Таким образом, на данном классе тестов hsat оказался эффективнее в среднем в 4,69 раза.

3.2.2. Реализация и тестирование параллельного гибридного (SAT+ROBDD)-решателя, функционирующего в MPI-среде

Одной из первых работ по параллелизму в SAT-задачах является статья [54], в которой была описана схема распараллеливания процедуры Дэвиса и Патнема, предполагающая межпроцессорное взаимодействие с целью балансировки загрузки процессоров. Первые программные реализации параллельных SAT-решателей в широком доступе были выставлены лишь

на конкурсе SAT-Race 2008 года ([61, 62, 109–111]).

Решатель MiraXT (см. [62]) является одним из лучших в упомянутом конкурсе. Именно с этим решателем проводилось сравнение параллельного варианта гибридного (SAT+ROBDD)-решателя (другие параллельные SAT-решатели (см. [110]) с рассматриваемыми криптографическими тестами за приемлемое время не справлялись). Следует отметить, что фактически MiraXT является многопоточным приложением и, следовательно, не может задействовать более одного рабочего узла кластера. В статье [62] была анонсирована версия данного решателя, реализованная с использованием стандарта MPI, однако найти эту версию в открытых источниках не удалось.

Описанный выше гибридный (SAT+ROBDD)-решатель был реализован в форме MPI-приложения и получил название mhsat. Основной его конструктивной особенностью является обмен в MPI-среде массивами конфликтных ограничений, представленными в виде ROBDD. Компактность последних делает соответствующие процедуры весьма эффективными. Особо отметим, что в доступных параллельных SAT-решателях (в частности, в MiraXT) фактический обмен ограничениями не производится, поскольку используется общая память (что само по себе не решает проблемы ее переполнения). На рисунке 3.8 описана общая схема межпроцессорных взаимодействий в решателе mhsat.

В соответствии с данной схемой на k вычислительных ядра кластера подаются k вариантов исходной SAT-задачи (в приведенном примере $k = 8$). На каждом ядре решение задачи стартует с собственным порядком угадывания переменных. Происходит параллельное накопление ограничений, после чего все ядра обмениваются накопленными ограничениями в соответствии с приведенной схемой.

Процесс работы mhsat можно рассматривать как последовательную реализацию перечисленных ниже этапов:

1. Этап независимого накопления каждым из ядер конфликтных ограничений в виде ROBDD, которые строятся в соответствии с локаль-

