

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

На правах рукописи

Отпущенников Илья Владимирович

**МЕТОДЫ И СРЕДСТВА
ПРЕОБРАЗОВАНИЯ ПРОЦЕДУРНЫХ
ОПИСАНИЙ ДИСКРЕТНЫХ ФУНКЦИЙ В
БУЛЕВЫ УРАВНЕНИЯ**

05.13.11 – Математическое и программное обеспечение вычислительных
машин, комплексов и компьютерных сетей

ДИССЕРТАЦИЯ

на соискание ученой степени
кандидата технических наук

Научный руководитель

к. т. н., доцент

Семёнов Александр Анатольевич

Иркутск – 2011

Содержание

Введение	4
Глава 1. Пропозициональное кодирование формальных вычислительных моделей	15
1.1. Базовые понятия и определения	15
1.2. Пропозициональное кодирование программ для формальных вычислительных моделей	17
1.2.1. Механизмы преобразования в булевы уравнения RAM-программ, вычисляющих дискретные функции . . .	20
1.2.2. Преобразования Цейтина	28
1.3. Основные алгоритмы решения систем булевых уравнений .	30
1.4. Необходимость разработки методов и средств преобразования процедурных описаний алгоритмов в булевы уравнения	32
Глава 2. Преобразование процедурных описаний дискретных функций в булевы уравнения	36
2.1. Описание проблемно-ориентированного языка ТА	37
2.2. Интерпретация ТА-программ. Формирование пропозиционального кода алгоритма	45
2.2.1. Базовые механизмы символьного исполнения ТА-программ	45
2.2.2. Символьное исполнение основных операторов языка ТА	53
2.2.3. Процедура декомпозиции сложных термов	58
2.2.4. Преобразование операций над целыми числами . . .	59
2.3. Программный комплекс Transalg для преобразования алгоритмов вычисления дискретных функций в булевы уравнения	65
Глава 3. Результаты преобразования в булевы уравнения алгоритмов вычисления некоторых дискретных функций .	70
3.1. Преобразование в булевы уравнения некоторых криптографических функций	70

3.1.1.	Преобразование в булевы уравнения алгоритма генератора ключевого потока шифра A5/1	71
3.1.2.	Преобразование в булевы уравнения алгоритма суммирующего генератора	74
3.1.3.	Преобразование в булевы уравнения алгоритма генератора Гиффорда	76
3.1.4.	Преобразование в булевы уравнения алгоритма DES	80
3.1.5.	Преобразование в булевы уравнения алгоритма хеширования MD5	87
3.2.	Исследование свойств дискретно-автоматных динамических систем	92
3.3.	Процедуры сведения оптимизационных задач с псевдобулевыми ограничениями к SAT-задачам	96
3.3.1.	Сведение к SAT задач 0-1-целочисленного линейного программирования	96
3.3.2.	Сведение к SAT квадратичных задач о назначениях (QAP)	101
Заключение		105
Литература		106
Приложение А. Грамматика языка ТА		120
Приложение Б. ТА-программа, вычисляющая 100 бит ключевого потока генератора A5/1		123
Приложение В. ТА-программа, реализующая шифрование 64 бит открытого текста алгоритмом DES		125
Приложение Г. ТА-программа, моделирующая функционирование генной сети		128

Введение

Актуальность темы исследования. Интенсивное развитие вычислительных технологий, наблюдающееся в последние годы, сделало возможным решение комбинаторных задач таких размерностей, которые раньше казались непреодолимыми. Следствием этого стало появление новых направлений в вычислительных разделах дискретной математики и математической логики. Как известно, подавляющее число практически значимых комбинаторных задач являются NP-трудными в общей постановке. Данное обстоятельство тем не менее не следует рассматривать как непреодолимое препятствие, поскольку во многих случаях тщательный анализ проблемы и правильный выбор алгоритмов и технологий позволяют найти решение за приемлемое время.

Одним из значимых достижений в исследовании комбинаторных проблем можно считать прогресс в решении систем логических (булевых) уравнений большой размерности. На сегодняшний день удается решать системы, содержащие сотни тысяч булевых переменных и уравнений (булевых ограничений). К булевым уравнениям эффективно сводятся многочисленные комбинаторные задачи. Процедура перехода от исходной постановки к системе булевых уравнений называется пропозициональным кодированием. Практически значимые задачи, которые имеет смысл сводить к булевым уравнениям, возникают в таких областях, как синтез и верификация схем в микроэлектронике, исследование безопасности коммуникационных протоколов, обоснование корректности программ, криптоанализ, а также при исследовании свойств динамических дискретно-автоматных моделей. Несмотря на то, что перечисленные проблемы важны в практическом отношении, работ, посвященных их пропозициональному кодированию, немного в сравнении с потоком статей по алгоритмам решения булевых уравнений.

Можно сослаться на результаты пропозиционального кодирования, представленные в статьях С. Прествича, Дж. Маркеса-Сильвы, Ф. Массаччи, Л. Марраро, И. Линс, И. Гента, Г. А. Опарина, В. Г. Богдановой, А. П. Новопашина, Дж. Гу, Н. Эйена, Н. Сорренсона и некоторых других авторов [1–12].

Многие проблемы из перечисленных выше областей естественным образом связаны с общей задачей обращения полиномиально вычислимых дискретных функций, когда зная алгоритм вычисления функции и некоторое слово из области ее значений, требуется найти неизвестный прообраз этого слова. Следует отметить, что в доступных работах по пропозициональному кодированию не содержится единого аппарата, который можно было бы применить к решению данной задачи. Проблема разработки и программной реализации такого аппарата представляется практически значимой, так как ее успешное решение даст возможность применять к широкому классу комбинаторных задач общую методологию, состоящую в сведении рассматриваемой задачи к булевым уравнениям и последующем использовании современных булевых решателей.

Для преобразования алгоритмов, вычисляющих дискретные функции, в булевы уравнения можно использовать различные подходы. Наиболее естественным представляется подход, при котором для описания алгоритмов вычисления функций используется некоторый процедурный язык. Построение системы булевых уравнений по тексту программы, написанной на таком языке, должно происходить автоматически в процессе ее интерпретации.

Механизмы преобразования программ в булевы уравнения должны предусматривать возможность выдавать пропозициональный код преобразуемого алгоритма в формах, допускающих эффективную (на практике) обработку посредством различных вычислительных методов. Наибольшей

эффективности поиска решений на сегодняшний день удастся добиться в отношении уравнений вида $\text{КНФ}=1$ (КНФ — конъюнктивная нормальная форма). Задачи поиска решений уравнений данного типа называются SAT-задачами, а специальные программные средства, предназначенные для их решения, называются SAT-решателями. Кроме « $\text{КНФ}=1$ » для итоговых представлений преобразуемых алгоритмов могут использоваться другие формы — все зависит от выбранного метода решения. Например, это могут быть уравнения вида $\text{ДНФ}=0$ (ДНФ — дизъюнктивная нормальная форма), системы алгебраических уравнений над $\text{GF}(2)$, «И-НЕ» графы (And-Inverter Graphs) и т.д.

Резюмируя все сказанное, отметим актуальность задачи разработки метода преобразования алгоритмических описаний дискретных функций в булевы уравнения и реализацию такого метода в форме программного комплекса. Описания дискретных функций должны выполняться на некотором процедурном языке, а механизмы преобразования получаемых программ в булевы уравнения должны учитывать различные особенности исходных постановок с целью ускорения последующего процесса поиска решений. Для использования различных вычислительных методов должна быть предусмотрена возможность вывода результирующего описания алгоритма в различных форматах.

Цель и задачи исследования. Целью диссертации является разработка общего подхода к проблеме преобразования алгоритмов вычисления всюду определенных дискретных функций в булевы уравнения, а также создание на этой основе инструментального средства, предназначенного для исследования широкого класса функций, возникающих в приложениях.

Для достижения указанной цели ставятся и решаются перечисленные ниже задачи.

1. Разработать общую методологию преобразования высокоуровневых описаний алгоритмов вычисления дискретных функций в булевы уравнения; сформулировать общие принципы и описать базовые механизмы таких преобразований.
2. Разработать проблемно-ориентированный язык процедурного типа, предназначенный для описания алгоритмов вычисления всюду определенных дискретных функций и обладающий семантикой, обеспечивающей корректное построение булевых уравнений.
3. Разработать и реализовать структуры данных и алгоритмы, позволяющие автоматизировать преобразования в булевы уравнения процедур вычисления дискретных функций, описанных на специально созданном языке.
4. Создать программный комплекс, интегрирующий все разработанные алгоритмы и структуры данных, который осуществляет преобразование программ вычисления дискретных функций, оптимизирует получаемые булевы структуры, а также выполняет проверку их корректности.
5. С использованием этого программного комплекса создать набор шаблонов, хранящих пропозициональные коды ряда функциональных примитивов, широко используемых при описании поведения дискретных систем.

Методы и инструменты исследования. Теоретическая часть исследований базируется на теории множеств, методах дискретной математики, теории булевых функций, теории вычислительной сложности, криптографии, теории процедурных языков программирования. Экспериментальная часть использует современные средства разработки программного обес-

печения (язык программирования C++, среда разработки и сборки приложений Visual Studio 2008 Express Edition).

Научная новизна. В диссертации предложен новый подход к преобразованию процедурных описаний дискретных функций в булевы уравнения. Для описания алгоритмов вычисления дискретных функций разработан новый проблемно-ориентированный язык ТА, обладающий C-подобным синтаксисом и оригинальной семантикой, которая обеспечивает построение систем булевых уравнений в процессе интерпретации ТА-программ. Для преобразования ТА-программ в булевы уравнения разработаны новые алгоритмы и структуры данных. Все разработанные алгоритмы интегрированы в программный комплекс Transalg, с применением которого были построены новые пропозициональные коды ряда криптографических преобразований и оптимизационных задач с псевдобулевыми ограничениями.

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

Теоретическая и практическая значимость работы. Теоретическая значимость работы заключается в сформулированной общей методологии преобразования процедурных описаний дискретных функций в булевы уравнения, в принципиальной возможности применения разработанных методов к исследованию широкого класса дискретных функций. Практическая значимость состоит в завершенности представленного в работе подхода и в возможности его применения к исследованию различных дискретных систем, поведение которых описывается полиномиально вычислимыми дискретными функциями. В частности, при помощи комплекса Transalg были построены пропозициональные коды ряда криптографических алгоритмов,

а также некоторых задач дискретной оптимизации.

Соответствие специальности. Работа относится к области анализа алгоритмов и является исследованием, цель которого состоит в обосновании и развитии методологии преобразования процедурных описаний дискретных функций в булевы уравнения. В диссертации представлен метод для осуществления таких преобразований, а также разработан проблемно-ориентированный язык программирования (язык ТА), предназначенный для описания алгоритмов вычисления дискретных функций. Семантика языка ТА обеспечивает эффективное построение булевых уравнений в процессе интерпретации ТА-программ. В диссертации разработаны и реализованы в виде программного комплекса (комплекс Transalg) алгоритмы, осуществляющие преобразование процедурных описаний, выполненных на языке ТА, в булевы уравнения. При помощи комплекса Transalg построены пропозициональные коды ряда криптографических алгоритмов, а также некоторых задач дискретной оптимизации. Таким образом, материал диссертации соответствует как формуле специальности 05.13.11, так и пунктам 1, 2 из «Области исследований».

Содержательная часть работы включает введение и три главы.

Первая глава посвящена основам методологии преобразования в булевы уравнения процедур вычисления дискретных функций в контексте формальной вычислительной модели — двоичной машины с произвольным доступом к памяти (RAM). Описываются основные механизмы построения булевых уравнений по программе для двоичной RAM.

В разделе 1.1 вводятся основные понятия и определения, дается общая формулировка задачи обращения дискретных функций.

В разделе 1.2 приведена основная теорема об эффективной сводимости проблемы обращения полиномиально вычислимых дискретных функций к задачам поиска решений систем булевых уравнений. Здесь же опи-

саны преобразования Цейтина — основной инструмент перехода от произвольных систем булевых уравнений к уравнениям в нормальных формах.

В разделе 1.3 проводится краткий обзор основных методов решения булевых уравнений.

В разделе 1.4 обосновывается актуальность основной цели диссертационной работы, которая заключается в разработке методов и программных средств преобразования алгоритмических описаний дискретных функций в булевы уравнения.

Вторая глава содержит теоретическую основу механизмов преобразования процедурных описаний дискретных функций в булевы уравнения. В этой главе описаны все базовые алгоритмы и структуры данных, используемые в процессе таких преобразований. Здесь же представлена архитектура программного комплекса Transalg и приведено описание основных его функциональных возможностей.

В разделе 2.1 представлен проблемно-ориентированный язык (язык ТА), предназначенный для описания алгоритмов, вычисляющих дискретные функции, с целью их преобразования в булевы уравнения. Язык ТА является процедурным языком с блочной структурой и С-подобным синтаксисом. Каждый блок — это список инструкций ТА-программы. Программа на языке ТА представляет собой набор определений функций, а также объявлений и определений глобальных переменных и констант. В языке ТА реализованы все основные примитивные конструкции, характерные для процедурных языков программирования.

Раздел 2.2 посвящен описанию основных механизмов интерпретации ТА-программ (семантике языка ТА). В соответствии с принятым подходом [13] интерпретационная семантика языка ТА описывается через ввод в рассмотрение абстрактной машины языка ТА (ТА-машины). Произвольная конфигурация ТА-машины фиксирует момент интерпретации некоторой

инструкции ТА-программы. При этом в памяти ТА-машины отображается информация о связи переменных, фигурирующих в тексте ТА-программы, с переменными, фигурирующими в системе булевых уравнений, и с логическими выражениями (термами). Здесь же описывается техника сокращения избыточности пропозиционального кода алгоритма, использующая специальный словарь термов. В этом же разделе подробно описаны механизмы символьной интерпретации основных операторов языка ТА. В заключительной части раздела рассмотрены преобразования в булевы уравнения операций над целыми числами.

В разделе 2.3 приведено описание программного комплекса Transalg, в котором были реализованы все алгоритмы и структуры данных, представленные во второй главе. Основное назначение комплекса Transalg состоит в преобразовании описаний алгоритмов вычисления дискретных функций на языке ТА в булевы уравнения. В этом же разделе описаны различные форматы выходных данных, генерируемых комплексом Transalg по ТА-программам.

В третьей главе диссертации представлены результаты применения программного комплекса Transalg к решению разнообразных комбинаторных задач.

В разделе 3.1 приведены результаты преобразования в булевы уравнения (и в конечном счете в SAT-задачи) ряда криптографических функций. Рассмотрены генераторы ключевого потока Рюппеля, Гиффорда, генератор, используемый в шифре A5/1, блочный шифр DES и алгоритм хеширования MD5.

В разделе 3.2 комплекс Transalg используется для построения SAT-задач, кодирующих проблемы поиска циклов автоматных отображений, описывающих поведение дискретных динамических моделей генных сетей [14, 15]. Получаемые SAT-задачи решались известным SAT-решателем Mini-

Sat 2.0 [16]. Для всех тестов либо были найдены циклы (длины до 10), либо было установлено, что таких циклов нет.

В разделе 3.3 приведено описание дополнительных модулей комплекса Transalg, предназначенных для сведения задач с псевдобулевыми ограничениями к SAT-задачам. При помощи программного комплекса Transalg к SAT-задачам были сведены задачи из семейства 0-1-ЦЛП (целочисленное линейное программирование) и задачи из семейства QAP (квадратичная задача о назначениях).

В Приложении 1 описана грамматика языка ТА в нотации Бэкуса-Наура. Приложение 2 содержит ТА-программу, вычисляющую 100 бит ключевого потока генератора A5/1. В Приложение 3 приводится текст ТА-программы, описывающей алгоритм вычисления криптографической функции DES. Приложение 4 содержит пример ТА-программы, моделирующей функционирование генной сети.

Апробация работы. Результаты диссертации докладывались и обсуждались на 5-й Международной научной конференции «Параллельные вычислительные технологии» (Москва, 2011 г.), на IX Всероссийской школе-семинаре с международным участием Sibecrypt-10 (Тюмень, 2010 г.), на XIV Байкальской международной школе-семинаре «Методы оптимизации и их приложения» (Северобайкальск, 2008 г.), на IX и X Всероссийских конференциях молодых ученых «Математическое моделирование и информационные технологии» (Иркутск, 2007 г., 2009 г.), на ежегодных конференциях «Ляпуновские чтения и презентация информационных технологий» (Иркутск, 2007–2010 гг.), а также на семинарах Института динамики систем и теории управления СО РАН, Института цитологии и генетики СО РАН, Института математики им. С. Л. Соболева СО РАН, Института систем информатики им. А. П. Ершова СО РАН.

Результаты диссертации были получены в процессе исследований по

следующим проектам:

- проект СО РАН «Интеллектуальные методы и инструментальные средства создания и анализа интегрированных распределенных информационно-аналитических и вычислительных систем для междисциплинарных исследований с применением ГИС, GRID и Веб-технологий» (№ гос. регистрации: 01.2.00708582) 2007–2009 гг.;
- проекта СО РАН «Интеллектуальные методы автоматизации решения задач в параллельных и распределенных вычислительных средах» (№ гос. регистрации: 01201001348), 2010–2011 гг.;
- грант РФФИ № 07-01-00400-а «Характеризация сложности обращения дискретных функций в задачах криптографии и интервального анализа»;
- грант РФФИ № 11-07-00377-а «Разработка параллельных алгоритмов решения булевых уравнений и их реализация в GRID-системах»;
- грант Президента РФ НШ-1676.2008.1.

Публикации и личный вклад автора. По теме диссертации опубликовано 12 работ. Наиболее значимые результаты представлены в [17–25]. В число указанных работ входят 4 статьи из Перечня ведущих рецензируемых журналов и изданий ВАК РФ (от 25.02.2011 г.), 1 статья в тематическом сборнике, 2 полных текста докладов в материалах международных конференций, а также одно свидетельство о регистрации программы для ЭВМ.

Все результаты, выносимые на защиту, получены автором лично. В основных публикациях по теме диссертации научному руководителю принадлежат постановки задач и некоторые теоретические результаты. В работах

[18], [23] и [24] Заикину О. С. принадлежат результаты по распараллеливанию схем решения SAT-задач. Все результаты по разработке и реализации алгоритмов сведения комбинаторных задач к булевым уравнениям принадлежат автору.

Структура работы. Диссертация состоит из введения, трех глав, заключения, списка литературы из 121 наименования и четырех приложений. Объем диссертации — 128 страниц. Диссертация содержит 19 рисунков и 7 таблиц.

Глава 1

Пропозициональное кодирование формальных вычислительных моделей

1.1. Базовые понятия и определения

В данном разделе приведены базовые определения и понятия, относящиеся к предмету исследования (см. [26–33]). Булевы и дискретные функции являются важнейшими объектами математической логики и дискретной математики. Булевыми переменными называются переменные, принимающие значения из множества {«ложь», «истина»}. Далее «ложь» обозначается символом 0, а «истина» — символом 1.

Через $\{0, 1\}^n$, $n \in \mathbb{N}$, обозначается множество всех двоичных слов длины n . Также рассматривается множество всевозможных двоичных слов произвольной конечной длины:

$$\{0, 1\}^* = \bigcup_{n \in \mathbb{N}} \{0, 1\}^n.$$

Далее под дискретными функциями понимаются любые функции вида $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$. Дискретные функции вида $f_n : \{0, 1\}^n \rightarrow \{0, 1\}^*$, $n \in \mathbb{N}$, называются дискретными функциями от n переменных. Функции вида $f_n : \{0, 1\}^n \rightarrow \{0, 1\}$ называются булевыми.

Область определения и область значения дискретной функции f будем обозначать соответственно через $\text{dom } f$ и $\text{range } f$.