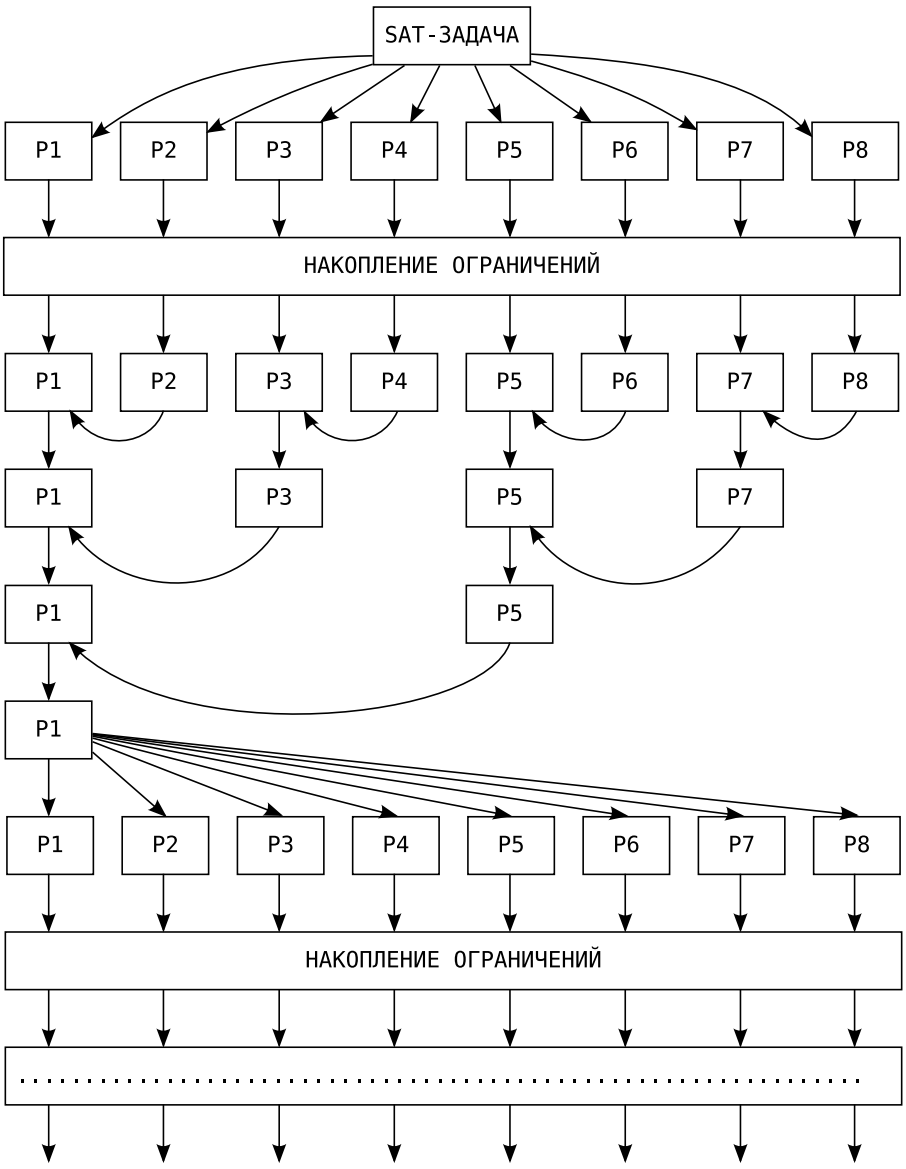


Рис. 3.8. Общая схема межпроцессорных взаимодействий в mhsat

ными для каждого ядра статистиками конфликности переменных.

2. Этап объединения баз накопленных конфликтных ограничений:

   а. Обмен локальными статистиками конфликтности переменных с целью построения общего порядка их означивания;

   б. Решение каждым вычислительным ядром проблемы модификации локальной ROBDD в соответствии с общим порядком, полученным на предыдущем шаге;

   в. Этап последовательного обмена базами конфликтных ограничений и объединения их в одну ROBDD при помощи алгоритма

Apply. Результатом выполнения данного этапа является ROBDD-представление общей базы конфликтных ограничений, построенное на первом вычислительном ядре в соответствии с общим порядком означивания переменных;

г. Этап распространения данной ROBDD между всеми вычислительными ядрами;

д. Решение каждым вычислительным ядром проблемы модификации полученной ROBDD в соответствии с порядком, соответствующим локальной статистике конфликтности переменных.

Решатель mhsat тестировался на 50 тестах, представляющих из себя КНФ из декомпозиционного семейства $\Delta\left(\tilde{X}\right)$, где $\tilde{X}$ — декомпозиционное множество, состоящее из 20 переменных:

$$\tilde{X} = \{x_4, \ldots, x_9, x_{24}, \ldots, x_{30}, x_{46}, \ldots, x_{52}\}. \tag{3.5}$$

В тестовых экспериментах данный решатель (подобно параллельным решателям, принимавшим участие в конкурсе SAT-Race 2008) запускался на всех ядрах одного четырехядерного процессора (Intel® Xeon® E5345 с тактовой частотой 2,33 ГГц). Проводилось сравнение эффективности mhsat с эффективностью решателя MiraXT. Помимо этого решатель mhsat сравнивался по эффективности с hsat, задействующим одно вычислительное ядро, и dminisat, решающим те же тестовые задачи, распараллеленные в соответствии с крупноблочным подходом (см. [106]) на 4 подзадачи (каждому ядру процессора выдавалась отдельная подзадача, никаких межпроцессорных обменов при этом не производилось). Общая схема данного эксперимента проиллюстрирована на рисунке 3.9.