Дискретные функции $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ и $f_n : \{0, 1\}^n \rightarrow \{0, 1\}^*$, $n \in \mathbb{N}$, называются всюду определенными (тотальными), если $\text{dom } f = \{0, 1\}^*$, $\text{dom } f_n = \{0, 1\}^n$.

Пусть $X = \{x_1, \dots, x_n\}$ — множество булевых переменных и пусть

$L(x_1, \dots, x_n)$ — произвольная формула исчисления высказываний [27]. Данная формула естественным образом задает тотальную булеву функцию от булевых переменных $x_1, \dots, x_n$. Тот факт, что при подстановке

$$x_1 = \alpha_1, \dots, x_n = \alpha_n, \alpha_i \in \{0, 1\}, i \in \{1, \dots, n\},$$

$L(x_1, \dots, x_n)$ принимает значение $\beta \in \{0, 1\}$ обозначаем через $L|_{(\alpha_1, \dots, \alpha_n)} = \beta$.

Выражения вида $L(x_1, \dots, x_n) = 0$, $L(x_1, \dots, x_n) = 1$ называются булевыми уравнениями (в некоторых источниках, например в [34], используется термин «логические уравнения»). Набор $(\alpha_1, \dots, \alpha_n)$, $\alpha_i \in \{0, 1\}$, $i \in \{1, \dots, n\}$, такой, что $L(x_1, \dots, x_n)|_{(\alpha_1, \dots, \alpha_n)} = \beta$, $\beta \in \{0, 1\}$, называется решением уравнения $L(x_1, \dots, x_n) = \beta$. Если такого набора не существует, то говорят, что данное уравнение не имеет решений.

Произвольный конечный набор булевых уравнений называется системой булевых уравнений (СБУ). Произвольный набор значений истинности булевых переменных, входящих в систему, являющийся решением всех ее уравнений, называется решением данной системы. Если такого набора не существует, то СБУ называется несовместной.

Далее в качестве базовой формальной вычислительной модели рассматривается детерминированная машина Тьюринга (ДМТ) с входным алфавитом $\Sigma = \{0, 1\}$ (см. [29]).

Произвольная ДМТ-программа M_f , останавливающаяся на произвольном входном слове из $\{0, 1\}^*$, естественным образом задает тотальную дискретную функцию $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$.

Очевидно, что данная функция может рассматриваться как счетное семейство дискретных функций вида $f_n : \{0, 1\}^n \rightarrow \{0, 1\}^*$, $n \in \mathbb{N}$. Стандартным образом (см. [29]) можно определить временную сложность программы M_f , вычисляющей функции данного семейства, как функцию от

длины входа n . Если сложность программы M_f ограничена полиномом от n , то функция f называется полиномиально вычислимой.

Поведение дискретных систем из широкого класса может быть описано полиномиально вычислимыми функциями. В качестве примеров можно привести различные управляющие/управляемые системы, в том числе на основе СБИС, протоколы обмена информацией в системах связи, шифрующие преобразования и др. Криптографические функции в данном контексте особенно важны как источники аргументированно сложных в вычислительном отношении тестов.

В последние годы в связи с интенсивным развитием таких областей как Hardware/Software model checking [35–38], комбинаторная оптимизация [39–41], криптография/криптоанализ возрастает актуальность проблем разработки как методов решения булевых уравнений, так и методов преобразования комбинаторных задач в булевы уравнения.

1.2. Пропозициональное кодирование программ для формальных вычислительных моделей

Базовые положения рассматриваемой далее техники пропозиционального кодирования программ для формальных вычислительных моделей впервые были предложены С. А. Куком в работе [42]. Следует отметить, что изначально теорема Кука воспринималась как результат «деструктивного» характера, поскольку для определяемого с ее помощью класса задач скорее всего не существует полиномиальных алгоритмов. Однако наблюдающийся рост мощности вычислительной техники позволяет не воспринимать подобную аргументацию трудности задачи как непреодолимое препятствие. Действительно, во многих практически важных частных случаях даже точные решения NP-трудных задач могут быть найдены за приемлемое время при

помощи эвристик, базирующихся на «разумных аргументах». Сказанное означает актуальность проблем разработки как эффективных на практике вычислительных алгоритмов решения систем булевых уравнений большой размерности, так и процедур сведения различных комбинаторных задач к булевым уравнениям.

Для разработки практически применимой технологии преобразования алгоритмов, вычисляющих дискретные функции, в булевы уравнения необходимо построение прототипа данной технологии на уровне формальной вычислительной модели. Выбираемая при этом формальная модель должна обеспечивать относительно простой переход от разработанного прототипа к процедурам преобразования в булевы уравнения алгоритмов, записанных на процедурных языках программирования. Кроме этого, архитектура такой модели должна быть максимально близка к архитектурам современных ЭВМ. Всем перечисленным требованиям удовлетворяет машина с произвольным доступом (RAM) (см. [43]). Далее используется вариант RAM, синтаксис программ которой фактически совпадает с синтаксисом программ для машины с неограниченными регистрами (МНР) в формализме Н. Катленда [28]. Он близок к синтаксису современных ассемблерных языков, что делает данный вариант RAM перспективной основой для разработки теоретической базы технологии преобразования алгоритмов вычисления дискретных функций в булевы уравнения.

Итак, далее рассматривается двоичная (бинарная) RAM в формализме Н. Катленда. Ее рабочая область состоит из бесконечной вправо ленты, разделенной на ячейки, которые пронумерованы натуральными числами. В каждой ячейке может быть записан только один бит. Произвольная RAM-программа — это нумерованный список команд, каждая из которых может быть командой одного из следующих двух типов:

1. команды записи в ячейку с номером k бита «0» или бита «1» — соответственно $B_0(k)$ и $B_1(k)$;
2. команды условного перехода $J(k, l, m)$ — сравнить содержимое ячеек с номерами k и l , в случае совпадения перейти к команде с номером m в списке, в противном случае перейти к команде, которая следует в списке за командой $J(k, l, m)$.

Вычисление останавливается либо после выполнения последней команды в программе (если данная команда не является командой условного перехода), либо если происходит ссылка на несуществующую команду.

Пример 1.1 (см. [29]) Программа проверки делимости на 4 произвольного натурального числа, двоичное представление которого имеет длину n , может быть записана в виде:

1. $J(1, n + 1, 3)$
2. $J(1, 1, 100)$
3. $J(1, n, 5)$
4. $J(1, 1, 100)$
5. $B_1(1)$

1	2	3		n+1
0	x_1	x_2	...	x_n

Рис. 1.1. Рабочая область RAM-машины.

В первой ячейке записывается бит ответа: «0» означает, что число не делится на 4, «1» — число делится на 4. В начальной конфигурации RAM в первой ячейке записан «0», а в следующих n ячейках записывается двоичное представление натурального числа. Программа проверяет два младших бита числа на равенство нулю, если это выполняется, то в первую ячейку записывается «1».

1.2.1. Механизмы преобразования в булевы уравнения

RAM-программ, вычисляющих дискретные функции

Итак, далее рассматривается тотальная функция

$$f : \{0, 1\}^* \rightarrow \{0, 1\}^*,$$

вычислимая программой M_f для машины Тьюринга с входным алфавитом $\Sigma = \{0, 1\}$ за полиномиальное время. Как отмечалось выше, программа M_f задает счетное семейство дискретных функций

$$f_n : \{0, 1\}^n \rightarrow \{0, 1\}^*, \text{ dom } f_n = \{0, 1\}^n, n \in \mathbb{N}.$$

Проблемой обращения произвольной функции f_n из данного семейства в точке $y \in \text{range } f_n$ называется следующая задача: зная M_f , число n и $y \in \text{range } f_n$, найти такое $x \in \{0, 1\}^n$, что $f_n(x) = y$.

Пропозициональный подход к данной задаче основан на следующем факте: процесс работы программы M_f на произвольном входе можно эффективно (в общем случае за полиномиальное от n время) представить в виде формулы исчисления высказываний. Истинность данной формулы при некоторых дополнительных условиях, характеризующих конкретный выход $y \in \text{range } f_n$, означает существование такого входа $x \in \{0, 1\}^n$, результатом трансформации которого посредством программы M_f является y . Фактически в этом состоит теорема Кука [42].

Мы приводим два утверждения из работы [44], являющиеся теоретической базой описываемых далее процедур преобразования алгоритмов.

Лемма 1.1 (о моделировании) Пусть M_f — ДМТ-программа, вычисляющая тотальную дискретную функцию $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$, и функция сложности программы M_f ограничена некоторым полиномом от n . Существует тотальная алгоритмически вычислимая функция g , которая за полиномиальное от n время по тексту программы M_f и числу n выдает текст

программы R_{f_n} , вычисляющей функцию f_n . Функция сложности $\vartheta(n)$, сопоставляемая получаемому семейству RAM-программ, ограничена сверху полиномом от n .

Данный факт означает наличие эффективной процедуры перехода от ДМТ-программы, вычисляющей f , к семейству RAM-программ, каждая из которых вычисляет функцию f_n , $n \in \mathbb{N}$. Как уже отмечалось, синтаксис RAM-программ близок к ассемблерным программам, что крайне важно при построении практических процедур пропозиционального кодирования алгоритмов. В техническом плане доказательство леммы 1.1 стандартно — работа ДМТ с входным алфавитом $\{0, 1\}$ на всевозможных входах длины n моделируется на описанной выше RAM. Данная лемма используется при доказательстве следующей теоремы.

Теорема 1.1. Пусть $f = \{f_n\}_{n \in \mathbb{N}}$ — семейство алгоритмически вычислимых за полиномиальное время дискретных функций и пусть $\{R_{f_n}\}_{n \in \mathbb{N}}$ — семейство RAM-программ, сопоставляемое f в соответствии с леммой о моделировании. Существует алгоритмически вычислимая тотальная функция h , которая, получая на входе текст программы R_{f_n} , за полиномиальное в общем случае от n время строит такую систему булевых уравнений $S(f_n)$, что:

1. для произвольного $y \in \text{range } f_n$ система $S(f_n)|_y$ совместна;
2. если $\hat{x}$ — решение системы $S(f_n)|_y$, то за линейное от $|\hat{x}|$ время возможен переход от $\hat{x}$ к $x \in \{0, 1\}^n : f_n(x) = y$.

Доказательство. Полное доказательство данного факта приведено в [44]. Здесь мы приводим несколько иное доказательство, дающее основу для последующего построения программного комплекса, осуществляющего

преобразования процедурных описаний алгоритмов вычисления дискретных функций в булевы уравнения.

Базовая идея пропозиционального кодирования алгоритмов состоит в построении символьного описания всех возможных эволюций соответствующего вычисления на произвольных входах. В нашем случае требуется описать поведение RAM-вычисления на всевозможных входных двоичных словах длины n . Это можно сделать, если описывать каждую последующую конфигурацию набором функций от предыдущей конфигурации. Если к некоторой конфигурации K^i применяется оператор условного перехода, то, вообще говоря, возможен переход как в K_+^{i+1} , так и в K_-^{i+1} , однако можно рассмотреть «абстрактную» конфигурацию K^{i+1} , ячейкам которой сопоставлены булевы переменные, принимающие (в зависимости от соответствующего условия) либо значения, которые формируют конфигурацию K_+^{i+1} , либо значения, которые формируют конфигурацию K_-^{i+1} . Тем самым, биты в ячейках памяти RAM в конфигурации K^{i+1} можно считать значениями булевых функций от содержимого памяти RAM в конфигурации K^i . Данные функции можно записать в явном виде, используя только текст программы R_{f_n} . Очень важно то, что при этом мы точно знаем, сколько потребуется булевых переменных, кодирующих содержимое памяти RAM в конфигурации K^{i+1} . Это число заведомо не больше максимального номера ячейки, к которой обращается программа R_{f_n} в процессе вычисления.

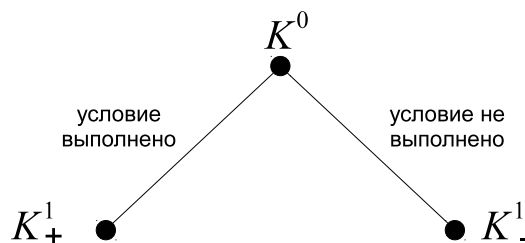


Рис. 1.2. Условный переход в RAM-программе.

Итак, процесс пропозиционального кодирования программы R_{f_n} состо-

ит в следующем. Предполагается, что R_{f_n} получает на вход произвольный двоичный вектор, кодируемый булевыми переменными $x_1, \dots, x_n$ (стартовая конфигурация K^0). Предположим, что первой командой в R_{f_n} является команда условного перехода $J(q, r, s)$. Символически переход из K^0 в K_+^1 или в K_-^1 представлен на рисунке 1.2.

Для описания конфигурации K^1 вводятся переменные $x_1^1, \dots, x_{\rho(n)}^1$, где $\rho(n)$ — максимальное натуральное число, фигурирующее в тексте программы R_{f_n} (рост функции $\rho(n)$ ограничен некоторым полиномом от n). Каждой переменной x_j^1 , $j \in \{1, \dots, \rho(n)\}$, ставим в соответствие формулу

$$x_j^1 \equiv \theta_j^1(x_1, \dots, x_n),$$

где $\theta_j^1(x_1, \dots, x_n)$ — формула исчисления высказывания, задающая некоторую булеву функцию от переменных $x_1, \dots, x_n$. Далее действуем по аналогии: содержимое конфигурации K^2 кодируется булевыми переменными x_j^2 , $j \in \{1, \dots, \rho(n)\}$, связанными с формулами

$$x_j^2 \equiv \theta_j^2(x_1^1, \dots, x_{\rho(n)}^1).$$

И так далее. Пусть

$$X^0 = \{x_1, \dots, x_n\}, X^1 = \{x_1^1, \dots, x_{\rho(n)}^1\}, \dots, X^e = \{x_1^e, \dots, x_{\rho(n)}^e\}$$

— множества переменных, соответствующие в указанном смысле конфигурациям $K^0, K^1, \dots, K^e$ (K^e — заключительная конфигурация). Описанный процесс изображен на рисунке 1.3.

Данному процессу сопоставляется следующая система булевых уравнений, обозначаемая через $S(f_n)$:

$$\left\{ \begin{array}{l} \bigwedge_{j=1}^{\rho(n)} (x_j^1 \equiv \theta_j^1(x_1, \dots, x_n)) \\ \dots \\ \bigwedge_{j=1}^{\rho(n)} (x_j^e \equiv \theta_j^e(x_1^{e-1}, \dots, x_{\rho(n)}^{e-1})) \end{array} \right. = 1 \quad (1.1)$$

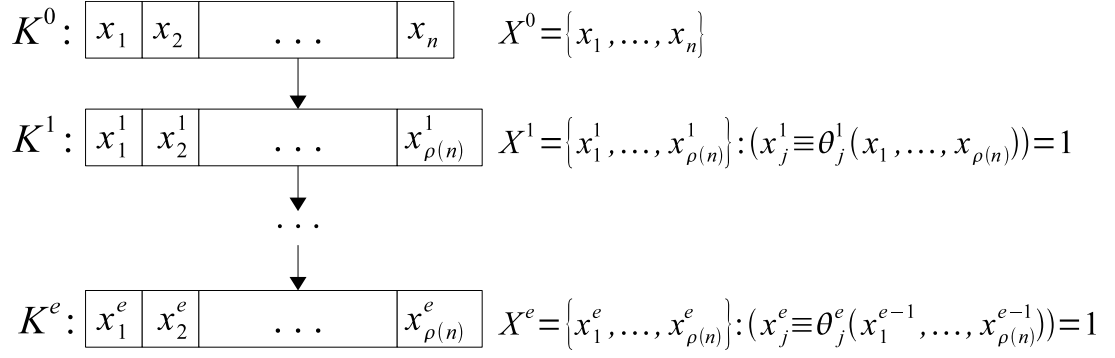


Рис. 1.3. Последовательность конфигураций RAM, вычисляющей дискретную функцию от n переменных.

В [44] показано, что система (1.1) обладает следующими свойствами:

1. размерность системы $S(f_n)$ ограничена полиномом от n ;
2. подстановка в $S(f_n)$ произвольного вектора $y \in \{0, 1\}^*$ дает несовместную систему, если $y \notin \text{range } f_n$. Если же $y \in \text{range } f_n$, то получаем систему $S(f_n)|_y$ относительно переменных из $X^0 \cup \dots \cup X^e$, имеющую решения;
3. пусть u — некоторое решение системы $S(f_n)|_y$, тогда из построения данной системы следует, что вектор длины n , образованный компонентами в u , которые соответствуют переменным из множества X^0 , дает прообраз y в $\{0, 1\}^n$ при отображении f_n .

Все сказанное означает справедливость теоремы 1.1. ■

Следующие примеры иллюстрируют основные принципы пропозиционального кодирования алгоритмов, описанные в доказательстве теоремы 1.1.

Пример 1.2. Рассматривается регистр сдвига с линейной обратной связью (далее РСЛОС) (см. [45], [46], [47]). РСЛОС длины n , задаваемый полиномом обратной связи $P(X) = X^n + c_{n-1}X^{n-1} + \dots + c_1X^1 + c_0$ над полем $\text{GF}(2)$, обозначается через $\langle n, P(X) \rangle$ (X — формальная переменная).

Выходная последовательность длины m данного генератора — это последовательность битов, являющаяся результатом первых m тактов работы РСЛОС. Сами по себе РСЛОС не являются криптографически стойкими, однако они используются в качестве примитивов в более сложных шифрах. Далее предполагается, что выходом регистра сдвига на каждом такте является крайний правый бит регистра.

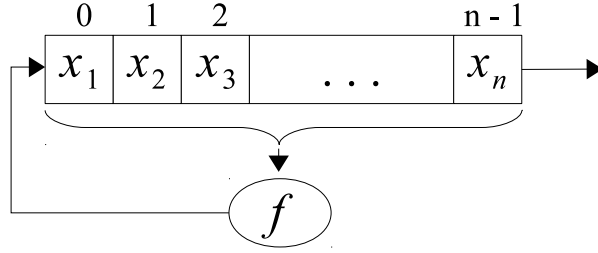


Рис. 1.4. Схема n -битного РСЛОС с функцией обратной связи f .

Рассмотрим n -битный РСЛОС как фрагмент памяти RAM. Полагаем, что множество булевых переменных $X^0 = \{x_1^0, \dots, x_n^0\}$ кодирует начальное заполнение регистра, которое соответствует начальной конфигурации RAM-машины. Множества булевых переменных $X^1, \dots, X^e$, кодирующие соответствующие конфигурации RAM, строятся по формулам

$$X^i = \{x_1^i \equiv f(x_1^{i-1}, \dots, x_n^{i-1}), x_2^i \equiv x_1^{i-1}, \dots, x_{n+i}^i \equiv x_{n+i-1}^{i-1}\}, i \in \{1, \dots, e\},$$

где $f(\cdot)$ — функция обратной связи (см. [45]). Множество булевых переменных $\{x_{n+1}^e, \dots, x_{n+e}^e\} \subset X^e$ кодирует выходную последовательность регистра сдвига. Пусть биты выходной последовательности известны, тогда, подставив их в систему булевых уравнений,

$$\left\{ \begin{array}{ll} (x_1^i \equiv f(x_1^{i-1}, \dots, x_n^{i-1})) & = 1 \\ (x_2^i \equiv x_1^{i-1}) & = 1 \\ \dots & \\ (x_{n+i}^i \equiv x_{n+i-1}^{i-1}) & = 1 \end{array} \right. , i \in \{1, \dots, e\} \quad (1.2)$$

получим совместную систему. Из решения данной системы определяем начальное заполнение $X^0 = \{x_1^0, \dots, x_n^0\}$ регистра сдвига.

Пример 1.3. В этом примере строится система булевых уравнений, описывающая процесс порождения ключевого потока в известном генераторе поточного шифрования A5/1. Общая схема работы данного генератора выглядит следующим образом (см. [46, 48]).

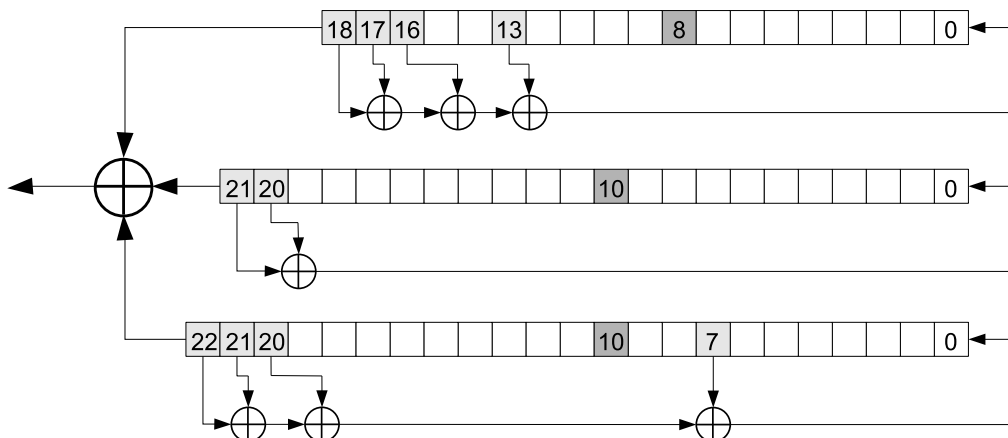


Рис. 1.5. Схема генератора ключевого потока A5/1.

В генераторе A5/1 используются три регистра с линейной обратной связью, задаваемые следующими полиномами обратной связи над $\text{GF}(2)$: $X^{19} + X^{18} + X^{17} + X^{14} + 1$ (длина регистра 19 бит); $X^{22} + X^{21} + 1$ (длина регистра 22 бита); $X^{23} + X^{22} + X^{21} + X^8 + 1$ (длина регистра 23 бита). Ячейки регистров нумеруются с нуля и справа налево.

Особенностью данного генератора является то, что на каждом такте могут сдвигаться не все регистры. Сдвиг регистра с номером m , $m \in \{1, 2, 3\}$, происходит, если значение функции $\chi_m(b_*^1, b_*^2, b_*^3)$ равно 1, и не происходит, если значение данной функции равно 0. Функция $\chi_m(\cdot)$ определяется следующим образом:

$$\chi_m(b_*^1, b_*^2, b_*^3) = \begin{cases} 1, & b_*^m = \text{majority}(b_*^1, b_*^2, b_*^3) \\ 0, & b_*^m \neq \text{majority}(b_*^1, b_*^2, b_*^3) \end{cases},$$

где $\text{majority}(A, B, C) = A \cdot B \vee B \cdot C \vee A \cdot C$ (функция большинства), b_*^1, b_*^2, b_*^3 — так называемые срединные биты, то есть биты, находящиеся в девятой ячейке первого РСЛОС, и в ячейках с номером 11 во втором и третьем РСЛОС (на рисунке данные ячейки закрашены в темно-серый цвет).

В каждом такте с регистров снимаются старшие биты, складываются по mod 2, результирующий бит является битом ключевого потока данного такта. Задача криптоанализа состоит в нахождении начальных заполнений регистров по некоторой известной последовательности битов ключевого потока и по известному алгоритму их порождения. Сведем данную задачу к задаче поиска решений системы булевых уравнений. Начальное заполнение регистров кодируем множеством булевых переменных $X^0 = \{x_1^0, \dots, x_{64}^0\}$.

Далее приведена система булевых уравнений, описывающая e шагов работы первого регистра в А5/1. Сдвиг регистра выполняется в зависимости от текущего значения битов b_*^1, b_*^2 и b_*^3 , и определяется булевой функцией $\chi_1(b_*^1, b_*^2, b_*^3)$. Система (1.2) принимает вид:

$$\left\{ \begin{array}{l} (x_1^i \equiv \chi_1(b_*^1, b_*^2, b_*^3) \cdot f(x_1^{i-1}, \dots, x_{19}^{i-1}) \vee \bar{\chi}_1(b_*^1, b_*^2, b_*^3) \cdot x_1^{i-1}) = 1 \\ (x_2^i \equiv \chi_1(b_*^1, b_*^2, b_*^3) \cdot x_1^{i-1} \vee \bar{\chi}_1(b_*^1, b_*^2, b_*^3) \cdot x_2^{i-1}) = 1 \\ \dots \\ (x_{19+i}^i \equiv \chi_1(b_*^1, b_*^2, b_*^3) \cdot x_{19+i-1}^{i-1} \vee \bar{\chi}_1(b_*^1, b_*^2, b_*^3) \cdot x_{19+i}^{i-1}) = 1 \end{array} \right. \quad (1.3)$$

$i \in \{1, \dots, e\}$. Аналогичным образом определяются системы, описывающие работу второго и третьего РСЛОС в каждый такт с номером $i \in \{1, 2, \dots, e\}$. Остается добавить уравнения, описывающие формирование битов ключевого потока:

$$(y_i \equiv x_{19}^i \oplus x_{41}^i \oplus x_{64}^i) = 1, \quad i \in \{1, \dots, e\}.$$

Объединив все перечисленные уравнения, получим систему булевых уравнений, описывающую e шагов работы генератора А5/1. Таким образом,

решение задачи криптоанализа генератора А5/1 по известным битам ключевого потока сводится к решению полученной системы с подставленными в нее значениями переменных $y_1, y_2, \dots, y_e$.

1.2.2. Преобразования Цейтина

От произвольной системы $S(f_n)|_y$ возможен эффективный (в общем случае за полиномиальное от n время) переход к одному уравнению вида КНФ=1. Этот переход осуществляется при помощи преобразований Цейтина. При этом множество переменных разрастается (не более чем полиномиально), однако между множеством решений $S(f_n)|_y$ и множеством решений получаемого уравнения вида КНФ=1 существует биекция [49].

Преобразования Цейтина были предложены Г. С. Цейтиным в 1968 году в работе [50] (перепечатана на английском языке в [51]). Данные преобразования в последние годы активно используются в задачах верификации схем из функциональных элементов (см. [52–57]). Далее приведено описание преобразований Цейтина в контексте проблемы приведения систем булевых уравнений к нормальным формам.

Дано булево уравнение

$$F(h_1(x_1^1, \dots, x_{r_1}^1), \dots, h_s(x_1^s, \dots, x_{r_s}^s)) = 1. \quad (1.4)$$

Формулы $h_i(x_1^i, \dots, x_{r_i}^i)$ задают некоторые (в общем случае сложные) булевы функции. Положим

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

Введем булеву функцию $\varphi_{h_1}(x_1^1, \dots, x_{r_1}^1, u_1)$, $\varphi_{h_1} : \{0, 1\}^{r_1+1} \rightarrow \{0, 1\}$, значение которой на произвольном векторе $(\alpha_1^1, \dots, \alpha_{r_1}^1, \beta_1) \in \{0, 1\}^{r_1+1}$ опре-

делим следующим образом:

$$\varphi_{h_1}(\alpha_1^1, \dots, \alpha_{r_1}^1, \beta_1) = \begin{cases} 1, & h_1(\alpha_1^1, \dots, \alpha_{r_1}^1) = \beta_1 \\ 0, & h_1(\alpha_1^1, \dots, \alpha_{r_1}^1) = \overline{\beta_1} \end{cases}$$

Сказанное означает, что функция φ_{h_1} может быть задана булевой формулой следующего вида:

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

Рассмотрим булево уравнение

$$\Sigma(\varphi_{h_1}(x_1^1, \dots, x_{r_1}^1, u_1)) \cdot F_{h_1(x_1^1, \dots, x_{r_1}^1) \rightarrow u_1}(x_1, \dots, x_n, u_1) = 1. \quad (1.5)$$

Здесь через $\Sigma(\varphi_{h_1}(x_1^1, \dots, x_{r_1}^1, u_1))$ обозначено представление функции φ_{h_1} над $\{x_1^1, \dots, x_{r_1}^1, u_1\}$ произвольной булевой формулой (как правило, для представления φ_{h_1} используется некоторая нормальная форма — ДНФ, КНФ или АНФ). Через $F_{h_1(x_1^1, \dots, x_{r_1}^1) \rightarrow u_1}(x_1, \dots, x_n, u_1)$ обозначена булева формула, полученная заменой вхождений одной или нескольких (быть может, всех) формул $h_1(x_1^1, \dots, x_{r_1}^1)$ в левой части (1.4) на вхождения литерала u_1 . Переход от уравнения (1.4) к уравнению (1.5) — это одна итерация преобразований Цейтина в применении к булевым уравнениям.

Теорема 1.2 (см. [49]) Множество решений уравнения (1.4) пусто тогда и только тогда, когда пусто множество решений уравнения (1.5). Если эти множества не пусты, то существует взаимно однозначное соответствие между ними. Переход от произвольного решения уравнения (1.5) к соответствующему решению уравнения (1.4) осуществляется за линейное время.

Пример 1.4. В примере 1.3 было показано, что задача логического криптоанализа генератора поточного шифрования А5/1 сводится к решению системы булевых уравнений вида (1.3). Тем не менее, вопрос о способе решения данной системы остается открытым. Например, для использования SAT-решателя требуется преобразовать систему к уравнению вида

КНФ=1. Покажем, как это можно сделать при помощи преобразований Цейтина на примере одного уравнения из (1.3).

Рассмотрим уравнение

$$(x_1^i \equiv \chi_1(b_*^1, b_*^2, b_*^3) \cdot f(x_1^{i-1}, \dots, x_{19}^{i-1}) \vee \overline{\chi_1}(b_*^1, b_*^2, b_*^3) \cdot x_1^{i-1}) = 1. \quad (1.6)$$

Учтем, что

$$f(x_1^{i-1}, \dots, x_{19}^{i-1}) = x_{19}^{i-1} \oplus x_{18}^{i-1} \oplus x_{17}^{i-1} \oplus x_{14}^{i-1}.$$

Введем дополнительные булевы переменные y_1, y_2 и рассмотрим формулы

$$y_1 \equiv \chi_1(b_*^1, b_*^2, b_*^3), \quad y_2 \equiv x_{19}^{i-1} \oplus x_{18}^{i-1} \oplus x_{17}^{i-1} \oplus x_{14}^{i-1}.$$

Рассмотрим уравнение

$$(y_1 \equiv \chi_1(b_*^1, b_*^2, b_*^3)) \cdot (y_2 \equiv x_{19}^{i-1} \oplus x_{18}^{i-1} \oplus x_{17}^{i-1} \oplus x_{14}^{i-1}) \cdot (x_1^i \equiv y_1 \cdot y_2 \vee \overline{y_1} \cdot x_1^{i-1}) = 1. \quad (1.7)$$

В соответствии со сказанным выше между множествами решений уравнений (1.6) и (1.7) существует биекция. Формулы в скобках в левой части (1.7) можно представить в виде КНФ, используя таблицы истинности.

Поступим аналогичным образом со всеми уравнениями системы (1.3). Все полученные КНФ объединим конъюнкцией. В результате имеем уравнение вида КНФ=1, между множеством решений которого и множеством решений системы (1.3) существует биекция.

1.3. Основные алгоритмы решения систем булевых уравнений

Для решения систем булевых уравнений можно использовать целый ряд методов. Одним из первых булевы уравнения начал решать основоположник булевой алгебры Дж. Буль [58]. Используемый им подход осно-

вывался на процедурах «переразрешения» уравнения или системы посредством выражения одних переменных через другие. Эти идеи развивались более 100 лет, и долгое время соответствующие методы (методы «булевой унификации») были доминирующими ([59–63]).

На сегодняшний день можно сделать вывод, что идеи булевой унификации малопригодны для решения уравнений, кодирующих аргументированно трудные задачи обращения дискретных функций. Это связано, как правило, с существенным разрастанием формул при попытках явного выражения одних переменных (скажем, переменных входа рассматриваемой функции) через другие. Данный эффект хорошо замечен при переразрешении систем, кодирующих криптографические алгоритмы (например, DES [64]).

Ниже перечислены некоторые подходы к решению булевых уравнений, показывающие хорошие результаты на обширных классах тестовых задач и интенсивно развивающиеся в последние годы.

- Сведение произвольных систем булевых уравнений к системам полиномиальных уравнений над полем $GF(2)$ с последующим их решением при помощи разнообразных модификаций алгоритма Бухбергера [65, 66];
- Методы, использующие идеи линеаризации (к последним относится, например, метод линеаризационного множества, предложенный и развитый в работах [67–69]);
- SAT-подход, в основе которого лежит техника перехода от произвольной системы булевых уравнений к одному уравнению вида $KNF=1$ с сохранением ряда свойств (преобразования Цейтина) [1, 44, 70, 71];
- Подход, использующий в своей основе двоичные диаграммы решений

(Binary Decision Diagrams, BDD, ROBDD), [72–74].

Наилучшие результаты при решении булевых уравнений, кодирующих аргументированно трудные задачи обращения дискретных функций, на данный демонстрирует SAT-подход, использующий решатели, базирующиеся на алгоритме DPLL ([16, 75–78]).

1.4. Необходимость разработки методов и средств преобразования процедурных описаний алгоритмов в булевы уравнения

Все сказанное выше аргументирует актуальность следующей задачи: разработать и реализовать программный комплекс, осуществляющий преобразование процедурных описаний алгоритмов, вычисляющих дискретные функции, в булевы уравнения. Такой комплекс может быть использован при исследовании дискретных систем из широкого класса.

Как известно, многие NP-трудные задачи возникли из совершенно конкретных практических постановок. В ряде направлений без умения решать данные задачи невозможно обойтись. Речь идет о проблемах синтеза и верификации дискретных управляющих/управляемых систем, проблемах экономики, производственного планирования, логистики и многих других. В этих областях вопросы построения практически эффективных алгоритмов решения соответствующих комбинаторных задач чрезвычайно актуальны. Большое число исследований посвящено построению приближенных алгоритмов решения NP-трудных проблем. Однако в некоторых приложениях, например, в задачах верификации микросхем приближенные алгоритмы (даже обладающие малой погрешностью) совершенно бесполезны, а необходимость поиска точных решений существенно сужает класс методов —

методы непрерывной математики, генетические алгоритмы, разнообразные «эволюционные эвристики» в общем случае не дают точных решений.

С другой стороны, проблематика решения булевых уравнений (особенно в форме SAT-задач) демонстрирует в последние годы впечатляющий алгоритмический прогресс. Современные алгоритмы решения SAT-задач можно воспринимать как эффективные вычислительные методы, применимые к решению задач из различных областей кибернетики. Сказанное означает, что автоматизированная система, позволяющая сводить к булевым уравнениям проблемы исследования различных дискретных систем, является практически востребованной.

Как уже отмечалось, в противовес потоку статей по алгоритмам поиска решений булевых уравнений, работ, посвященных методам сведения к булевым уравнениям комбинаторных проблем, относительно немного. Хорошим введением в эту проблематику является обзор [1]. В [11], [79], [5] и некоторых других работах рассматривались процедуры построения КНФ, кодирующих различные комбинаторные «игровые» задачи (задача о ферзях, задача о Ханойской башне и др.). Механизмы преобразования в булевы уравнения задач планирования и составления расписаний были описаны в статьях [80–82]. Ряд работ посвящен использованию «SAT-технологий» в общей задаче удовлетворения ограничений (CSP) [83–87]. В статьях [8–10] аппарат булевых уравнений применяется к планированию вычислений в распределенных средах.

Следует отметить, что в большинстве перечисленных работ пропозициональное кодирование применялось к конкретным классам задач и, в основном, использовало технику программирования в булевых ограничениях.

Между тем, как уже отмечалось выше, весьма широкий класс комбинаторных проблем допускает описание в контексте задач обращения по-

линомиально вычислимых дискретных функций. В этой ситуации ставится задача преобразования в булевы уравнения некоторого процедурного описания рассматриваемой функции. В качестве такого описания обычно выступает программа для формальной вычислительной модели (например, для машины Тьюринга). Однако работать с языками формальных моделей на практике весьма затруднительно.

Решить проблему преобразования в булевы уравнения алгоритмических описаний дискретных функций в принципе возможно при помощи современных систем описания аппаратуры [88, 89]. Однако эти системы ориентированы на задачи проектирования аппаратных реализаций алгоритмов. Поэтому при программировании на данных языках приходится учитывать многочисленные нюансы, обусловленные особенностями реализации рассматриваемого алгоритма в аппаратной архитектуре. Так, язык VHDL основан на параллельной многопроцессорной модели. Нижний уровень модели образует архитектура виртуального процессорного элемента (ВПЭ), а верхний уровень — множество ВПЭ, объединенных некоторой системой межпроцессорных связей.

Структура ВПЭ состоит из АЛУ (арифметико-логическое устройство), ОЗУ (оперативное запоминающее устройство) данных, ОЗУ программы, источников сигналов (входы) и приемников сигналов (выходы). АЛУ выполняет элементарные операции над данными (дискретные преобразования). В ОЗУ программ хранится программа в виде последовательности операторов. Операторы, включая условные операторы, выполняются последовательно друг за другом, как в обычных языках программирования. Такая программа в VHDL-программе называется процессом — ВПЭ выполняет его как отдельный вычислительный процесс.

Таким образом, задача описания алгоритма вычисления дискретной функции на языке VHDL превращается в задачу описания взаимодействий

между перечисленными выше элементами многопроцессорной модели с перечисленными ниже особенностями:

1. все процессы исполняются параллельно;
2. необходимо учитывать временные задержки в распространении сигналов между ВПЭ;
3. работа процессов должна быть синхронизирована.

Сказанное означает, что системы разработки и верификации аппаратуры слабо пригодны для задач преобразования в булевы уравнения описаний алгоритмов, так как требуют от исследователя квалификации разработчика схемотехнических устройств.

Основные выводы по первой главе. В данной главе обоснована актуальность проблемы преобразования процедурных описаний дискретных функций в булевы уравнения. На основе формальных вычислительных моделей (RAM) сформулированы базовые принципы методологии пропозиционального кодирования алгоритмов. Все эти принципы допускают перенос на современные вычислительные архитектуры и современные средства программирования.

Основной вывод первой главы состоит в необходимости разработки специализированного языка описания дискретных функций, который обладал бы, с одной стороны, синтаксисом, близким к современным процедурным языкам, а с другой, семантикой, позволяющей эффективно преобразовывать соответствующие процедурные описания в булевы уравнения. Такой язык, а также механизмы преобразования выражений на данном языке в булевы уравнения представлены во второй главе диссертации.

Глава 2

Преобразование процедурных описаний дискретных функций в булевы уравнения

Далее мы развиваем общую технику преобразования в булевы уравнения программ вычисления дискретных функций, написанных на специальном высокоуровневом языке. Получаемая система булевых уравнений является результатом применения к рассматриваемой программе пошаговой интерпретации специального вида. Фактически речь идет о символьном исполнении программы (symbolic execution, [90–93]) — технике, используемой в статическом анализе программ [94, 95]. Мы позволим себе привести цитату из статьи [90], которая является одной из первых работ по символьному исполнению программ.

Каждый язык программирования имеет семантику исполнения, описывающую правила, по которым операторы языка манипулируют входными данными в соответствии с инструкциями программы. Также мы можем определить альтернативную семантику символьного исполнения для языка программирования, в котором реальные входные данные не используются, но могут описываться некоторыми символами. Символьное исполнение — это естественное расширение обычного исполнения в том смысле, что обычное исполнение является конкретизацией символьного. При символьном исполнении семантика основных операторов языка расширена таким образом, чтобы принимать на входе символы и выдавать на выходе формулы.