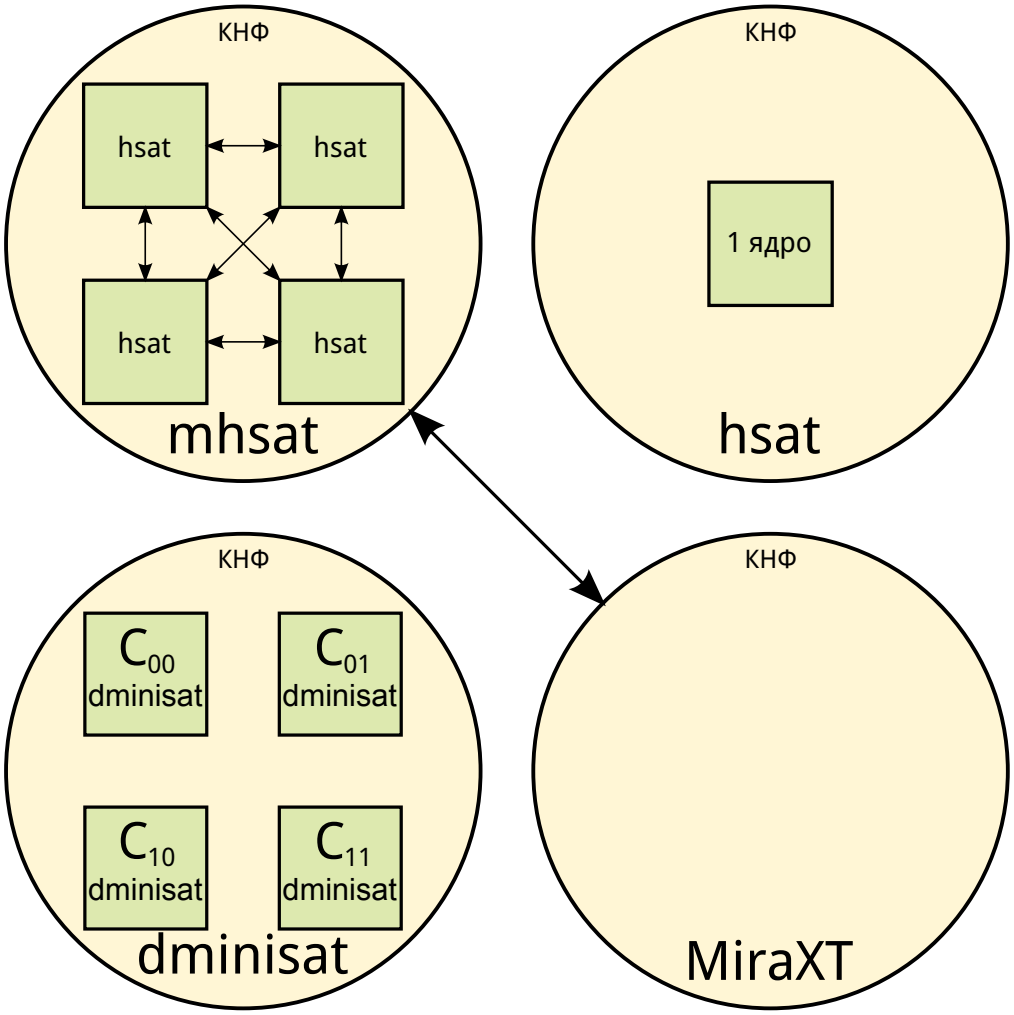


Рис. 3.9. Общая схема эксперимента по сравнению эффективности решателей mhsat, hsat, MiraXT и dminisat

Результаты численного эксперимента

	mhsat	hsat	MiraXT	dminisat			
				КНФ C_{00}	КНФ C_{01}	КНФ C_{10}	КНФ C_{11}
1	454	2255	1674	3924	2563	2376	4405
2	446	1434	1090	3098	3463	3005	2520
3	283	2552	3126	1518	2254	2285	1748
4	865	3695	> 4000	1194	2841	3178	2859
5	701	1595	1230	2906	2457	2320	1865
6	1196	5143	> 4000	1439	1351	2070	2820

Продолжение на следующей странице

Начало на предыдущей странице

	mhsat	hsat	MiraXT	dminisat			
				КНФ C_{00}	КНФ C_{01}	КНФ C_{10}	КНФ C_{11}
7	483	1928	> 4000	1680	2357	1580	1343
8	555	1614	2847	2927	2044	1713	1058
9	351	894	547	1825	1608	2366	874
10	876	1833	3138	3792	4279	4018	3530
11	950	2162	1070	992	2558	2123	3818
12	1056	3619	3180	8230	4720	5554	5977
13	546	3642	> 4000	4857	4904	2939	4002
14	415	2402	692	2128	2086	870	1122
15	1055	8903	1414	3568	2841	3121	3066
16	811	7001	1334	4495	4377	4298	3494
17	626	1566	411	1113	1470	862	418
18	773	1709	1294	2203	3128	3593	3161
19	909	2792	1090	1844	2174	1704	1221
20	832	2771	504	1051	1489	848	1249
21	607	1557	563	1031	1222	918	1168
22	588	1575	1746	1945	2587	2316	2061
23	283	1438	465	654	794	1092	655
24	316	1660	1138	1457	2007	1735	1723
25	1046	7422	2169	5400	6031	4486	2474
26	370	1104	1156	2578	2377	2427	1033
27	1762	4062	2354	6362	5146	6975	1661
28	1024	3243	2388	1635	1672	1171	1100
29	1060	2795	3394	6647	5596	4558	3563
30	458	2613	777	2575	1429	2220	1958
31	352	782	762	923	1850	1577	1620
32	579	2003	2618	1660	1885	4408	2486
33	472	940	723	1674	1543	720	1357

Продолжение на следующей странице

Начало на предыдущей странице

	mhsat	hsat	MiraXT	dminisat			
				КНФ C_{00}	КНФ C_{01}	КНФ C_{10}	КНФ C_{11}
34	633	3824	2158	5856	2430	9783	2985
35	1039	4809	2283	5486	5632	5175	4622
36	420	1077	980	1090	1636	2539	2476
37	653	2968	1088	2159	2287	909	1191
38	417	1612	267	628	710	551	439
39	596	1347	> 4000	2114	2311	2653	2036
40	538	3249	3151	2405	4358	3523	3384
41	683	4907	3603	3574	2633	2893	2543
42	568	1172	1208	2896	1421	3127	2704
43	706	1581	3451	1810	3705	2834	2477
44	428	5507	642	1013	1726	1529	1800
45	680	4023	2067	4569	6592	2109	2283
46	354	1038	903	1563	1251	2023	1943
47	433	1370	> 4000	1677	1451	2253	2048
48	416	2234	611	1999	1856	1281	310
49	376	7089	1864	3387	3318	3359	2831
50	523	1465	615	652	2228	889	1221