Особо отметим, что процесс пропозиционального кодирования алгоритмов в том смысле, как он описан в первой главе, представляет собой специальный случай символьного исполнения программ. Действительно,

при пропозициональном кодировании результатом интерпретации каждой инструкции программы является формула исчисления высказываний и булево уравнение вполне конкретного вида. Особенность пропозиционального кодирования заключается в том, что обе ветви произвольного оператора условного перехода могут быть эффективно преобразованы в одну формулу. В этом основное отличие от ранних работ по символьному исполнению программ (см. [90]), в которых каждая ветвь условного оператора интерпретируется отдельной формулой.

Далее представлен проблемно-ориентированный процедурный язык (язык ТА) (см. [96]), обладающий операционной семантикой (см. [13]). Программы на данном языке (ТА-программы) представляют собой алгоритмические описания дискретных функций. При интерпретации произвольной ТА-программы происходит ее символьное исполнение, результатом которого является система булевых уравнений $S(f_n)$. Подстановка в $S(f_n)$ некоторого $y \in \text{range } f_n$ дает систему $S_y(f_n)$, из произвольного решения которой можно эффективно выделить такое слово $x \in \{0, 1\}^n$, что $f_n(x) = y$. Для поиска решений систем вида $S_y(f_n)$ можно использовать современные булевы решатели.

2.1. Описание проблемно-ориентированного языка ТА

Язык ТА представляет собой процедурный язык описания алгоритмов с блочной структурой и С-подобным синтаксисом (стандарт ISO/IEC 9899:1999). Вся терминология, применяемая далее при описании языка ТА, является стандартной и может быть найдена в любом из общедоступных источников по теории программирования и компиляции (например, см. [97–99]). Особо отметим, что некоторые конструкции языка ТА совпадают с их прототипами в языке С [100]. Далее приводится описание всех

основных конструкций языка ТА с использованием нотации Бэкуса-Наура (БНФ).

В таблице 2.1 приведены лексемы (терминальные символы грамматики) языка программирования ТА. Терминальные символы могут разделяться произвольным количеством «пустых» символов. К пустым символам относятся пробелы, переносы строк, табуляция.

Таблица 2.1. Лексемы языка ТА.

Категория лексем	Лексемы
Ключевые слова	<code>_in</code> , <code>_out</code> , <code>_mem</code> , <code>define</code> , <code>void</code> , <code>int</code> , <code>bool</code> , <code>bit</code> , <code>if</code> , <code>else</code> , <code>for</code> , <code>return</code>
Идентификаторы	<code>[a – zA – Z_0 – 9]</code>
Целочисленные константы	<code>[0 – 9]</code>
Разделители	<code>,</code> <code>[</code> <code>]</code> <code>(</code> <code>)</code> <code>{</code> <code>}</code> <code>;</code> <code>:</code>
Операции	<code>=</code> , <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>%</code> , <code>&</code> , <code> </code> , <code>^</code> , <code>!</code> , <code> </code> , <code><<</code> , <code>>></code> , <code><< *</code> , <code>>> *</code> , <code><<<</code> , <code>>>></code> , <code>></code> , <code>>=</code> , <code><</code> , <code><=</code> , <code>==</code> , <code>!=</code>

Алфавит идентификаторов в языке ТА содержит все буквы латинского алфавита, все цифры, а также символ подчеркивания. Идентификатор не может начинаться с цифры. Идентификаторы языка ТА чувствительны к регистру. Например, идентификаторы `MyVar`, `myvar` и `MYVAR` могут быть бесконфликтно связаны с тремя различными переменными, относящимися к одной области видимости. Идентификаторы связываются с переменными, массивами, функциями и именованными константами. Любой идентификатор языка относится к некоторой области видимости, в пределах которой он может использоваться.

Понятие области видимости, используемое в языке ТА, стандартно (см. [97, 100]). Используются два типа области видимости — глобальная и локальная. Любой блок в языке ТА создает локальную область видимости.

сти. В глобальную область видимости попадают все переменные (массивы), объявленные вне какого-либо блока. Тело любой функции заключается в фигурные скобки и образует локальную область видимости (блок). Определение функций допускается только в глобальной области видимости, т.е. запрещено определять функцию в теле любой другой функции.

Программа на языке ТА представляет собой список функций. Среди них выделяется функция с именем `main`, с вызова которой начинается выполнение программы. При входе из функции `main` выполнение программы завершается. Данная функция присутствует в любой ТА-программе. Кроме определения функций в глобальной области видимости могут присутствовать объявления переменных, массивов и объявления именованных констант.

```
<program> ::= <definition_list>
<definition_list> ::= <definition> <definition_list>|<empty>
<definition> ::= <var_definition>|<array_definition>|
                  <const_definition>|<function_definition>
```

В языке ТА реализованы все основные примитивные конструкции, характерные для процедурных языков программирования:

1. Объявление/определение переменной или массива переменных;
2. Определение именованной константы;
3. Определение функции;
4. Оператор присваивания;
5. Составной оператор;
6. Оператор условного перехода;
7. Оператор цикла;
8. Оператор вызова функции;
9. Оператор возврата из функции.

Язык ТА является проблемно-ориентированным языком программирования с операционной семантикой [13, 96]. Для описания процесса его

интерпретации в пункте 2.2.1 вводится абстрактная (языковая) ТА-машина.

Типы данных. В языке ТА поддерживается несколько типов данных. Основным является тип `bit`. Этот тип служит для объявления переменных, связанных с областями памяти ТА-машины, в которых хранятся входные/выходные слова, а также логические выражения, получаемые в процессе символьного исполнения ТА-программы. Более подробное описание работы с типом `bit` приведено в пункте 2.2.1.

Целочисленный тип `int` представляет собой четырехбайтовое знаковое целое число. Переменные данного типа хранят целочисленные параметры ТА-программы. Например это могут быть длины входного и выходного слов, количество итераций в циклах, целочисленные константы, используемые при вычислении дискретной функции.

Тип `bool` используется для объявления целочисленных переменных, принимающих значения в множестве $\{0, 1\}$. Данный тип применяется при сведении задач с псевдобулевыми ограничениями к SAT-задачам (см. пункт 3.3).

Тип `void` играет служебную роль — он используется в заголовках определений функций на месте типа возвращаемого значения в том случае, если функция не должна ничего возвращать.

Константы. Целочисленные константы кроме десятичной системы счисления можно записывать в восьмеричной или шестнадцатеричной системе. Если целая константа начинается с 0, то она записана в восьмеричном представлении, а если с 0x или 0X — в шестнадцатеричном.

В языке ТА имеется возможность объявления именованных констант, т.е. таких констант, с которыми связываются идентификаторы. Объявление именованной константы начинается с ключевого слова `define`, за которым следует идентификатор и константа.

```
<const_def> ::= "define" <id> <const>
```

Переменные и массивы. Объявление переменной начинается с ключевого слова, обозначающего тип данных, за которым следует идентификатор переменной. Переменные целочисленных типов могут инициализироваться константами, другими целочисленными переменными или выражениями над ними. В случае отсутствия явной инициализации по умолчанию такая переменная инициализируется константой 0. Переменные типа `bit` могут инициализироваться термами (выражениями) над другими переменными типа `bit`. Глобальные переменные типа `bit` могут объявляться с атрибутом `_in` или `_out`. При объявлении локальных переменных типа `bit` использовать атрибуты `_in` и `_out` запрещено. Назначение данных атрибутов поясняется в пункте 2.2. Также в языке ТА используется атрибут `_mem`, применяемый для декомпозиции сложных термов (см. пункт 2.2.3).

```

    <attr> ::= ["__in" | "__out" | "__mem"]
    <var_definition> ::= <attr> <data_type> <id> ["=" <expression>];

```

Объявление массива отличается от объявления переменной списком индексов, перечисляемых после идентификатора. Каждый индекс в данном списке заключен в отдельные квадратные скобки. Язык TA допускает объявление массивов любых размерностей.

```

<array_definition> ::= <attr> <data_type> <id> "["<const>"]
                        {"["<const>"]"} ["=" <array_init>];

```

Пример 2.1. Объявление и инициализация двумерного целочисленного массива:

```
int reg[3][2] = {{1,2}, {3,4}, {5,6}};
```

Выражения языка ТА. Структура допустимого выражения языка ТА описывается в БНФ следующим образом:

```

<expression> ::= <operand> {<operation> <operand>}
  <operand> ::= <id> [{"["<expression>"]"} | "("<param_list>")" |
    <const> | ["!" | "-" ]<operand> | "("<expression>")"
  <operation> ::= "+" | "-" | "*" | "/" | "&" | "^" | "|" | "!" | "||" | "<" | ">" |
    "<<" | ">>" | "<*" | ">*" | ">" | ">=" | "<" | "<=" | "==" | "!=" | "="

```

При синтаксическом разборе программ выражения удобно представлять в виде бинарных деревьев, структура которых зависит от приоритетов операций (рисунок 2.1).

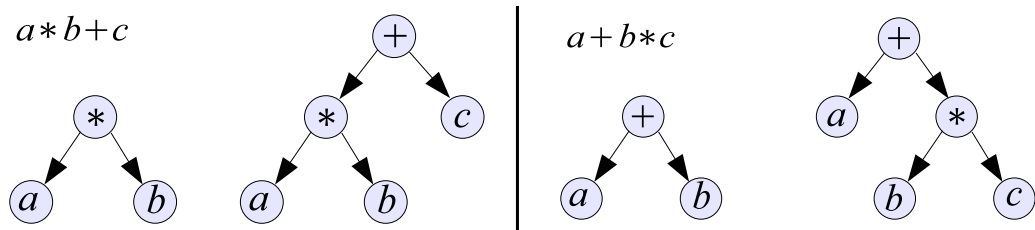


Рис. 2.1. Структуры данных для представления выражений языка ТА.

В таблице 2.2 перечислены поддерживаемые в языке ТА операции и указан их приоритет (приоритет операции тем выше, чем меньше соответствующее численное значение).

Оператор присваивания. Левым операндом оператора присваивания может быть только переменная (элемент массива). Далее следует терминальный символ грамматики «=», после которого следует любое допустимое выражение языка ТА. При этом данные, являющиеся результатом интерпретации этого выражения, должны иметь тот же самый тип (или преобразуемый к нему), что и переменная, являющаяся левым операндом оператора присваивания. Далее приведено описание оператора присваивания языка ТА в БНФ:

```

<assignment> ::= <lValue> "=" <expression>
  <lValue> ::= <id> [{"["<expression>"]"}

```

Составной оператор. Составной оператор — это список инструкций ТА-программы, разделенных символом «;», заключенный в фигурные скобки. В соответствии с подходом, принятым в языке C [100], всякий состав-

Таблица 2.2. Приоритеты операций языка ТА.

Операция	Приоритет
!, – (унарный)	1
*, /, %	2
+, –	3
<<, >>, << *, >> *, <<<, >>>	4
<, >, <=, >=	5
==, !=	6
&	7
	8
^	9
	10
=	11

ной оператор языка ТА является блоком. Синтаксис составного оператора в БНФ:

```
<block> ::= "{" <statement_list> "}"
<statement_list> ::= <statement> <statement_list> | <empty>
```

Оператор условного перехода. Оператор условного перехода осуществляет выбор одной из нескольких альтернативных ветвей выполнения программы. Описание данного оператора на языке БНФ выглядит следующим образом:

```
<if_stmt> ::= "if" "(" <expression> ")" <stmt>
           {"else" "if" "(" <expression> ")" <stmt>}
           ["else" <stmt>]
```

Оператор условного перехода начинается с ключевого слова `if`, за которым следует любое допустимое выражение языка ТА. Язык ТА поддерживает вложенные условные переходы. При этом допускается использо-

вание конструкции `else if`, поддерживаемой в С-подобных языках программирования. При символьном исполнении ТА-программ механизмы интерпретации условных операторов существенным образом отличаются от общепринятых. Подробное описание этих механизмов приведено в пункте 2.2.2.

Оператор цикла. Цикл используется для многократного выполнения некоторого оператора. Описание на языке БНФ:

```
<for_stmt> ::= "for" "(" <stmt> ";" <expression> ";" <stmt> ")" <stmt>
```

Цикл состоит из заголовка и тела. Заголовок цикла начинается с ключевого слова `for`, за которым в круглых скобках через точку с запятой перечисляются оператор инициализации цикла, предикат цикла и шаг цикла. Интерпретация оператора цикла в процессе символьного исполнения ТА-программы рассматривается в пункте 2.2.2.

Определение функций. Средства языка ТА позволяют определять пользовательские функции. Синтаксис определения функции в БНФ имеет следующий вид:

```
<function_definition> ::= <data_type> <id> "(" <param_list> ")" <body>
    <param_list> ::= <param> {"," <param>} | <empty>
    <param> ::= <data_type> <id>
```

Пример 2.2. Определение на языке ТА булевой функции от трех переменных:

```
bit majority(bit x, bit y, bit z){
    return x&y | x&z | y&z;
}
```

Оператор вызова функции и оператор возврата из функции. Оператор вызова функции состоит из идентификатора вызываемой функции, за которым в круглых скобках через запятую перечисляются фактические параметры функции. Синтаксис оператора вызова функции в БНФ:

```
<function_call> ::= <id> "(" <call_params> ")"  
<call_params> ::= <empty> | <expression> {"," <expression>}
```

Оператор возврата из функции начинается с ключевого слова **return**, за которым следует некоторое выражение языка ТА:

```
<return_stmt> ::= "return" <expression>
```

Комментарии. В языке ТА, так же, как и в языке С, поддерживаются два вида комментариев — строковый и блочный.

2.2. Интерпретация ТА-программ. Формирование пропозиционального кода алгоритма

В данном разделе подробно описана семантика языка ТА, обеспечивающая символьное исполнение ТА-программ и построение систем булевых уравнений.

2.2.1. Базовые механизмы символьного исполнения ТА-программ

В соответствии с принятым подходом для определения интерпретационной семантики (см. [13]) необходимо описание абстрактной машины, исполняющей инструкции ТА-языка (т. н. «языковая машина», [13]). Произвольная конфигурация абстрактной машины ТА-языка (далее «ТА-машина») фиксирует момент интерпретации одной или нескольких инструкций ТА-программы. При этом в памяти ТА-машины отображается информация о связи некоторых переменных, фигурирующих в тексте ТА-программы, с переменными, фигурирующими в системе булевых уравнений, и с логическими выражениями (термами).

Следует особо подчеркнуть, что переменные, встречающиеся в тексте программы (далее «переменные программы»), являются идентификаторами областей памяти ТА-машины. Переменные, встречающиеся в системе булевых уравнений, — это булевы переменные, называемые далее переменными кода. Таким образом, переменные программы и переменные кода — это разные сущности. Тем не менее переменные программы и переменные кода в памяти ТА-машины связываются при помощи специальной структуры данных `var_object`.

Как уже говорилось выше, в языке ТА основным типом данных является тип `bit`. Как правило, переменная, имеющая этот тип, в каждой конфигурации ТА-машины связана с некоторой переменной кода и некоторым термом. Схематично данная связь отображена на рисунке 2.2.

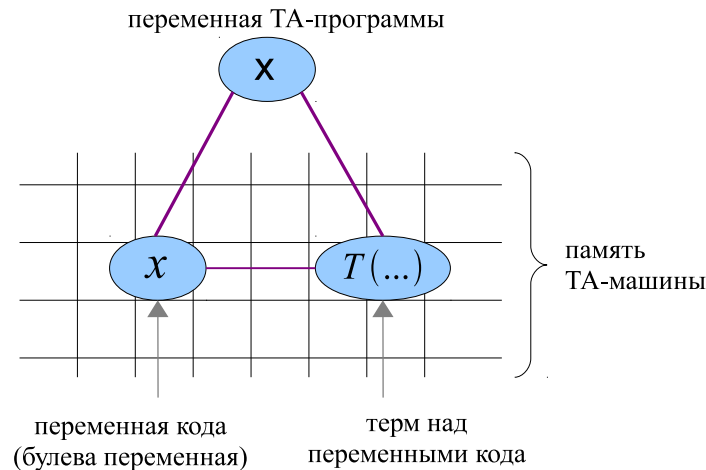


Рис. 2.2. Связь переменной ТА-программы типа `bit` с переменной кода.

Таким образом, значениями переменной x , имеющей тип `bit`, в памяти ТА-машины являются слова вида

$$x|T(u_1, \dots, u_k).$$

Здесь x — переменная кода, связанная с x , $T(u_1, \dots, u_k)$ — терм исчисления высказываний над булевыми переменными (переменными кода) $u_1, \dots, u_k$ («|» — разделяющий символ).

Допустимыми операциями над переменными, имеющими тип **bit**, являются произвольные операции исчисления высказываний. Так, если x — переменная типа **bit**, значение которой в памяти ТА-машины представлено словом

$$x|T(u_1, \dots, u_k),$$

и в ТА-программе имеется инструкция вида $y = \bar{x}$, то значение переменной y в памяти ТА-машины будет представлено словом

$$y|\neg T(u_1, \dots, u_k).$$

Пусть x, y — переменные типа **bit**, значения которых в памяти ТА-машины представлены словами

$$x|T(u_1, \dots, u_k), \quad y|R(v_1, \dots, v_l)$$

соответственно, $\circ$ — произвольная бинарная логическая связка, и в ТА-программе имеется инструкция вида $z = x \circ y$. Тогда значение переменной z в памяти ТА-машины будет представлено словом

$$z|T(u_1, \dots, u_k) \circ R(v_1, \dots, v_l).$$

Процесс интерпретации ТА-программы — это процесс ее последовательного символьного исполнения: результатом исполнения произвольной инструкции являются некоторая формула (терм) исчисления высказываний и булево уравнение специального вида, которые конструируются в памяти ТА-машины.

Таким образом, что процессу символьного исполнения ТА-программы соответствует последовательность конфигураций ТА-машины: $K^0, \dots, K^e$ (K^0 — начальная, K^e — заключительная конфигурации). Для каждого $i \in \{0, \dots, e\}$ обозначим через X^i множество переменных кода, связанных с

переменными программы в конфигурации K^i . Тогда

$$X = \bigcup_{i=0}^e X^i$$

это множество всех переменных кода, введенных в процессе символьного исполнения рассматриваемой ТА-программы. Множество X^0 образовано булевыми переменными, которые кодируют входные данные дискретной функции. Множество переменных кода, которые кодируют выход этой функции, будем обозначать через Y ($Y \subset X$).

Переменные программы, связанные в конфигурации K^0 с переменными кода из множества X^0 , объявляются в тексте ТА-программы с атрибутом «_in». Переменные программы, которые в процессе символьного исполнения связываются с переменными кода из множества Y , объявляются с атрибутом «_out». Переменные программы с атрибутами _in и _out — это переменные с глобальной областью видимости.

Пример 2.3. Рассмотрим ТА-программу, которая реализует регистр сдвига с линейной обратной связью (РСЛОС, [45]), заданной полиномом (над $\text{GF}(2)$) $P(Z) = Z^{19} + Z^{18} + Z^{17} + Z^{14} + 1$ (Z — формальная переменная).

```
__in bit reg[19];
__out bit output[100];
bit shiftReg(){
    bit x = reg[18];
    bit y = reg[18]^reg[17]^reg[16]^reg[13];
    for(int j = 18; j > 0; j = j - 1){
        reg[j] = reg[j - 1];
    }
    reg[0] = y;
    return x;
}
void main(){
    for(int i = 0; i < 100; i = i + 1){
```

```

    output[i] = shiftReg();
}
}

```

Массив переменных `reg` описывает текущее состояние регистра сдвига. На каждом шаге переменные программы `reg[0], ..., reg[18]` связываются через структуры `var_object` с переменными кода. Так, на начальном шаге переменные `reg[0], ..., reg[18]` связаны с переменными кода, образующими множество $X^0 = \{x_1, \dots, x_{19}\}$.

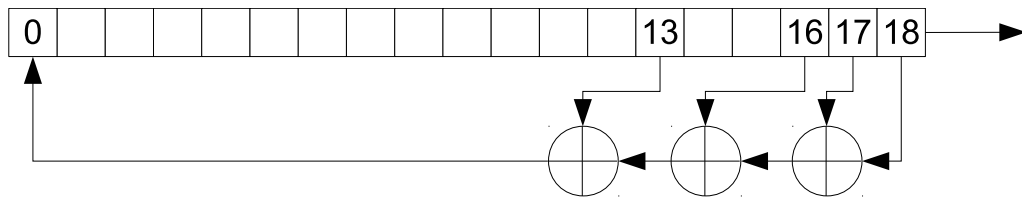


Рис. 2.3. Схема РСЛОС с полиномом обратной связи $P(Z) = Z^{19} + Z^{18} + Z^{17} + Z^{14} + 1$.

Представленная программа реализует 100 тактов работы регистра сдвига. В теле основной функции `main` организован цикл, в котором вызывается функция сдвига регистра `shiftReg`. При символьном исполнении ТА-программы значения, возвращаемые данной функцией, — это булевы переменные, кодирующие биты выхода рассматриваемого РСЛОС. С данными переменными через структуры `var_object` связаны переменные, объявляемые в тексте программы с атрибутом `_out`.

Сдвиг регистра обновляет значения всех элементов массива `reg`. В результате первого сдвига регистра вводятся новые переменные кода, образующие множество $X^1 = \{x_{20}, \dots, x_{38}\}$. Эти переменные связываются через структуры `var_object` с переменными программы `reg[0], ..., reg[18]` и через них с переменными множества X^0 . Данная связь интерпретируется

следующей системой булевых уравнений:

$$\left\{ \begin{array}{l} (x_{20} \equiv x_{19} \oplus x_{18} \oplus x_{17} \oplus x_{14}) = 1 \\ (x_{21} \equiv x_1) = 1 \\ (x_{22} \equiv x_2) = 1 \\ \dots \\ (x_{38} \equiv x_{18}) = 1 \end{array} \right. \quad (2.1)$$

Переменная x_{19} кодирует первый бит ключевого потока, полученный в результате первого сдвига рассматриваемого РСЛОС.

Можно заметить, что пропозициональный код, представляемый системой (2.1), является избыточным, поскольку переменные x_1 и x_{21} кодируют одну и ту же информацию. Аналогично для переменных x_2 и $x_{22}, \dots, x_{18}$ и x_{38} . Избежать подобных ситуаций позволяет описываемая далее техника, которая использует при интерпретации инструкций произвольных ТА-программ специальный словарь термов.

Рассмотрим произвольную ТА-программу, вычисляющую функцию $f_n : \{0, 1\}^n \rightarrow \{0, 1\}^*$. Словарь термов Σ определяется как динамически расширяемое множество. На начальном шаге в словаре Σ находятся только переменные из множества X^0 (то есть переменные, кодирующие входную информацию). В дальнейшем каждый новый терм, попадающий в словарь, является результатом преобразования некоторой операции присваивания следующего вида:

$$x = \Phi(x_{j_1}, \dots, x_{j_r}).$$

Здесь x — переменная программы, $\Phi(x_{j_1}, \dots, x_{j_r})$ — выражение над переменными программы. Интерпретатор обрабатывает выражение $\Phi(x_{j_1}, \dots, x_{j_r})$ и на его основе формирует терм $\varphi(x_{j_1}, \dots, x_{j_r})$ над переменными кода. Затем проверяется словарь термов Σ на предмет наличия в нем терма φ .

Если $\varphi \notin \Sigma$, то создается новая переменная кода $\tilde{x}$, терм $\varphi(x_{j_1}, \dots, x_{j_r})$ добавляется в Σ , а имеющаяся система булевых уравнений дополняется новым уравнением вида

$$(\tilde{x} \equiv \varphi(x_{j_1}, \dots, x_{j_r})) = 1.$$

Если $\varphi \in \Sigma$, это означает, что кодируемая этим термом информация уже учтена в пропозициональном коде преобразуемой программы, то есть в системе булевых уравнений присутствует уравнение вида

$$(x' \equiv \varphi(x_{j_1}, \dots, x_{j_r})) = 1,$$

где x' — некоторая переменная кода. В этом случае переменная программы x связывается с переменной кода x' при помощи структуры `var_object`. Именно этот прием позволяет избегать ввода булевых переменных, кодирующих одну и ту же информацию.

В соответствии с базовым принципом пропозиционального кодирования алгоритмов все переменные кода, входящие в множество X^i , определяются при помощи булевых функций от переменных кода, входящих в множество X^{i-1} , $i \in \{1, \dots, e\}$. При этом каждой переменной кода $\tilde{x} \in X^i$ соответствует булево уравнение вида

$$(\tilde{x} \equiv \varphi(x_{i_1}, \dots, x_{i_k})) = 1, \tag{2.2}$$

где $\{x_{i_1}, \dots, x_{i_k}\} \subseteq X^{i-1}$, $\varphi(x_{i_1}, \dots, x_{i_k})$ — терм исчисления высказываний. При интерпретации выражений языка ТА данный принцип реализуется при помощи словаря термов Σ : каждая переменная кода из множества $\bigcup_{i=1}^e X^i$ связывается с соответствующим ей термом $\varphi(x_{i_1}, \dots, x_{i_k}) \in \Sigma$ при помощи булева уравнения вида (2.2). Итогом этого процесса является система $S(f_n)$.

Пример 2.4. Рассмотрим процесс преобразования ТА-программы из примера 2.3. На начальном шаге в словарь термов включаются переменные

кода, образующие множество $X^0 = \{x_1, \dots, x_{19}\}$. При этом связи между переменными программы, объединенными в массив `reg`, и переменными кода из множества X^0 выглядят следующим образом:

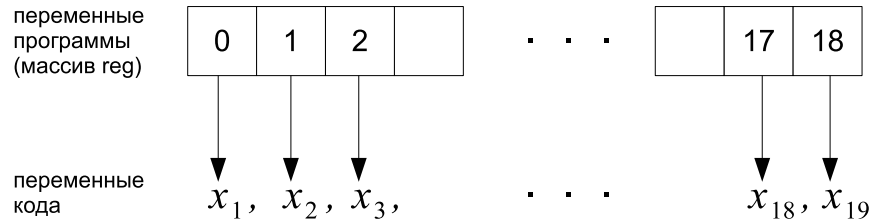


Рис. 2.4. Связь переменных программы с переменными кода до сдвига регистра.

При интерпретации рассматриваемой ТА-программы в словарь Σ добавляется терм

$$\varphi = x_{19} \oplus x_{18} \oplus x_{17} \oplus x_{14},$$

кодирующий функцию обратной связи, создается новая переменная кода x_{20} и добавляется в систему булевых уравнений уравнение

$$(x_{20} \equiv x_{19} \oplus x_{18} \oplus x_{17} \oplus x_{14}) = 1.$$

Кроме этого, обновляются связи между элементами массива `reg` и переменными кода (рисунок 2.5).

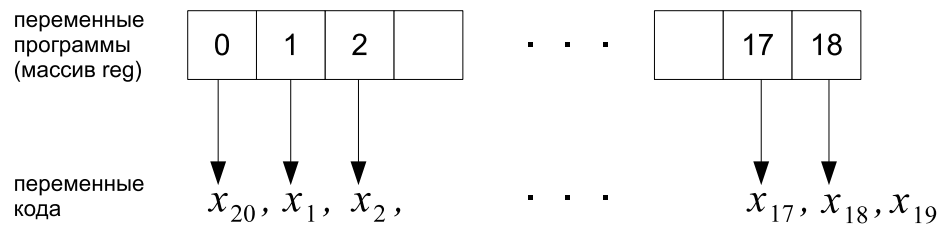


Рис. 2.5. Связь переменных программы с переменными кода после сдвига регистра.

Особо отметим, что для переменных программы `reg[1], ..., reg[18]` новые переменные кода не создавались — посредством структур `var_object` данные переменные были связаны с переменными кода $x_1, \dots, x_{18}$ соответственно.

2.2.2. Символьное исполнение основных операторов языка ТА

Оператор условного перехода. В первом разделе отмечалось, что наиболее трудный этап при преобразовании алгоритмов в булевы уравнения — это интерпретация условных переходов. Оператор условного перехода — это оператор вида (в псевдокоде)

$$\begin{aligned} &\text{if } \Phi(x_{j_1}, \dots, x_{j_r}) \text{ then Оператор_1;} \\ &\text{else Оператор_2;} \end{aligned}$$

здесь Оператор_1 и Оператор_2 — произвольные операторы языка ТА (оператор присваивания, оператор условного перехода, оператор цикла, оператор вызова функции или составной оператор), $\Phi(x_{j_1}, \dots, x_{j_r})$ — выражение над переменными программы.

Интерпретация оператора условного перехода начинается с анализа выражения

$$\Phi(x_{j_1}, \dots, x_{j_r}),$$

результатом которого является терм φ над переменными кода. После этого выполняется интерпретация операторов Оператор_1 и Оператор_2, в процессе которой строятся два альтернативных множества конфигураций ТА-машины Δ_1 и Δ_2 . Данные множества образованы конфигурациями, в которых зафиксированы изменения в памяти ТА-машины, являющиеся результатами выполнения операторов Оператор_1 и Оператор_2 соответственно. Затем интерпретатор определяет множество переменных программы, которые изменили свое значение хотя бы в одной из конфигураций, представленных в Δ_1 и Δ_2 . Пусть x — произвольная такая переменная, изменяющая свое значение по крайней мере в одной из конфигураций K_1 и K_2 ($K_1 \in \Delta_1$, $K_2 \in \Delta_2$). Пусть значениями x в K_1 и K_2 являются слова

$$\tilde{x}|\delta_1, \tilde{x}|\delta_2$$

соответственно. Здесь $\tilde{x}$ — новая переменная кода, δ_1, δ_2 — некоторые термы над переменными кода. Тогда в результате интерпретации оператора условного перехода с переменной программы x связывается новая переменная кода $\tilde{x}$, в словарь термов Σ добавляется терм $\varphi \cdot \delta_1 \vee \overline{\varphi} \cdot \delta_2$, а в систему булевых уравнений добавляется уравнение

$$(\tilde{x} \equiv \varphi \cdot \delta_1 \vee \overline{\varphi} \cdot \delta_2) = 1.$$

Далее рассматривается конструкция из нескольких вложенных операторов условного перехода (в псевдокоде).

```

if  $\Phi_1(\dots)$  then Оператор_1;
else if  $\Phi_2(\dots)$  then Оператор_2;
...
else if  $\Phi_n(\dots)$  then Оператор_n;
else Оператор_{n+1};

```

В соответствии со сказанным выше каждое выражение Φ_i , $i \in \{1, \dots, n\}$, преобразуется в терм φ_i над множеством переменных кода, строится $n + 1$ множество $\Delta_1, \dots, \Delta_{n+1}$ альтернативных конфигураций ТА-машины и определяется множество переменных программы, каждая из которых изменила свое значение хотя бы в одной из конфигураций, представленных в $\Delta_1, \dots, \Delta_{n+1}$. Пусть x — произвольная такая переменная и ее значениями в конфигурациях $K_1, \dots, K_{n+1}$, $K_j \in \Delta_j$, $j \in \{1, \dots, n + 1\}$, ТА-машины являются слова

$$\tilde{x}|\delta_1, \dots, \tilde{x}|\delta_{n+1}.$$

В результате интерпретации оператора условного перехода с переменной программы x связывается новая переменная кода $\tilde{x}$, в словарь термов Σ добавляется терм

$$\varphi_1 \cdot \delta_1 \vee \overline{\varphi_1} \cdot \varphi_2 \cdot \delta_2 \vee \dots \vee \overline{\varphi_1} \cdot \overline{\varphi_2} \cdot \dots \cdot \overline{\varphi_{n-1}} \cdot \varphi_n \cdot \delta_n \vee \overline{\varphi_1} \cdot \dots \cdot \overline{\varphi_n} \cdot \delta_{n+1},$$

а в систему булевых уравнений добавляется уравнение

$$(\tilde{x} \equiv \varphi_1 \cdot \delta_1 \vee \overline{\varphi_1} \cdot \varphi_2 \cdot \delta_2 \vee \dots \vee \vee \overline{\varphi_1} \cdot \overline{\varphi_2} \cdot \dots \cdot \overline{\varphi_{n-1}} \cdot \varphi_n \cdot \delta_n \vee \overline{\varphi_1} \cdot \dots \cdot \overline{\varphi_n} \cdot \delta_{n+1}) = 1.$$

Пример 2.5. Рассматриваются два РСЛОС, обозначаемые далее «regA» и «regB». Рассмотрим ситуацию, в которой условие сдвига для каждого РСЛОС определяется значением некоторой булевой функции от битов его текущего состояния. Иными словами, сдвиг регистра происходит лишь тогда, когда значение данной функции равно 1. Алгоритм, реализующий функцию сдвига произвольного РСЛОС, описан в примере 2.3, поэтому определения процедур `shiftRegA` и `shiftRegB` в следующей ТА-программе опущены.

```
__in bit regA[19];
__in bit regB[17];
__out bit output[100];
bit shiftRegA(){...}
bit shiftRegB(){...}
bit conditionA(){return regA[18] ^ regB[16];}
bit conditionB(){return regA[18] & regB[16];}
void main(){
    for(int i = 0; i < 100; i = i + 1){
        if(conditionA())
            shiftRegA();
        else if(conditionB())
            shiftRegB();
        else{
            shiftRegA();
            shiftRegB();
        }
        output[i] = regA[18] ^ regB[16];
    }
}
```

На начальном шаге преобразования формируется множество переменных кода

$$X^0 = \{\underbrace{x_1, \dots, x_{19}}_{\text{regA}}, \underbrace{x_{20}, \dots, x_{36}}_{\text{regB}}\},$$

кодирующих содержимое массивов **regA** и **regB** (вход рассматриваемой дискретной функции).

Обрабатывая предикаты оператора условного перехода, интерпретатор строит термы над переменными кода

$$\varphi_1 = x_{19} \oplus x_{36}, \varphi_2 = x_{19} \cdot x_{36}.$$

В процессе обработки ветвей оператора условного перехода формируются альтернативные конфигурации ТА-машины K_1 , K_2 и K_3 . Обозначим через δ_1 и δ_2 термы, полученные в результате преобразования выражений программы, описывающих функции обратной связи регистров **regA** и **regB**. Термы δ_1 и δ_2 добавляются в словарь Σ , а система булевых уравнений дополняется уравнениями

$$(x_{37} \equiv \delta_1) = 1, (x_{38} \equiv \delta_2) = 1.$$

Для обновления связей между переменными программы и переменными кода в процессе интерпретации рассматриваемого условного оператора формируются структуры данных, схематично изображенные на рисунке 2.6.

Рассмотренная схема дает эффективную процедуру связывания переменных программы с переменными кода. В результате обработки рассматриваемого оператора условного перехода, например переменная **regA[0]** будет связана через структуру **var_object** с новой переменной кода x_{39} , в словарь Σ будет добавлен терм

$$\varphi_1 \cdot x_{37} \vee \overline{\varphi_1} \cdot \varphi_2 \cdot x_1 \vee \overline{\varphi_1} \cdot \overline{\varphi_2} \cdot x_{37},$$

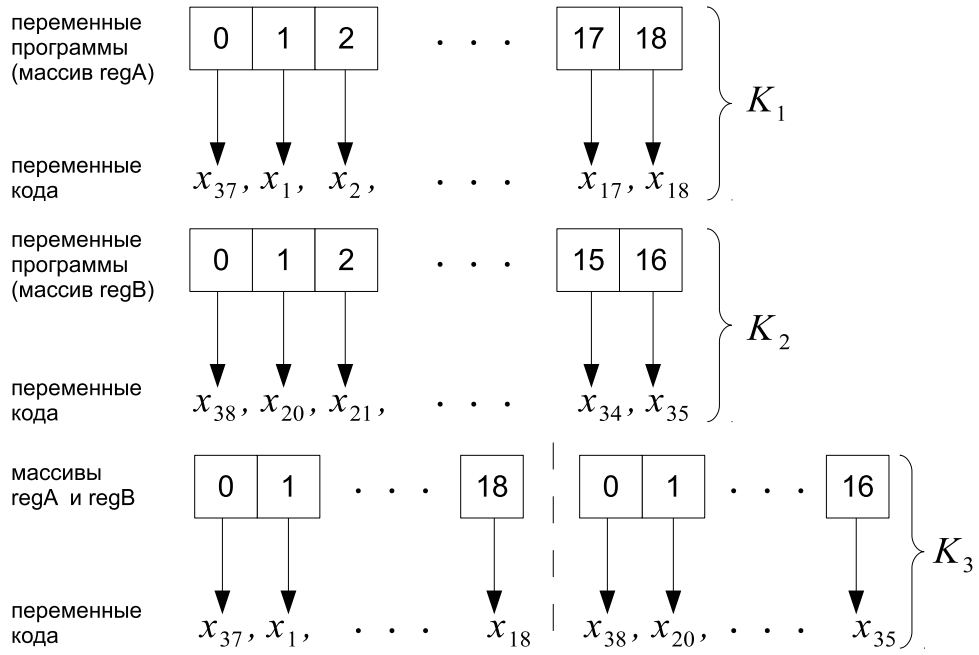


Рис. 2.6. Обновление связей между переменными программы и переменными кода в конфигурациях K_1 , K_2 и K_3 .

а в систему булевых уравнений будет добавлено уравнение

$$(x_{39} \equiv \varphi_1 \cdot x_{37} \vee \overline{\varphi_1} \cdot \varphi_2 \cdot x_1 \vee \overline{\varphi_1} \cdot \overline{\varphi_2} \cdot x_{37},) = 1.$$

Оператор цикла. При интерпретации оператора цикла символьное исполнение применяется только к телу цикла. Телом цикла может быть произвольный оператор языка ТА. Каждой итерации цикла сопоставляется некоторый набор конфигураций ТА-машины, формируемых в процессе интерпретации тела цикла. Число таких наборов конфигураций равно числу итераций цикла, которое известно заранее (в ТА-программе это некоторая целочисленная константа).

Оператор вызова функции и оператор возврата из функции. Интерпретация оператора вызова функции состоит из двух этапов. На первом этапе происходит символьная интерпретация всех аргументов функции. Затем выполняется переход в тело функции и осуществляется символьное исполнение ее операторов с использованием значений аргументов,

полученных на предыдущем этапе. После интерпретации всех операторов в теле функции интерпретатор возвращается в точку вызова этой функции. При этом возможна передача в данную точку некоторого логического выражения (результата символьной интерпретации функции).

2.2.3. Процедура декомпозиции сложных термов

В некоторых случаях требуется предоставить возможность исследователю явно связать выбранный терм над переменными кода с новой переменной кода и поместить его в словарь Σ . Например, подобная необходимость может возникнуть, если процедура преобразования добавляет в словарь термов слишком сложные выражения. В целях упрощения кода сложные булевы функции могут рассматриваться как композиции более простых, каждая из которых кодируется новой переменной кода.

Пример 2.6. Рассмотрим следующий фрагмент ТА-программы:

```
__in bit reg[10];
__in bit h;
__out bit output[5];
void shiftReg(){...}
void main(){
    ...
    //x, y, z - некоторые булевы переменные
    h = (x&y | x&z | y&z)^reg[0]^reg[1]^reg[8];
    ...
}
```

Обрабатывая инструкцию

```
h = (x&y | x&z | y&z)^reg[0]^reg[1]^reg[8];
```

интерпретатор помещает в словарь Σ терм

$$(x \cdot y \vee x \cdot z \vee y \cdot z) \oplus r_0 \oplus r_1 \oplus r_8.$$

Здесь x, y, z, r_0, r_1, r_8 — переменные кода, связанные с переменными программы $x, y, z, \text{reg}[0], \text{reg}[1], \text{reg}[8]$ на текущем шаге интерпретации.