Таблица 3.4. Результаты тестирования в секундах (подробно)

В таблице 3.4 приведены подробные результаты численного эксперимента. Гибридный решатель, работающий в MPI-среде, оказался быстрее остальных решателей в 46-ти тестах из 50-ти. В 4-х тестах лучшее время продемонстрировал решатель MiraXT. Следует также отметить, что в MiraXT встроено ограничение, запрещающее данному решателю работать более 4000 секунд. В проведенном численном эксперименте данное ограничение «сработало» 6 раз.

В таблице 3.5 представлена краткая информация по вычислительному эксперименту (худшие, лучшие и средние временные показатели для рассматриваемых решаталей).

		Решатель			
		mhsat	hsat	dminisat	MiraXT
Время	Лучшее	283	782	310	267
	Худшее	1762	8903	9783	> 4000
	Среднее	651,26	2799,52	3336,46	$> 1875,7$

Таблица 3.5. Результаты тестирования в секундах (кратко)

		Решатель		
		hsat	dminisat	MiraXT
Превосходство	Лучшее	18,85	15,45	11,05
	Худшее	1,99	1,63	0,61
	Среднее	4,3	5,12	$> 2,88$

Таблица 3.6. Среднее превосходство mhsat перед другими решателями (в число раз)

Основываясь на проведенных численных экспериментах, можно сделать следующие выводы:

- на рассмотренном классе тестов самым эффективным решателем оказался mhsat; при этом MiraXT отстает от него более, чем в 2,88 раза;
- mhsat в режиме 4-х ядер оказался быстрее своей последовательной версии в среднем в 4,3 раза (сверхлинейное ускорение);
- mhsat в режиме 4-х ядер демонстрирует лучшие результаты по сравнению с крупноблочной схемой распараллеливания задачи с использованием dminisat в среднем в 5,12 раза;
- крупноблочная схема с использованием dminisat проигрывает даже последовательной реализации гибридного решателя (hsat).

Заключение

Основная цель настоящей диссертации состояла в разработке алгоритмов, сочетающих в себе элементы SAT- и ROBDD-подходов и применимых к задачам обращения дискретных функций, возникающим в практических приложениях. Перечислим основные результаты, полученные в диссертации.

1. Разработан новый вычислительный метод решения задач обращения полиномиально вычислимых функций, базирующийся на гибридном (SAT+ROBDD) логическом выводе; основу метода составили новые алгоритмы логического вывода на ROBDD и ROBDD-аналоги основных механизмов, применяемых в нехронологическом DPLL-выводе; базовые свойства всех алгоритмов обоснованы в форме теорем, построены оценки их вычислительной трудоемкости.

2. Разработан ROBDD-решатель систем логических уравнений, использующий оригинальные эвристические алгоритмы разбиения рассматриваемых систем на слои и эффективные процедуры управления оперативной памятью ЭВМ. Решатель показал высокую эффективность на задачах исследования некоторых автоматных моделей генных сетей.

3. Разработан программный комплекс, реализующий метод гибридного (SAT+ROBDD)-вывода и ориентированный на решение задач обращения полиномиально вычислимых дискретных функций. Данный программный комплекс реализован с использованием стандарта MPI и предназначен для работы в распределенных вычислительных средах. Принципиальная новизна комплекса состоит возможности эффективного обмена в распределенных средах булевыми ограничениями, представляемыми в виде ROBDD.

4. Проведено тестирование построенного программного комплекса на аргументированно трудных задачах обращения некоторых крипто-

графических функций. Эффективность комплекса на рассмотренных классах тестов оказалась существенно выше эффективности известных программных систем.

Резюмируя сказанное, отметим, что основным теоретическим результатом диссертации является представленный в ней новый вычислительный метод обращения полиномиально вычислимых функций, включающий в себя новые алгоритмы обработки дискретных данных, которые могут быть использованы в решении практических задач широкого спектра (исследование дискретно-автоматных моделей, обращение дискретных функций, криптоанализ и т. п.). Основным практическим результатом диссертации является программный комплекс, в котором были реализованы все разработанные алгоритмы.

Литература

1. Bryant R. E. Graph-Based Algorithms for Boolean Function Manipulation // IEEE Transactions on Computers. 1986. Vol. 35, no. 8. Pp. 677–691.

2. Мендельсон Э. Введение в математическую логику. Москва: «Наука», 1971. С. 320.