Для упрощения сложных термов в языке ТА используется специальный атрибут `_mem`. С применением данного атрибута рассматриваемый фрагмент ТА-программы приобретает следующий вид:

```
__in bit reg[10];
__in bit h;
__out bit output[5];
void shiftReg(){...}
void main(){
    ...
    //x, y, z - некоторые булевы переменные
    __mem bit u = x&y | x&z | y&z;
    __mem bit v = reg[0]^reg[1]^reg[8];
    h = u ^ v;
    ...
}
```

В результате интерпретации данного фрагмента в словарь термов Σ добавляются термы

$$x \cdot y \vee x \cdot z \vee y \cdot z, r_0 \oplus r_1 \oplus r_8, u \oplus v.$$

При этом создаются новые переменные кода u, v, h , связанные с переменными программы u, v и h соответственно. Атрибут `_mem` можно использовать как для глобальных, так и для локальных переменных типа `bit`.

2.2.4. Преобразование операций над целыми числами

Большой практический интерес представляют дискретные функции, осуществляющие преобразования над целыми числами. В тексте ТА-программы целые числа представляются булевыми массивами. Однако в выражениях языка ТА допускаются операции над буквенными обозначениями чисел.

Преобразование логических операций. В языке ТА, так же, как и во многих языках программирования (см. [100]), допускаются побитовые логические операции над двоичными представлениями целых чисел. При символьной интерпретации в памяти ТА-машины целые числа представляются массивами, состоящими из булевых переменных (переменных кода). Операции $|$, $\&$, $\wedge$, $!$ над целыми числами при символьном исполнении рассматриваются как побитовые логические операции ИЛИ, И, исключающее ИЛИ и отрицание применительно к соответствующим массивам.

Пример 2.7. Рассмотрим следующую ТА-программу:

```
__in bit a[3];
__in bit b[3];
__out bit c[3];
void main(){
    c = a ^ b;
}
```

Массивы $\mathbf{a}$, $\mathbf{b}$ и $\mathbf{c}$ в памяти ТА-машины представлены словами (a_0, a_1, a_2) , (b_0, b_1, b_2) и (c_0, c_1, c_2) , где $a_i, b_i, c_i, i \in \{0, 1, 2\}$, — булевы переменные. Результатом символьного исполнения данной программы является следующая система булевых уравнений:

$$\begin{cases} (c_0 \equiv a_0 \oplus b_0) = 1 \\ (c_1 \equiv a_1 \oplus b_1) = 1 \\ (c_2 \equiv a_2 \oplus b_2) = 1. \end{cases}$$

Размеры массивов, к которым применяются логические операции, могут различаться. В этом случае перед выполнением операции массив, размер которого меньше, дополняется необходимым числом нулевых элементов (логических нулей).

Преобразование целочисленного сложения. Рассмотрим следующую ТА-программу (n — произвольная натуральная константа).

```

__in bit a[n];
__in bit b[n];
__out bit c[n+1];
void main(){
    c = a + b;
}

```

Данная программа реализует дискретную функцию, которая получает на вход два целых неотрицательных числа a , b и на выходе выдает их сумму — целое неотрицательное число c . Данные целые числа представлены в памяти ТА-машины массивами булевых переменных $a = \{a_0, \dots, a_{n-1}\}$, $b = \{b_0, \dots, b_{n-1}\}$ и $c = \{c_0, \dots, c_n\}$. Далее приведен результат символьной интерпретации данной программы в предположении, что для сложения двух неотрицательных целых чисел используется алгоритм «столбик».

$$\left\{ \begin{array}{l} (c_0 \equiv a_0 \oplus b_0) = 1 \\ (p_0 \equiv a_0 \cdot b_0) = 1 \\ (p_j \equiv \text{maj}(a_j, b_j, p_{j-1})) = 1, j = \overline{1, n-1} \\ (c_i \equiv a_i \oplus b_i \oplus p_{i-1}) = 1, i = \overline{1, n-1} \\ (c_n \equiv p_{n-1}) = 1. \end{array} \right. \quad (2.3)$$

Запись «maj(x, y, z)» обозначает терм $x \cdot y \vee x \cdot z \vee y \cdot z$. Для кодирования битов переноса вводятся новые переменные кода p_i , $i = \overline{0, n-1}$. Несложно заметить, что в процессе преобразования вводится $2n+1$ новых переменных кода.

Преобразование целочисленного умножения. Рассматривается дискретная функция, которая получает на вход два целых числа a , b и на выходе выдает их произведение — целое неотрицательное число c . Полагаем, что в памяти ТА-машины данные числа представлены массивами булевых переменных

$$a = \{a_0, \dots, a_n\}, b = \{b_0, \dots, b_m\}, c = \{c_0, \dots, c_{n+m}, c_{n+m+1}\}.$$

Также предположим, что для вычисления значения рассматриваемой функции используется процедура умножения «столбиком». На первом шаге символьной интерпретации ТА-программы, вычисляющей данную функцию, строится матрица

$$S = \|s_{ij}\|, s_{ij} = a_j \cdot b_i, i \in \{0, \dots, m\}, j \in \{0, \dots, n\}.$$

Строки матрицы S кодируют двоичные представления чисел, которые необходимо сложить столбиком.

Результатом символьного исполнения ТА-программы, реализующей процедуру умножения двух чисел, является система булевых уравнений, которую можно представить в виде конъюнкции m подсистем. Для каждого $i \in \{1, \dots, m\}$ соответствующая подсистема имеет следующий вид:

$$\left\{ \begin{array}{l} (c_k^i \equiv c_k^{i-1}) = 1, k = \overline{0, i-1} \\ (c_i^i \equiv c_i^{i-1} \oplus s_{i0}) = 1 \\ (c_j^i \equiv c_j^{i-1} \oplus s_{i(j-i)} \oplus p_{j-i}^i) = 1, j = \overline{i+1, n+i-1} \\ (c_{n+i}^i \equiv s_{in} \oplus p_n^i) = 1 \\ (c_{n+i+1}^i \equiv s_{in} \cdot p_n^i) = 1. \end{array} \right. \quad (2.4)$$

Здесь c^0 — это первая строка матрицы S , т. е.

$$c^0 = (a_0 \cdot b_0, a_1 \cdot b_0, \dots, a_n \cdot b_0).$$

Через c^i , где $i \in \{1, \dots, m\}$, обозначим вектор, состоящий из булевых переменных, которые находятся в левых частях логических эквивалентностей уравнений системы (2.4). Тогда компоненты вектора c^m — это элементы массива s . Биты переноса p_j^i определяются по тем же правилам, что и для описанной ранее операции целочисленного сложения.

Преобразование операций сдвига. Операндами операций сдвига являются целые числа. Левый операнд — это целое неотрицательное чис-

ло, к которому применяется сдвиг; правый операнд — целое неотрицательное число, определяющее количество позиций, на которые выполняется сдвиг. Символ операции задает направление и характер сдвига двоичного представления целого числа. В таблице 2.3 рассмотрено действие различных операций побитового сдвига на целое число a , закодированное словом $(a_1, \dots, a_n)$, где $a_i, i \in \{1, \dots, n\}$, — булевы переменные. При этом предполагается, что m — некоторое целое неотрицательное число, причем $m < n$.

Таблица 2.3. Операции побитового сдвига в языке ТА.

Выражение	Результат	Примечание
$a \ll m$	$(a_{m+1} \dots a_n \underbrace{0 \dots 0}_m)$	Сдвиг влево с обнулением освободившихся позиций.
$a \gg m$	$(\underbrace{0 \dots 0}_m a_1 \dots a_{n-m})$	Сдвиг вправо с обнулением освободившихся позиций.
$a \lll m$	$(a_{m+1} \dots a_n a_1 \dots a_m)$	Циклический сдвиг влево.
$a \ggg m$	$(a_{n-m+1} \dots a_n a_1 \dots a_{n-m})$	Циклический сдвиг вправо.
$a \ll *m$	$(a_{m+1} \dots a_n \underbrace{a_n \dots a_n}_m)$	Сдвиг влево с «залипанием» битов.
$a \gg *m$	$(\underbrace{a_1 \dots a_1}_m a_1 \dots a_{n-m})$	Сдвиг вправо с «залипанием» битов.

На практике возможны случаи, когда $m \geq n$. Тогда при сдвиге с обнулением все компоненты слова $(a_1, \dots, a_n)$ становятся нулями; при сдвиге вправо (влево) с залипанием все элементы данного слова заменяются на a_1 (a_n). Очевидно, что циклический сдвиг на m позиций равносителен циклическому сдвигу на остаток от деления m на n .

Преобразование операций сравнения. В языке ТА предусмотрен механизм преобразования в булевы уравнения выражений следующего ви-

да:

$$a > b, a \geq b, a < b, a \leq b, a == b, a \neq b,$$

где a и b — целые неотрицательные числа.

Далее рассматривается процесс интерпретации выражения $a > b$. В памяти ТА-машины числа a и b представлены массивами $a = \{a_1, \dots, a_n\}$ и $b = \{b_1, \dots, b_m\}$, которые состоят из булевых переменных. С данными массивами интерпретатор связывает новую переменную кода c . В предположении, что $n = m$ в словарь термов Σ будет добавлен терм

$$a_m \cdot \bar{b}_m \vee (a_m \equiv b_m) \cdot a_{m-1} \cdot \bar{b}_{m-1} \vee \dots \vee (a_m \equiv b_m) \cdot \dots \cdot (a_2 \equiv b_2) \cdot a_1 \cdot \bar{b}_1, \quad (2.5)$$

а в систему булевых уравнений будет добавлено уравнение

$$(c \equiv (a_m \cdot \bar{b}_m \vee (a_m \equiv b_m) \cdot a_{m-1} \cdot \bar{b}_{m-1} \vee \dots \vee (a_m \equiv b_m) \cdot \dots \cdot (a_2 \equiv b_2) \cdot a_1 \cdot \bar{b}_1)) = 1.$$

Если размеры булевых массивов a и b не равны, например $n > m$, то в результате интерпретации выражения $a > b$ в Σ добавляется терм

$$\Gamma \vee a_n \vee a_{n-1} \vee \dots \vee a_{m+1},$$

а в систему булевых уравнений добавляется уравнение

$$(c \equiv \Gamma \vee a_n \vee a_{n-1} \vee \dots \vee a_{m+1}) = 1,$$

где Γ — это терм (2.5).

Аналогичным образом интерпретируются остальные операции сравнения целых чисел.

2.3. Программный комплекс Transalg для преобразования алгоритмов вычисления дискретных функций в булевы уравнения

Описанный выше подход к преобразованию ТА-программ в булевы уравнения через их символьную интерпретацию был реализован в виде программного комплекса Transalg [25]. Данный комплекс предоставляет возможность преобразования в булевы уравнения произвольных алгоритмов, вычисляющих всюду определенные дискретные функции, в том числе алгоритмов, содержащих условные переходы. Дополнительные возможности комплекса Transalg позволяют строить тестовые примеры для решателей булевых уравнений, используя пропозициональный код алгоритма в качестве шаблона. При этом предусмотрена возможность добавления к пропозициональному коду алгоритма ограничений, кодирующих дополнительные условия.

Программный комплекс Transalg написан на языке C++, с помощью инструментальной среды разработки Visual Studio 2008 Express Edition. Комплекс представляет собой Win32-приложение с интерфейсом командной строки. Тем не менее, программный код всех разработанных алгоритмов является платформо-независимым и при желании может быть скомпилирован под любую операционную систему.

На вход комплексу Transalg подается файл с расширением alg, содержащий текст программы на языке ТА, которая вычисляет дискретную функцию. Текст ТА-программы обрабатывается модулем синтаксического разбора. Результатом работы данного модуля является дерево синтаксического разбора — внутреннее представление программы, вычисляющей дискретную функцию. Модуль преобразования осуществляет обход дерева



Рис. 2.7. Архитектура программного комплекса Transalg.

синтаксического разбора, при этом в соответствии с семантикой операторов языка ТА (в смысле символьной интерпретации) строятся булевы уравнения. Фазы анализа текста ТА-программы, построения дерева синтаксического разбора и обхода полученного дерева реализованы стандартным способом (см., например [97, 99, 101, 102]).

Нетривиальным и наиболее трудоемким этапом является процедура построения булевых уравнений в соответствии с семантикой символьного исполнения конструкций языка ТА. На этом этапе возникают многочисленные локальные проблемы, от решения которых существенно зависит как объем пропозиционального кода (в смысле общего числа задействованных в

нем булевых переменных и числа уравнений), так и его структура. Все описанные выше алгоритмы и структуры данных, обеспечивающие символьное исполнение ТА-программ, реализованы в виде библиотеки семантических подпрограмм. В данной библиотеке присутствуют подпрограммы для интерпретации каждого оператора и операции языка ТА. Символьное исполнение ТА-программы включает в себя обход дерева синтаксического разбора, идентификацию конструкций языка ТА и вызовы соответствующих семантических подпрограмм, которые при помощи структур `var_object` используют информацию о переменных ТА-программы и взаимодействуют со словарем термов. Результатом работы семантических подпрограмм является некоторый фрагмент пропозиционального кода рассматриваемого алгоритма, то есть система булевых уравнений.

В комплексе Transalg процесс вывода итогового пропозиционального кода разбит на этапы. На первом этапе по описанию алгоритма на языке ТА строится система булевых уравнений во внутреннем формате Transalg-a (т. н. «LS-формат»). По смыслу такие системы аналогичны системам $S(f_n)$, рассмотренным в первой главе. От систем в LS-формате осуществляется переход к различным нормальным формам при помощи преобразований Цейтина (см. п. 1.2.2, [50, 51]). Известные примеры применения преобразований Цейтина (см. [12]) предполагают ввод новой переменной для кодирования каждой элементарной операции. Такой подход далеко не всегда оптимален, поскольку приводит к чрезмерному разрастанию множества дополнительных переменных. В комплексе Transalg преобразования Цейтина осуществляются в отношении подформул от многих переменных (до 15 переменных). При этом КНФ-представление булевой функции вида $y \equiv f(x_1, \dots, x_k)$ строится в два этапа: сначала строится СКНФ по таблице истинности, а затем данная СКНФ минимизируется при помощи свободно распространяемой утилиты Espresso [103], которая встроена в Transalg в

виде отдельного модуля. Вычислительные эксперименты показали, что использование Espresso позволяет сократить общий объем пропозиционального кода (в КНФ) в среднем на 30 процентов. Итоговая КНФ сохраняется в текстовом файле в формате DIMACS [104].

В последние годы интенсивно развиваются методы решения алгебраических уравнений [66], базирующиеся на алгоритме Бухбергера [65]. В связи с этим в комплексе Transalg предусмотрена возможность построения пропозициональных кодов алгоритмов в форме систем полиномиальных уравнений над полем $GF(2)$.

В некоторых областях исследований (главным образом, при проектировании и верификации СБИС) дискретные функции представляются схемами из функциональных элементов над произвольным полным базисом. В последнее время для этой цели используется базис $\{\&, \neg\}$, а соответствующие структуры называются «И-НЕ» графами (And-Inverter Graphs). Одним из основных аргументов в пользу применения данного способа записи пропозиционального кода алгоритма является тот факт, что графы, в отличие от КНФ и полиномов, сохраняют информацию о структуре рассматриваемой дискретной функции. Использование этих дополнительных знаний может существенно увеличить эффективность алгоритмов логического поиска. Для записи «И-НЕ» графа используется стандартный формат AIGER [105]. В комплексе Transalg AIGER является одним из возможных выходных форматов.

Дополнительно комплекс Transalg позволяет работать с некоторыми задачами дискретной оптимизации. Постановки этих задач являются не процедурными и описываются системами ограничений (уравнений и неравенств над множеством целых чисел). На данный момент комплекс Transalg поддерживает работу с задачами 0-1-целочисленного линейного программирования (0-1-ЦЛП, см. раздел 3.3.1), а также с квадратичной задачей о

назначениях (QAP, см. раздел 3.3.2). Для каждой из этих задач реализован отдельный модуль, осуществляющий преобразование ограничений задачи в исходной постановке в соответствующую систему булевых уравнений. При этом используются базовые алгоритмы и структуры данных комплекса Transalg.

Ниже перечислены форматы пропозиционального кода алгоритмов, поддерживаемые в комплексе Transalg:

1. LS-формат: система булевых уравнений в общем виде. Каждое уравнение занимает отдельную строку файла и представлено в инфиксной записи (например, $x_1 \& x_2 \& x_3 = 1$);
2. DIMACS-формат для описания КНФ (ДНФ) [104];
3. DIMACS-подобный формат для описания систем полиномиальных уравнений над полем $GF(2)$;
4. формат AIGER [105] для представления И-НЕ графов.

Основные выводы по второй главе. В данной главе представлен проблемно-ориентированный язык (язык ТА), предназначенный для описания алгоритмов вычисления дискретных функций. Процесс интерпретации ТА-программы представляет собой форму символьного исполнения, результатом которого является система булевых уравнений. Разработаны и реализованы алгоритмы и структуры данных, используемые при преобразовании ТА-программ в булевы уравнения. Разработаны механизмы оптимизации пропозиционального кода (специальный словарь термов, процедура декомпозиции сложных термов, техники, комбинирующие преобразования Цейтина с минимизацией булевых функций). Все разработанные алгоритмы и структуры данных объединены в программный комплекс Transalg.

Глава 3

Результаты преобразования в булевы уравнения алгоритмов вычисления некоторых дискретных функций

В данной главе приведены результаты использования программного комплекса Transalg, который был разработан в соответствии с описанными выше принципами. Данный комплекс применялся для преобразования в булевы уравнения некоторых криптографических алгоритмов. При этом преследовались цели получения аргументировано трудных тестов для решателей систем булевых уравнений. Также с применением программного комплекса Transalg были построены булевы уравнения (вида КНФ=1), кодирующие задачи поиска циклов автоматных отображений, моделирующих поведение одного класса генных сетей. В заключительной части главы комплекс Transalg используется для сведения к SAT-задачам некоторых задач дискретной оптимизации — 0-1-ЦЛП (0-1-целочисленное линейное программирование) и QAP (Quadratic Assignment Problem).

3.1. Преобразование в булевы уравнения некоторых криптографических функций

Одной из наиболее наглядных областей применения развитой техники преобразования процедурных описаний алгоритмов является криптография. В статье [106] было отмечено, что пропозициональные коды алгоритмов шифрования можно использовать для построения аргументированно трудных тестов для разнообразных решателей комбинаторных задач

(в том числе для SAT-решателей). В дальнейшем криптоанализ, рассматриваемый как процесс поиска решений булевых уравнений (в частности, SAT-задач), стали называть логическим криптоанализом [3]. Логический криптоанализ оказался эффективным в применении к некоторым генераторам ключевого потока [71, 107, 108].

Генераторы поточного шифрования — это быстро вычисляемые дискретные функции, преобразующие двоичные последовательности конечной длины (инициализирующие последовательности) в бесконечные периодические двоичные последовательности (ключевой поток). Задача криптоанализа генератора заключается в нахождении инициализирующей последовательности по известному фрагменту ключевого потока и алгоритму функционирования генератора. Обработывая данный алгоритм, записанный на языке ТА, Transalg строит систему булевых уравнений, кодирующих процесс порождения произвольного фрагмента ключевого потока. Подстановка в полученную систему анализируемого фрагмента ключевого потока дает систему, из решения которой можно эффективно выделить искомый секретный ключ (инициализирующую последовательность). Наиболее значимым на текущий момент практическим результатом в этом направлении является успешный логический криптоанализ генератора поточного шифрования A5/1, реализованный в специально построенной для этой цели Grid-среде [108].

3.1.1. Преобразование в булевы уравнения алгоритма генератора ключевого потока шифра A5/1

Схема работы генератора поточного шифрования A5/1 была описана во втором примере первой главы. Рассмотрим программу на языке ТА, вычисляющую фрагмент ключевого потока данного генератора. Начальное заполнение регистров сдвига является неизвестным секретным ключом. Бу-

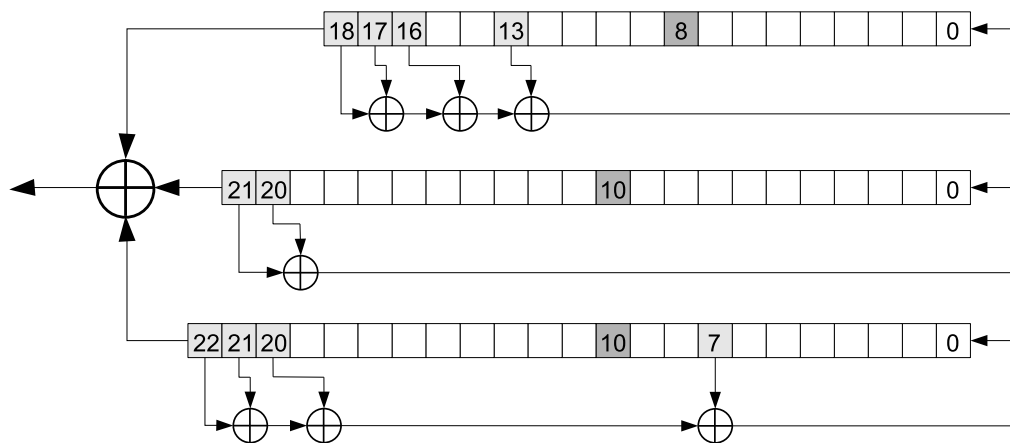


Рис. 3.1. Схема генератора ключевого потока A5/1.

левые переменные, кодирующие секретный ключ, образуют множество X^0 . В тексте ТА-программы присутствуют 64 переменные, идентифицирующие области памяти, хранящие содержимое ячеек РСЛОС. Эти переменные образуют массивы `regA`, `regB` и `regC`. Смысл следующего фрагмента ТА-программы состоит в том, что на шаге инициализации с элементами массивов `regA`, `regB` и `regC` связываются посредством структур `var_object` переменные кода, образующие множество X^0 :

```
__in bit regA[19];
__in bit regB[22];
__in bit regC[23];
```

Процедура сдвига первого РСЛОС выглядит следующим образом:

```
void shiftRegA(){
    bit y = regA[18]^regA[17]^regA[16]^regA[13];
    for(int j = 18; j > 0; j = j - 1)
        regA[j] = regA[j - 1];
    regA[0] = y;
}
```

Для каждого РСЛОС создается аналогичная процедура. Основная функция `main`, реализующая алгоритм генератора A5/1, имеет следующий вид:

```
void main(){
    int midA = 8;
```

```

int midB = 10;
int midC = 10;
bit maj;
for(int i = 0; i < 100; i = i + 1){
    //вычисляем функцию большинства
    maj = majority(regA[midA],regB[midB],regC[midC]);
    if(!(maj^regA[midA])) shiftRegA();
    if(!(maj^regB[midB])) shiftRegB();
    if(!(maj^regC[midC])) shiftRegC();
    result[i] = regA[18]^regB[21]^regC[22];
}
}

```

Результатом одной итерации цикла **for** является один бит ключевого потока, который записывается в массив **result**. Массив переменных типа **bit**, предназначенный для сохранения результата работы программы, должен быть заранее объявлен с атрибутом **_out** в глобальной области видимости. Например (для 100 тактов):

```
__out bit result[100];
```

Таблица 3.1. Результаты пропозиционального кодирования алгоритма А5/1.

Длина ключевого потока	Число переменных	Число дизъюнктов
128	8768	39936
160	10944	49920
192	13120	59904
224	15296	69888
256	17472	79872
384	26176	119808
512	34880	159744

Средствами комплекса Transalg были получены системы булевых уравнений, кодирующие алгоритм А5/1 для различных значений параметра

«длина ключевого потока». Системы уравнений с помощью преобразований Цейтина были приведены к виду $\text{КНФ}=1$. Характеристики полученных КНФ представлены в таблице 3.1.

3.1.2. Преобразование в булевы уравнения алгоритма суммирующего генератора

Впервые суммирующий генератор был описан в работе Р. Рюппеля [109] (см. также [45]). Данный генератор состоит из $R \geq 2$ РСЛОС и специального регистра, называемого сумматором. Содержимое регистров сдвига и сумматора в начальный момент времени образует секретный ключ.

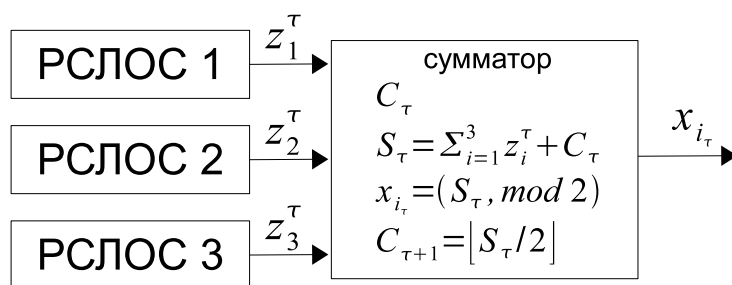


Рис. 3.2. Схема работы суммирующего генератора.

Сумматор представляет собой одномерный массив ячеек памяти размерности $\lceil \log R \rceil$. Отдельная ячейка позволяет хранить один бит информации. В начальный момент времени ($\tau = 0$) в сумматоре записан вектор двоичного представления натурального числа C_0 . В каждый такт с номером τ , $\tau \in \{1, 2, \dots\}$, синхронно снимаемые с выходов РСЛОС биты подаются на вход сумматора, где целочисленно складываются с $C_{\tau-1}$. Полученное натуральное число обозначается через S_τ . Выходом генератора суммирования в такте с номером τ , $\tau \in \{1, 2, \dots\}$, является младший бит двоичного представления числа S_τ . Значением сумматора в такте с номером τ , $\tau \in \{1, 2, \dots\}$, является двоичное представление числа $\lfloor \frac{S_\tau}{2} \rfloor$.

Рассмотрим задачу пропозиционального кодирования генератора суммирования, состоящего из трех РСЛОС ($R = 3$), функции обратной связи которых определяются следующими полиномами над $\text{GF}(2)$ (Z — формальная переменная):

$$P_1(Z) = Z^{19} + Z^{18} + Z^{17} + Z^{14} + 1$$

$$P_2(Z) = Z^{22} + Z^{21} + 1$$

$$P_3(Z) = Z^{23} + Z^{22} + Z^{21} + Z^8 + 1$$

ТА-программа, вычисляющая 180 бит ключевого потока для данного суммирующего генератора, выглядит следующим образом:

```
__in bit regA[19];
__in bit regB[22];
__in bit regC[23];
__in bit summator[2];
__out bit output[180];
void main(){
    for(int i = 0; i < 180; i = i + 1){
        summator = shiftRegA() + shiftRegB() + shiftRegC() + summator;
        output[i] = summator[0];
        summator = summator >> 1;
    }
}
```

Здесь `shiftRegA`, `shiftRegB` и `shiftRegC` — процедуры, вычисляющие функции обратной связи для РСЛОС (см. пример 2.3). Объявления массивов переменных с атрибутами `_in` и `_out` позволяют процедуре преобразования создать множество переменных кода $X^0 = \{x_1, \dots, x_{66}\}$, кодирующих вход алгоритма, и множество переменных $X^e = \{x_{i_1}, \dots, x_{i_{180}}\}$, кодирующих выход алгоритма.

Пропозициональное кодирование процедуры сдвига регистров осуществляется по схеме, описанной ранее во второй главе (см. пример 2.4). Таким

образом, результатом преобразования первого шага работы суммирующего генератора является следующая система булевых уравнений:

$$\left\{ \begin{array}{l} (y_1 \equiv x_{19} \oplus x_{18} \oplus x_{17} \oplus x_{14}) = 1 \\ (y_2 \equiv x_{41} \oplus x_{40}) = 1 \\ (y_3 \equiv x_{64} \oplus x_{63} \oplus x_{62} \oplus x_{49}) = 1 \\ (y_4 \equiv x_{19} \oplus x_{41} \oplus x_{64} \oplus x_{65}) = 1 \\ (y_5 \equiv x_{19} \cdot x_{41} \oplus x_{66} \oplus x_{64} \cdot x_{65} \oplus (x_{19} \oplus x_{41}) \cdot (x_{64} \oplus x_{65})) = 1 \end{array} \right.$$

Здесь переменные y_1, y_2, y_3 — это новые переменные кода, связанные через структуру `var_object` с переменными программы `regA[0]`, `regB[0]` и `regC[0]`, а переменные y_4, y_5 кодируют соответственно младший и старший биты натурального числа, которое является новым значением сумматора. Тем самым переменная y_4 кодирует первый бит ключевого потока. Таким образом, кодирование с помощью комплекса Transalg одного такта работы рассматриваемого суммирующего генератора добавляет в кодировку 5 новых булевых уравнений и 5 дополнительных переменных.

Результатом преобразования ТА-программы, реализующей 180 тактов работы данного генератора, является система из 900 булевых уравнений от 966 переменных. Итоговая КНФ состоит из 11532 дизъюнктов над множеством из 966 булевых переменных.

3.1.3. Преобразование в булевы уравнения алгоритма генератора Гиффорда

В 1984 году группой под руководством Д. Гиффорда был разработан генератор поточного шифрования, названный в последствие генератором Гиффорда [110]. Поточковый шифр на базе данного генератора использовался с 1984 по 1988 год для передачи текстовой информации агентством Associated Press [111]. Первая успешная атака на генератор Гиффорда опи-

сана в статье [112], где был представлен специально разработанный для криптоанализа данного генератора весьма громоздкий математический аппарат. Отличительной особенностью генератора Гиффорда является то, что в нем не используются РСЛОС.

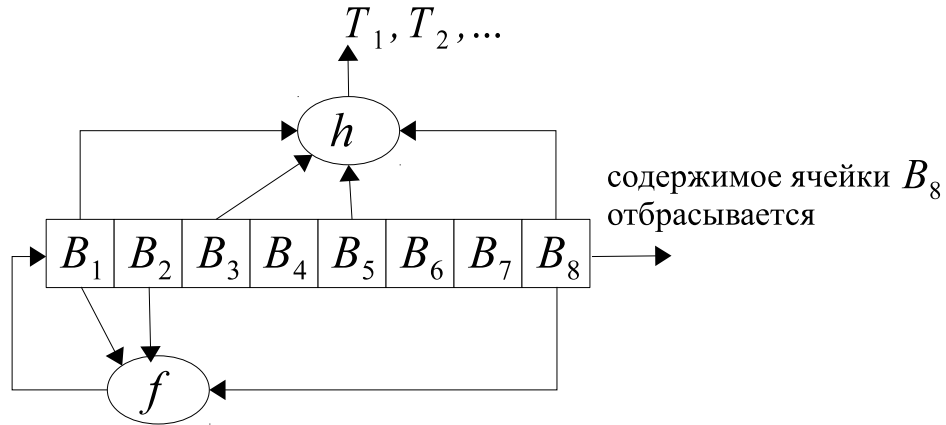


Рис. 3.3. Схема работы генератора Гиффорда.

Рабочая область (регистр) генератора Гиффорда состоит из восьми ячеек, каждая из которых хранит 8 бит (1 байт) информации. В начальный момент времени $\tau = 0$ в ячейках $B_1, B_2, \dots, B_8$ записаны байты $b_1^0, \dots, b_8^0$ инициализирующей последовательности (общая длина 64 бита). Ключевой поток формируется из последовательности байт $T_1, T_2, \dots$, выдаваемых генератором в моменты времени $\tau, \tau \in \{1, 2, \dots\}$. Для получения байта ключевого потока применяется фильтрующая нелинейная функция $h : \{0, 1\}^{32} \rightarrow \{0, 1\}^8$:

$$h(B_1, B_3, B_5, B_8) = \text{Extract_Byte}_3((B_1 || B_3) \times (B_5 || B_8)).$$

Здесь $||$ — конкатенация содержимого соответствующих ячеек, а $\times$ — целочисленное умножение. Аргументом функции Extract_Byte_3 является целое неотрицательное число, а значением — третий байт двоичного представления данного числа. На рисунке 3.4 изображена схема вычисления функции h .

Обновление регистра выполняется на каждом шаге сразу после вычисления байта ключевого потока. Сначала вычисляется функция обратной связи

$$b_1^{\tau+1} = f(b_1^\tau, b_2^\tau, b_8^\tau) = b_1^\tau \oplus (>> *(b_2^\tau)) \oplus (<< (b_8^\tau)),$$

где τ — номер шага. Операции $>> *$ и $<<$ обозначают соответственно операции сдвига вправо на один бит с сохранением старшего бита (так называемый sticky right-shift) и сдвига влево на один бит с обнулением младшего бита. Например, для $B(x_1, \dots, x_8)$ имеем:

$$>> *(B) = (x_1, x_1, x_2, \dots, x_7), \quad << B = (x_2, x_3, \dots, x_8, 0).$$

Далее текущее заполнение регистра сдвигается на один байт вправо. При этом содержимое ячейки B_8 отбрасывается, а в ячейку B_1 записывается байт $b_1^{\tau+1}$.

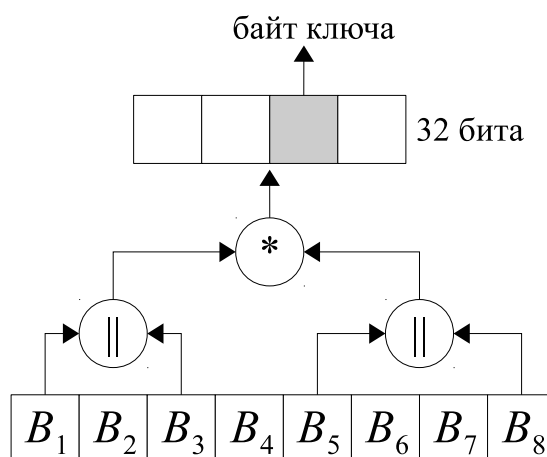


Рис. 3.4. Схема нелинейной функции h в генераторе Гиффорда.

Ниже приведена ТА-программа, описывающая процесс вычисления 20 байт ключевого потока генератором Гиффорда. Массив переменных программы **reg** идентифицирует ячейки регистра генератора. При этом внешний индекс массива **reg** означает номер байта в регистре, а внутренний индекс — номер бита в соответствующем байте. Множество X^0 составляют 64 переменных кода. Переменные программы, образующие массив **output**,

связываются с переменными кода из множества X^e . Массивы `mult` и `x` являются вспомогательными структурами, необходимыми для корректной организации вычислений. Функция `main` реализует алгоритм генератора Гиффорда, вычисляющего 20 байт ключевого потока. При этом каждый шаг работы генератора является результатом выполнения трех инструкций ТА-программы. Процедура `shiftReg` реализует вычисление функции обратной связи $f(b_1^T, b_2^T, b_8^T)$ и обновление регистра генератора. Функция `mul(a, b, n)` отвечает за формирования пропозиционального кода для первых n бит произведения целых неотрицательных чисел a и b . Массив фиктивных переменных программы `mult` сохраняет результат работы функции `mul`. Последняя команда извлекает из массива `mult` 3-й байт и посредством булевых уравнений связывает термы, кодирующие данный байт, с переменными кода из множества X^e .

```
define len 20
__in bit reg[8][BYTE];          //8-байтный регистр
__out bit output[len][BYTE];    //20 байт выхода
bit mult[3][BYTE];
bit x[BYTE];
void shiftReg(){
    x = reg[0]^(reg[1] >>* 1)^(reg[7] << 1);
    for(int j = 7; j > 0; j = j - 1)
        reg[j] = reg[j-1];
    reg[0] = x;
}
void main(){
    for(int i = 0; i < len; i = i + 1){
        shiftReg();
        mult = mul((reg[0]||reg[2]), (reg[4]||reg[7]), 24);
        output[i] = mult[2]; //3-й байт
    }
}
```

Здесь `len`, `BYTE` — именованные целочисленные константы со значениями 20 и 8 соответственно.

Итогом обработки данной ТА-программы комплексом Transalg, с выполнением всех рассмотренных ранее преобразований, является булево уравнение вида КНФ=1. Полученная КНФ состоит из 49080 дизъюнктов над множеством из 4884 булевых переменных.

3.1.4. Преобразование в булевы уравнения алгоритма DES

Криптоалгоритм Data Encryption Standard (DES) в 1977 году был принят в качестве стандарта шифрования данных в США и широко использовался в криптографических системах вплоть до конца прошлого века. Несмотря на то, что данный алгоритм считается устаревшим и в 2002 году был заменен на более современный AES, интерес специалистов к нему не иссяк. Алгоритм DES долгие годы был стимулом для развития различных методов криптоанализа. В частности, именно задача криптоанализа DES привела Э. Бихама и А. Шамира в 1995 г. к формулировке основных положений дифференциального криптоанализа [113]. На сегодняшний день единственным практически реализованным методом криптоанализа DES является простой перебор (как правило, с применением специализированных вычислительных архитектур [114]). В связи с этим алгоритм DES интересен для тестирования эффективности современных решателей булевых уравнений. Впервые пропозициональный код алгоритма DES был представлен в работе [4]. Журнальный вариант данной статьи появился годом позже [3]. В данном разделе приводится результат пропозиционального кодирования алгоритма DES при помощи программного комплекса Transalg. Полученный код в КНФ существенно экономнее кода, представленного в [3].

Алгоритм DES представляет собой симметричный блочный алгоритм шифрования, построенный на основе сети Фейстеля ([45, 46]). Открытый

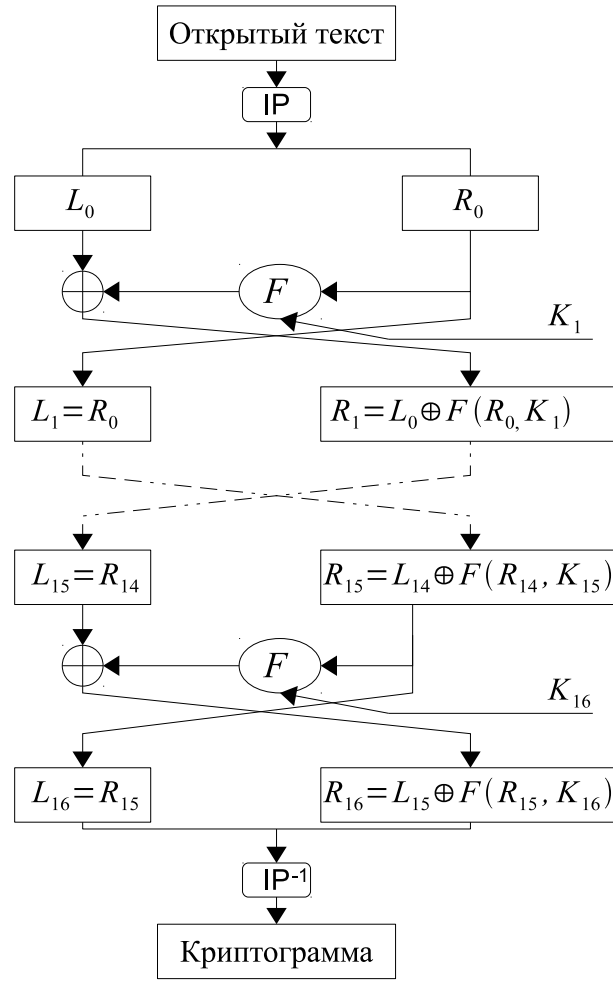


Рис. 3.5. Схема алгоритма DES.