3. Яблонский С. В. Введение в дискретную математику. Москва: «Наука», 1986. С. 384.

4. Нигматуллин Р. Г. Сложность булевых функций. Москва: «Наука», 1991. С. 240.

5. Сэвидж Дж. Э. Сложность вычислений. Москва: «Факториал», 1998. С. 368.

6. Закревский А. Д. Логические уравнения. Минск: Наука и техника, 1975. С. 94.

7. Rudeanu S. Boolean functions and equations. Amsterdam-London: North-Holland Publ. Comp., 1974. P. 442.

8. Катленд Н. Вычислимость. Введение в теорию рекурсивных функций. Москва: «Мир», 1983. С. 256.

9. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. Москва: «Мир», 1982. С. 416.

10. Stockmeyer L. Classifying of computational complexity of problems // The Journal of Symbolic Logic. 1987. Vol. 52, no. 1. Pp. 1–43.

11. Стокмейер Л. Классификация вычислительной сложности проблем // Кибернетический сборник. Новая серия. 1989. Т. 26. С. 20–83.

12. Семенов А. А. О сложности обращения дискретных функций из одного класса // Дискретный анализ и исследование операций. 2004. Т. 11, № 4. С. 44–55.

13. Hachtel G. D., Somenzi F. Logic synthesis and verification algorithms. Kluwer Ac. Publ., 2002.

14. Семенов А. А., Отпущенников И. В., Кочемазов С. Е. Пропозициональный подход в задачах тестирования дискретных автоматов // Современные технологии. Системный анализ. Моделирование. 2009. Т. 4. С. 48–56.

15. Семенов А. А., Заикин О. С., Беспалов Д. В., Ушаков А. А. SAT-подход в криптоанализе некоторых систем поточного шифрования // Вычислительные технологии. 2008. Т. 13, № 6. С. 134–150.

16. Системная компьютерная биология // Под ред. Н. А. Колчанова, С. С. Гончарова, В. А. Лихошвая, В. А. Иванисенко. Новосибирск: Изд-во СО РАН, 2008. С. 767.

17. Cook S. A. The complexity of theorem-proving procedures // Proceedings of the third Annual ACM Symposium on Theory of Computing. 1971. Vol. 35, no. 8. Pp. 151–159.

18. Кук С. А. Сложность процедур вывода теорем // Кибернетический сборник. Новая серия. 1975. Т. 12. С. 5–15.

19. Семенов А. А. Трансляция алгоритмов вычисления дискретных функций в выражения пропозициональной логики // Прикладные алгоритмы в дискретном анализе. Серия: Дискретный анализ и информатика. 2008. Т. 2. С. 70–98.

20. Buchberger B. Gröbner-Bases: An Algorithmic Method in Polynomial Ideal Theory // Multidimensional Systems Theory. 1985. Vol. 6. Pp. 184–232.

21. Кокс Д., Литтл Дж., О'Ши Д. Идеалы, многообразия и алгоритмы. Москва: «Мир», 2000. С. 687.

22. Агибалов Г. П. Логические уравнения в криптоанализе генераторов ключевого потока // Вестник Томского гос. ун-та. Приложение. 2003. № 6. С. 31–41.

23. Тимошевская Н. Е. Задача о кратчайшем линеаризационном множестве // Вестник Томского гос. ун-таа. Приложение. 2005. № 14. С. 79–83.

24. Тимошевская Н. Е. О линеаризационно эквивалентных покрытиях // Вестник Томского гос. ун-та. Приложение. 2005. № 14. С. 84–91.

25. Büttner W., Simonis H. Embedding Boolean expressions into logic programming // Journal of Symbolic Computation. 1987. Vol. 4. Pp. 191–205.

26. Brown F. M. Boolean Reasoning: The Logic Of Boolean Equations. Dover Publications, 2003. P. 304.

27. Meinel Ch., Theobald T. Algorithms and Data Structures in VLSI-Design: OBDD-Foundations and Applications. Springer-Verlag, 1998.

28. Цейтин Г. С. О сложности вывода в исчислении высказываний // Записки научных семинаров ЛОМИ АН СССР. 1968. Т. 8. С. 234–259.

29. Tseitin G. On the complexity of derivation in propositional calculus // Studies in Constructive Mathematics and Mathematical Logic. 1968. Vol. 2. Pp. 234–259.

30. Семенов А. А. О преобразованиях Цейтина в логических уравнениях // Прикладная дискретная математика. 2009. № 4. С. 28–50.

31. Пападимитриу Х., Стайглиц К. Комбинаторная оптимизация. Алгоритмы и сложность. Москва: Мир, 1985. С. 510.

32. SATLive! URL: `http://www.satlive.org/`.

33. Черемисинова Л. Д., Новиков Д. Я. Проверка схемной реализации частичных булевых функций // Вестник Томского гос. ун-та. Управление, вычислительная техника, информатика. 2008. № 4 (5). С. 102–111.

34. Drechsler R. Formal Verification of Circuits. Kluwer Academic Publishers, 2000.

35. Advanced Formal Verification // Ed. by R. Drechsler. Kluwer Academic Publishers, 2004.

36. Kuehlmann A., Cornelis A. J., van Eijk. Combinational and Sequential Equivalence Checking // Logic synthesis and Verification, Ed. by S. Hassoun, T. Sasao, R. K. Brayton. Kluwer Academic Publishers, 2002. Pp. 343–372.

37. Robinson J. A. A Machine-Oriented Logic Based on the Resolution Principle // Journal of the ACM (JACM). 1965. Vol. 12, no. 1. Pp. 23–41.

38. Робинсон Дж. А. Машинно-ориентированная логика, основанная на принципе резолюций // Кибернетический сборник. Новая серия. 1970. Vol. 7. Pp. 194–217.

39. Riazanov A., Voronkov A. The Design and Implementation of Vampire // AI Communications. 2002. Vol. 15, no. 2-3. Pp. 91–110.

40. McCune W., Wos L. Otter: The CADE-13 Competition Incarnations // Journal of Automated Reasoning. 1997. Vol. 18, no. 2. Pp. 211–220.

41. Kalman J. A. Automated reasoning with Otter. Princeton, N. J.: Rinton Press, 2001.

42. Колмероэ А., Кануи А., ван Канегем М. Пролог — теоретические основы и современное развитие // Логическое программирование. 1988. С. 27–133.

43. Тей А., Грибомон П., Луи Ж. Логический подход к искусственному интеллекту. Москва: «Мир», 1991. С. 429.

44. Братко И. Алгоритмы искусственного интеллекта на языке PROLOG. «Вильямс», 2004. С. 640.

45. Gu J., Purdom P., Franco J., Wah B. W. Algorithms for the satisfiability (SAT) problem: A Survey // DIMACS Series in Discrete Mathematics and Theoretical Computer Science. 1997. Vol. 35, no. 4. Pp. 19–152.

46. Gu J. Local search for satisfiability (SAT) problem // IEEE Transactions on Systems, Man, and Cybernetics. 1993. Vol. 23, no. 4. Pp. 1108–1129.

47. Gu J. Global optimization for satisfiability (SAT) problem // IEEE Transactions on Knowledge and Data Engineering. 1994. Vol. 6, no. 3. Pp. 361–381.

48. Gu J. Optimization Algorithms for the Satisfiability (SAT) Problem // Advances in Optimization and Approximation, Ed. by D.-Z. Du; Kluwer Academic Publishers. 1994. Pp. 72–154.

49. Davis M., Logemann G., Loveland D. A machine program for theorem-proving // Communications of the ACM. 1962. Vol. 5, no. 7. Pp. 394–397.

50. Marques-Silva J. P., Sakallah K. A. GRASP: A search algorithm for propositional satisfiability // IEEE Transactions on Computers. 1999. Vol. 48, no. 5. Pp. 506–521.

51. Zhang L., Madigan C. F., Moskewicz M. H., Malik S. Efficient conflict driven learning in a boolean satisfiability solver // In Proceedings of International Conference on Computer-Aided Design. 2001. Pp. 279–285.

52. Goldberg E., Novikov Y. Berkmin: The Fast and Robust SAT Solver // Automation and Test in Europe (DATE). 2002. Pp. 142–149.

53. Een N., Sörensson N. An Extensible SAT-solver [extended version 1.2]. 2003.

54. Böhm M., Speckenmeyer E. A Fast Parallel SAT-Solver — Efficient Workload Balancing // Annals of Mathematics and Artificial Intelligence. 1996. Vol. 17, no. 3–4. Pp. 381–400.

55. Chrabakh W., Wolski R. GridSAT: A Chaff-based Distributed SAT Solver for the Grid // ACM/IEEE Supercomputing Conference. 2003.

56. Jurkowiak B., Li C.M., Utard G. Parallelizing Satz using Dynamic Workload Balancing // LICS 2001 Workshop on Theory and Applications of Satisfiability Testing. 2001. Pp. 205–211.

57. Zhang H., Bonacina M., Hsiang J. PSATO: a Distributed Propositional Prover and its Application to Quasigroup Problems // Journal of Symbolic Computation. 1996. Vol. 21, no. 4. Pp. 543–560.

58. C. Sinz W. Blochinger, Küchlin W. PaSAT — Parallel SAT-Checking with Lemma Exchange: Implementation and Applications // LICS 2001 Workshop on Theory and Applications of Satisfiability Testing. 2001.

59. Feldman Y., Dershowitz N., Hanna Z. Parallel Multithreaded Satisfiability Solver: Design and Implementation // Electronic Notes in Theoretical Computer Science. 2005. Vol. 128, no. 3. Pp. 75–90.

60. Schubert T., Lewis M., Becker B. PaMira – a Parallel SAT Solver with Knowledge Sharing // 6th International Workshop on Microprocessor Test and Verification. 2005. Pp. 29–34.

61. Gil L., Flores P., Silveira L. M. PMSat: a parallel version of MiniSAT // Journal on Satisfiability, Boolean Modeling and Computation. 2008. Vol. 6. Pp. 71–98.

62. Schubert T., Lewis M., Becker B. PaMiraXT: Parallel SAT Solving with Threads and Message Passing // Journal on Satisfiability, Boolean Modeling and Computation. Special Issue on Parallel SAT Solving. 2009. Vol. 6. Pp. 203–222.

63. Davis M., Putnam H. A Computing Procedure for Quantification Theory // Journal of the ACM (JACM). 1960. Vol. 7, no. 3. Pp. 201–215.

64. Haken A. The intractability of resolution // Theoretical Computer Science. 1985. Vol. 39. Pp. 297–308.

65. Хакен А. Труднорешаемость резолюций // Кибернетический сборник. Новая серия. 1991. Vol. 28. Pp. 179–194.

66. Ben-Sasson E., Impagliazzo R., Wigderson A. Near Optimal Separation on Tree-like and General Resolution // Combinatorica. 2004. Pp. 585–603.

67. Alekhnovich M., Johannsen J., Pitassi T., Urquhart A. An exponential separation between regular and general resolution // In 34th Annual ACM Symposium on Theory of Computing. 2002. Pp. 448–456.

68. Jeroslaw R. G., Wang J. Solving propositional satisfiability problems // Annals of Mathematics and Artificial Intelligence. 1990. Vol. 1. Pp. 167–187.

69. Moskewicz M. W., Madigan C. F., Zhao Y. et al. Chaff: engineering an efficient SAT solver // Proceedings of the 38th annual Design Automation Conference. 2001. Pp. 530–535.

70. Beame P., Kautz H., Sabharwal A. Understanding the power of clause learning // Proc. Of 18th Intern. Joint Conf. on Artificial Intelligence (IJCAI). 2003. Pp. 1194–1201.

71. Семенов А. А., Заикин О. С. Неполные алгоритмы в крупноблочном параллелизме комбинаторных задач // Вычислительные методы и программирование. 2008. Т. 9, № 1. С. 112–122.

72. Zhang H. SATO: An efficient propositional prover // Proceedings of International Conference on Automated deduction. 1997. Pp. 272–275.

73. Lynce I., Marques-Silva J. P. Efficient data structures for backtrack search SAT solvers // 5th International Symposium on the Theory and Applications of Satisfiability Testing (SAT'02). 2002.

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

75. Bryant R. E. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams // ACM Computing Surveys. 1992. Vol. 24, no. 3. Pp. 293–318.

76. Кларк Э. М., Грамберг О., Пелед Д. Верификация моделей программ: Model Checking. Москва: МЦНМО, 2002.

77. Shilov N. V., Yi K. How to find a coin: propositional program logics made easy // Current Trends in Theor. Comput. Sci., World Scientific. 2004. Vol. 2. Pp. 181–213.

78. Гаранина Н. О., Шилов Н. В. Верификация комбинированных логик знаний, действий и времени в моделях // Системная информатика. 2005. Vol. 10. Pp. 114–173.

79. Семичева Н. Л. Нахождение бесповторных представлений недоопределенных частичных булевых функций // серия «Дискретная математика и информатика». 2008. Т. 19. С. 35.

80. Зубков О. В. Оценки числа бесповторных булевых функций в бинарных базисах // серия «Дискретная математика и информатика». 2002. Т. 15. С. 34.

81. Игнатьев А. С., Семенов А. А., Хмельнов А. Е. Решение систем логических уравнений с использованием BDD // Вестник Томского гос. ун-та. Приложение. 2006. № 17. С. 25–29.

82. Игнатьев А. С., Семенов А. А., Хмельнов А. Е. Использование двоичных диаграмм решений в задачах обращения дискретных функций // Вестник Томского гос. ун-та. Серия: управление, вычислительная техника. 2009. № 1(6). С. 115–129.

83. Семенов А. А., Игнатьев А. С. Логические уравнения и двоичные диаграммы решений // Прикладные алгоритмы в дискретном анализе. Серия: Дискретный анализ и информатика. 2008. Т. 2. С. 99–126.

84. Andersen H. R. An Introduction to Binary Decision Diagrams. Lecture Notes. Copenhagen: Technical University of Denmark, 1997.

85. Valiant L. G. The complexity of computing the permanent // Theoretical Computer Science. 1979. Vol. 8. Pp. 189–202.

86. Valiant L. G. The complexity of enumeration and reliability problems // SIAM Journal on Computing. 1979. Vol. 8. Pp. 410–421.

87. Семенов А. А. Декомпозиционные представления логических уравнений в задачах обращения дискретных функций // Известия РАН. Теория и системы управления. 2009. № 5. С. 47–61.

88. Левитин А. Алгоритмы. Введение в разработку и анализ. «Вильямс», 2006. С. 576.

89. Колпаков А. В., Латыпов Р. Х. Приближенные алгоритмы минимизации двоичных диаграмм решений на основе линейных преобразований переменных // Автоматика и телемеханика. 2004. Т. 6. С. 112–128.

90. ван дер Варден Б. Л. Алгебра. Москва: «Мир», 1979. С. 649.

91. Игнатьев А. С., Семенов А. А. Алгоритмы работы с ROBDD как с базами булевых ограничений // Прикладная дискретная математика. 2010. № 1. С. 86–104.

92. Хмельнов А. Е., Игнатьев А. С., Семенов А. А. Двоичные диаграммы решений в логических уравнениях и задачах обращения дискретных функций // Вестник НГУ. Серия: Информационные технологии. 2009. Т. 7, № 4. С. 36–52.

93. Семенов А. А. Трансляция алгоритмов вычисления дискретных функций в выражения пропозициональной логики // Прикладные алгоритмы в дискретном анализе. Серия: Дискретный анализ и информатика. 2008. Т. 2. С. 70–98.

94. Zhang L., Madigan C., Moskewicz M., Malik S. Efficient conflict driven learning in a boolean satisfiability solver // Proc. Intern. Conf. on Computer Aided Design (ICCAD). 2001. Pp. 279–285.

95. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. Москва: МЦНМО: БИНОМ, 2004. С. 960.

96. Wikipedia. URL: `http://en.wikipedia.org/wiki/Thrash_(computer_science)`.

97. Fortune S. J. A sweepline algorithm for Voronoi diagrams // Algorithmica. 1987. Pp. 153–174.

98. Григоренко Е. Д., Евдокимов А. А., Лихошвай В. А., Лобарева И. А. Неподвижные точки и циклы автоматных отображений, моделирующих функционирование генных сетей // Вестник Томского гос. ун-та. Приложение. 2005. Т. 14. С. 206–212.

99. Евдокимов А. А., Лихошвай В. А., Комаров А. В. О восстановлении структуры дискретных моделей функционирования сетей // Вестник Томского гос. ун-та. Приложение. 2005. Т. 14. С. 213–217.

100. Евдокимов А. А., Кочемазов С. Е., Семенов А. А. Символьные алгоритмы в исследовании дискретных моделей некоторых классов генных сетей // Препринт. Издательство ИДСТУ СО РАН. 2010.

101. Игнатьев А. С., Семенов А. А., Беспалов Д. В. Двоичные диаграммы решений в параллельных алгоритмах обращения дискретных функций // Труды III Международной научной конференции ПАВТ'09. Нижний Новгород, ННГУ. 2009. С. 688–696.

102. Игнатьев А. С., Семенов А. А., Беспалов Д. В., Заикин О. С. Гибридный подход (SAT+ROBDD) в задачах криптоанализа поточных систем шифрования // Прикладная дискретная математика. Приложение. 2009. № 1. С. 19–20.

103. MiniSat. URL: http://minisat.se/MiniSat.html.

104. Biryukov A., Shamir A., Wagner D. Real Time Cryptanalysis of A5/1 on a PC // Fast Software Encryption Workshop. 2000. Pp. 1–18.

105. Menezes A., van Oorshot P., Vanstone S. Handbook of Applied Cryptography. CRC Press, 1996. P. 657.

106. Семенов А. А., Заикин О. С., Беспалов Д. В. и др. Решение задач обращения дискретных функций на многопроцессорных вычислительных системах // Труды Четвертой Международной конференции «Параллельные вычисления и задачи управления» (PACO'2008). Москва: 26–29 октября 2008. С. 152–176.

107. Заикин О. С., Семенов А. А. Технология крупноблочного параллелизма в SAT-задачах // Проблемы управления. 2008. Т. 1. С. 43–50.

108. Заикин О. С. Декомпозиционные представления данных в крупноблочном параллелизме SAT-задач // Прикладные алгоритмы в дискретном анализе. Серия: Дискретный анализ и информатика. 2008. Т. 2. С. 49–69.

109. JSAT Volume 6. URL: http://jsat.ewi.tudelft.nl/content/volume6-contents.html.

110. SAT-Race 2008. URL: http://baldur.iti.uka.de/sat-race-2008/.

111. Hamadi Y., Jabbour S., Sais L. ManySAT: a Parallel SAT Solver // Journal on Satisfiability, Boolean Modeling and Computation. Special Issue on Parallel SAT Solving. 2009. Vol. 6. Pp. 245–262.