текст обрабатывается блоками длины 64 бита. Длина ключа составляет 56 бит. На начальном шаге над блоком открытого текста выполняется специальная перестановка (т. н. начальная перестановка), после чего этот блок делится на левую L_0 и правую R_0 части, длиной по 32 бита каждая. Процесс шифрования в алгоритме DES состоит из 16 раундов, на каждом из которых выполняются одностипные преобразования над блоком данных, определяемые нелинейной функцией F (рисунок 3.5). Обозначим через K_0 исходный секретный ключ. Для каждого раунда формируется отдельный ключ K_i , $i \in \{1, \dots, 16\}$. Для этого ключ $K_{i-1} = \{k_1, k_2, \dots, k_{56}\}$ делится на две части $K_{i-1}^l = \{k_1, \dots, k_{28}\}$ и $K_{i-1}^r = \{k_{29}, \dots, k_{56}\}$, к каждой из которых применяется циклический сдвиг влево на 1 или 2 бита (в зависимости от

раунда). Конкатенация полученных векторов дает ключ K_i .

Аргументами функции Фейстеля F являются 32-битный блок данных R и 56-битный ключ K . Из ключа K с помощью таблицы перестановки выбирается 48 бит (т. н. «перестановка со сжатием») — «подключ» K' . Применение к блоку данных R еще одной перестановки дает блок из 48 бит (т. н. «перестановка с расширением»), который складывается по модулю 2 с K' . Полученный 48-битный вектор подается на вход S-блокам, в которых выполняется подстановка. К выходу S-блоков (32-бита) применяется перестановка в P-блоке. Результирующий 32-битный блок данных является значением функции F .

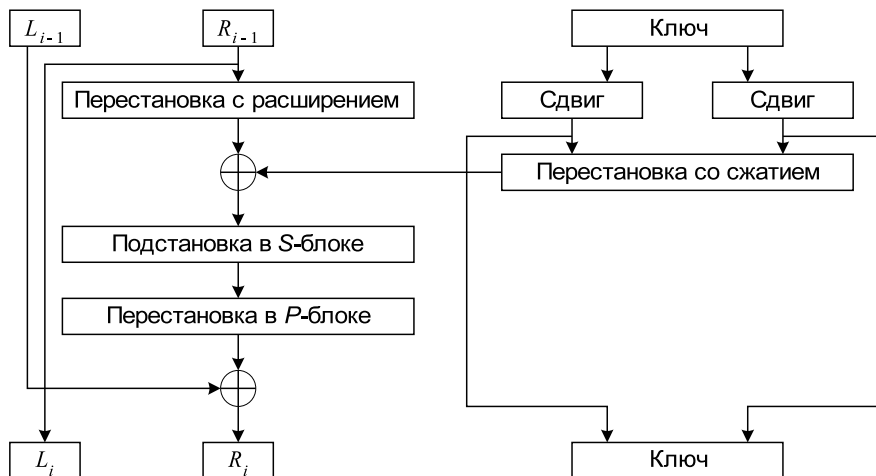


Рис. 3.6. Один раунд алгоритма DES.

На i -том раунде функции F передается блок данных R_{i-1} и ключ K_i , $i \in \{1, \dots, 16\}$. Вычисленное значение функции F скалывается по модулю 2 с блоком данных L_{i-1} , и результат сложения образует 32-битный блок данных R_i . На месте блока L_i далее используется блок R_{i-1} . Таким образом, на каждом этапе 32-битные блоки данных R_i и L_i вычисляются по формулам

$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i), \quad L_i = R_{i-1}, \quad i \in \{1, \dots, 16\}.$$

К конкатенации блоков L_{16} и R_{16} применяется заключительная перестанов-

ка, которая дает 64-битный блок шифртекста.

Построим ТА-программу, осуществляющую вычисления по описанному выше алгоритму (полный текст программы приводится в Приложении В). Неизвестными входными данными программы являются 64 бита открытого текста и 56 бит ключа. Результат вычисления представляют 64 бита шифртекста. Таким образом, ТА-программа начинается с объявления соответствующих массивов переменных программы:

```
__in bit data[64];  
__in bit key[2][28];  
__out bit code[64];
```

Все перестановки, фигурирующие в алгоритме DES, заданы таблицами (массивами) натуральных чисел. Целое число j в i -й позиции такого массива означает, что элемент, находящийся в позиции с номером j в двоичном векторе, к которому применяется данная перестановка, следует поместить в позицию с номером i . Например, перестановка в Р-блоке определяется следующей таблицей:

16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

Например, число 16 в первой позиции означает, что Р-блок перемещает 16-й элемент двоичного вектора в позицию с номером 1. В ТА-программе для представления перестановок используются целочисленные массивы:

```
int PBox[32] = {16, 7, ..., 25}; //перестановка в Р-блоке
```

Блоком подстановки (S-блоком) называется дискретная функция вида $f_s : \{0, 1\}^6 \rightarrow \{0, 1\}^4$. В алгоритме DES S-блок задается таблицей натуральных чисел, состоящей из 4 строк и 16 столбцов. Пусть на вход функции f_s поступает двоичный вектор $x = \{x_1, \dots, x_6\}$. На его основе определяются числа m и n : вектор $\{x_1, x_6\}$ интерпретируется как двоичная запись числа m , а вектор $\{x_2, x_3, x_4, x_5\}$ — как двоичная запись числа n . Выходом функции является двоичная запись натурального числа, расположенного на пересечении строки с номером m и столбца с номером n таблицы S-блока.

В ТА-программе S-блок записывается в виде двумерного целочисленного массива:

```
int SBox[4][16] = {{...},{...},{...},{...}}; //S-блок
```

В язык ТА добавлена специальная процедура **MakeSBox**, реализующая подстановку в S-блоке. Процедура **MakeSBox** имеет следующий синтаксис:

```
MakeSBox(SBox, input, output);
```

Здесь **SBox** — имя двумерного массива, описывающего S-блок, **input** — имя массива из 6 булевых переменных, кодирующих входные данные S-блока, **output** — имя массива из 4 булевых переменных, кодирующих выход S-блока.

Новый примитив ТА-языка (процедура **MakeSBox**) позволяет как упростить написание ТА-программ, выполняющих подстановки в S-блоках, так и оптимизировать пропозициональный код блоков подстановки. Далее описан механизм кодирования S-блоков. Работа процедуры **MakeSBox** начинается с построения таблицы истинности для дискретной функции f_s , заданной рассматриваемым S-блоком (таблица 3.2).

Таблица 3.2. Фрагмент таблицы истинности функции f_s , реализуемой 1-м S-блоком.

in1	in2	in3	in4	in5	in6	out1	out2	out3	out4
0	0	0	0	0	0	1	1	1	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	1	0	0
...	...	...	...	...	...	...	...	...	...
1	1	1	1	1	1	1	1	0	1

Функция f_s при этом рассматривается как набор из 4 булевых функций f_i , $i = \overline{1, 4}$. Таблица истинности для функции f_i состоит из колонок (in1, in2, in3, in4, in5, in6, out*i*), $i = \overline{1, 4}$. Обозначим через $x_1, \dots, x_6$ булевы

переменные, кодирующие вход S-блока, а через y_1, y_2, y_3, y_4 — булевы переменные, кодирующие выход S-блока. По таблицам истинности функций f_i над множеством переменных $\{x_1, \dots, x_6, y_i\}$ строятся реализующие их КНФ, которые минимизируются при помощи утилиты Espresso [103]. Процедура `MakeSBox` добавляет в пропозициональный код алгоритма 4 новых КНФ, представляющих булевы функции, заданные формулами

$$y_i \equiv f_i(x_1, \dots, x_6), i \in \{1, 2, 3, 4\}.$$

Далее приведена процедура `main`, которая осуществляет шифрование 1 блока открытого текста. Перед входом в основной цикл, реализующий шифрование, объявляются массивы переменных программы, предназначенные для работы с промежуточными результатами вычислений. Далее в цикле выполняются 16 раундов алгоритма DES. Через `compression_replace` — обозначен массив целых чисел, задающий перестановку со сжатием, через `exp_replace` — обозначен массив, задающий перестановку с расширением.

```
void main(){
    //переменные для результатов промежуточных вычислений
    bit lData[32];    //левая половина блока
    bit rData[32];    //правая половина блока
    bit crKey[48];    //перестановка ключа со сжатием
    bit erData[8][6]; //перестановка данных с расширением
    bit sOut[8][4];   //выход S-блоков
    bit spOut[32];     //выход P-блока
    bit Key[56];       //новый ключ
    //разбиваем блок данных на 2 половины
    lData = data;
    rData = data >> 32; //сдвиг в сторону младшего бита
    //16 этапов алгоритма DES
    for(int i = 0; i < 16; i = i + 1){
        //формирование ключа
        key[0] = key[0] <<< key_shift[i];
        key[1] = key[1] <<< key_shift[i];
        Key[56] = key[0] || key[1];
    }
```

```

//перестановка со сжатием
for(int j = 0; j < 48; j = j + 1)
    crKey[j] = Key[compress_replace[j]];
//перестановка с расширением
for(int j = 0; j < 8; j = j + 1)
    for(int s = 0; s < 6; s = s + 1)
        erData[j][s] = rData[exp_replace[8*j + s]]
erData = erData ^ crKey; //48 новых переменных кода
//подстановка в S-блоках (32 новых переменных кода)
for(int j = 0; j < 8; j = j + 1)
    MakeSBox(SBox[j], erData[j], sOut[j]);
//перестановка в P-блоке
for(int j = 0; j < 32; j = j + 1)
    spOut[j] = sOut[P[j]];
spOut = spOut ^ lData; //32 новых переменных кода
lData = rData;          //новый левый блок данных
rData = spOut;          //новый правый блок данных
}
code = lData || rData;
}

```

Особо отметим тот факт, что при пропозициональном кодировании операции сдвига ключа и операции перестановки комплекс Transalg не создает новых переменных кода — в разделе 2.2.1 описан механизм, который обновляет связи переменных программы с уже существующими элементами словаря термов Σ . Это означает, что операции перестановки и сдвига не вносят в пропозициональный код алгоритма шифрования новой информации, никак не усложняя, таким образом, задачу логического криптоанализа. Transalg создает новые переменные кода при обработке команд, выполняющих сложение по модулю 2 (48 и 32 новых переменных кода), а также создает 32 переменных для кодирования выходов S-блоков. Таким образом, в каждом раунде в пропозициональный код алгоритма добавляет-

ся 112 новых переменных. В таблице 3.3 приводятся характеристики КНФ для алгоритма DES, шифрующего 64 бита открытого текста, в зависимости от числа раундов. Итоговые КНФ, которые выдает комплекс Transalg, сравниваются с КНФ, полученными в работе [3].

Таблица 3.3. Результаты кодирования алгоритма DES, шифрующего 1 блок данных.

Число раундов	Программный комплекс Transalg				F. Massacci,	
	Без минимизации		Минимизация [103]		L. Marraro, [3]	
	Переменные	Дизъюнкты	Переменные	Дизъюнкты	Переменные	Дизъюнкты
3	456	7104	456	4950	1430	9064
4	568	9472	568	6600	2368	14811
5	680	11840	680	8250	3032	18738
...	...	...	...	...	...	...
14	1688	33152	1688	23100	9008	54081
15	1800	35520	1800	24750	9672	58008
16	1912	37888	1912	26400	10336	61935

Можно заметить, что пропозициональный код DES, полученный в результате применения комплекса Transalg, существенно компактнее кода, приведенного в статьях [3, 4]. Столь значительное сокращение объема кода достигается в основном за счет техники использования словаря термов Σ , позволяющей обрабатывать перестановки элементов в двоичных массивах без ввода дополнительных переменных кода.

3.1.5. Преобразование в булевы уравнения алгоритма хеширования MD5

Хеш-функцией называется дискретная функция, отображающая двоичные последовательности произвольной конечной длины в двоичные по-

следовательности фиксированной длины

$$h : \{0, 1\}^* \rightarrow \{0, 1\}^n, n \in \mathbb{N}.$$

Результат такого отображения называется хеш-значением. Разработано множество алгоритмов хеширования с различными характеристиками (см. [45, 46]). Большое практическое значение имеют криптографические хеш-функции. Эти функции используются в системах идентификации и аутентификации при построении цифровых подписей и обработке паролей. Используемые на практике криптографические хеш-функции должны обладать содержательной аргументацией трудности задач их обращения. Современные представления о криптографически стойких хеш-функциях формулируются в виде следующих требований [46]:

1. *Однонаправленность.* По известной функции h и известному хеш-значению t задача поиска двоичного вектора $M : h(M) = t, M \in \{0, 1\}^*$ является вычислительно трудной.
2. *Устойчивость к коллизиям первого рода.* По известной функции h и известному двоичному вектору M задача нахождения двоичного вектора $N \neq M$, такого, что $h(N) = h(M)$, вычислительно трудна.
3. *Устойчивость к коллизиям второго рода.* Задача поиска пары двоичных векторов (M, M') , таких, что $h(M) = h(M')$, вычислительно трудна.

Задача обращения применительно к хеш-функции h заключается в поиске по заданному хеш-значению t такого двоичного вектора M , что справедливо равенство $h(M) = t$. Для решения задачи достаточно найти любой из множества прообразов t при отображении h .

Задачи обращения криптографических хеш-функций интересны как с позиций анализа и разработки новых криптографических примитивов, так

и с позиций построения аргументированно трудных тестов для булевых решателей. Программный комплекс Transalg позволяет свести задачу обращения произвольной хеш-функции к поиску решений системы булевых уравнений. Проблему пропозиционального кодирования алгоритмов хеширования рассмотрим на примере широко известного алгоритма MD5 ([115], см. также [45, 46]).

Алгоритм MD5 обрабатывает входной текст блоками по 512 бит, каждый из которых делится на 16 32-битных подблока. Выходом алгоритма является последовательность из четырех 32-битных блоков, составляющих 128-битное хеш-значение.

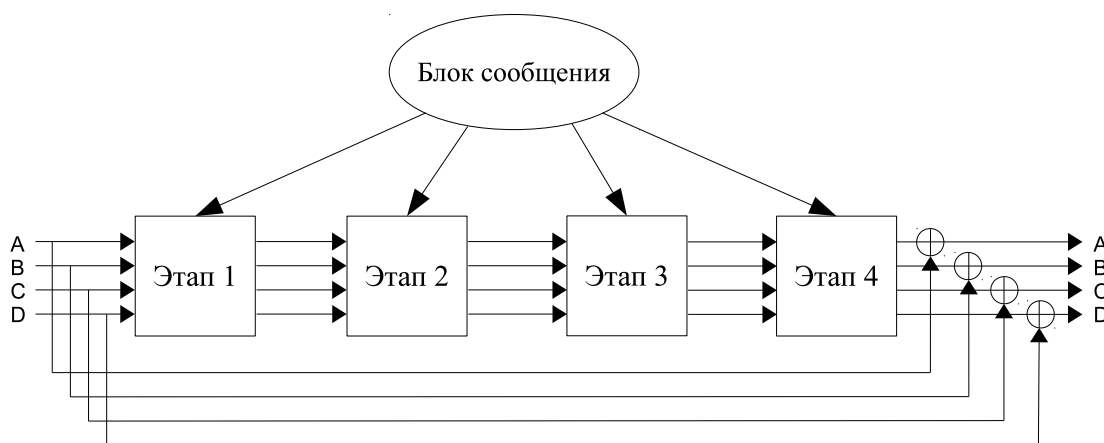


Рис. 3.7. Схема главного цикла алгоритма MD5.

Информация, поступающая на вход алгоритму, в ТА-программе представлена массивом **data**. Вычисленное программой хеш-значение сохраняется в массиве **hash**.

```
__in bit data[16][32]; //вход
__out bit hash[4][32]; //выход
```

Особенностью алгоритма MD5 является использование специальных массивов, образованных т. н. «переменными сцепления». На шаге инициализации соответствующие массивы заполнены известными булевыми константами. Однако в процессе построения хеша значения переменных сцепления изменяются: каждому последующему 512-битному блоку сообщения

ставятся в соответствие новые значения переменных сцепления, зависящие от содержимого предыдущего 512-битного блока. На заключительном шаге значения переменных сцепления образуют хеш-значение.

Массивы, хранящие переменные сцепления, объявляются с атрибутом `_mem` (см. пункт 2.2.3). Ниже приведена инициализация данных массивов в соответствии со стандартом MD5.

```
__mem bit A[32] = 0x01234567;  
__mem bit B[32] = 0x89ABCDEF;  
__mem bit C[32] = 0xFEDCBA98;  
__mem bit D[32] = 0x76543210;
```

Рассмотрим процесс хеширования одного 512-битного блока данных. Работа алгоритма MD5 начинается с копирования значений переменных сцепления в локальные переменные:

```
bit a[32] = A;  
bit b[32] = B;  
bit c[32] = C;  
bit d[32] = D;
```

Основу процедуры хеширования составляют четыре этапа (рисунок 3.7), на каждом из которых происходит преобразование целых чисел. На первом этапе результатом преобразования чисел A,B,C,D являются числа A^1 , B^1 , C^1 , D^1 , на втором этапе результатом преобразования чисел A^1 , B^1 , C^1 , D^1 являются числа A^2 , B^2 , C^2 , D^2 и так далее.

На каждом этапе выполняется 16 однотипных процедур, процесс работы которых поясняется рисунком 3.8. На данном рисунке показано, как преобразуются целые числа A,B,C,D. В данном случае в результате такого преобразования изменяется только число A. Вообще же работа каждой из 16 функций каждого этапа регламентируется стандартом MD5.

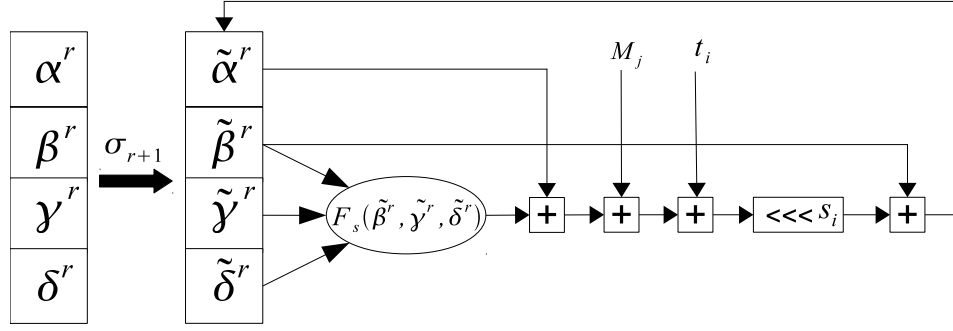


Рис. 3.8. Одна операция алгоритма MD5.

Набор чисел $(\tilde{\alpha}^r, \tilde{\beta}^r, \tilde{\gamma}^r, \tilde{\delta}^r)$ является результатом применения к набору $(\alpha^r, \beta^r, \gamma^r, \delta^r)$ перестановки σ_{r+1} , $r \in \{0, \dots, 15\}$:

$$(\tilde{\alpha}^r, \tilde{\beta}^r, \tilde{\gamma}^r, \tilde{\delta}^r) = \sigma_{r+1} : (\alpha^r, \beta^r, \gamma^r, \delta^r).$$

Все такие перестановки жестко зафиксированы в стандарте MD5. Для каждого этапа определяется нелинейная функция F_s , $s \in \{1, 2, 3, 4\}$.

Дискретное преобразование на каждом этапе является суперпозицией 16 функций. Каждая из этих функций получает на входе фрагмент исходного текста, вектор $(\tilde{\alpha}^r, \tilde{\beta}^r, \tilde{\gamma}^r, \tilde{\delta}^r)$, а также константы t_i , s_i , $i \in \{1, \dots, 64\}$. Выходом функции является набор целых чисел $(\alpha^{r+1}, \beta^{r+1}, \gamma^{r+1}, \delta^{r+1})$, определяемый следующим образом:

$$\begin{aligned} \alpha^{r+1} &= \tilde{\beta}^r + \left(\left(\tilde{\alpha}^r + F_1(\tilde{\beta}^r, \tilde{\gamma}^r, \tilde{\delta}^r) + M_j + t_i \right) \lll s_i \right) \\ \beta^{r+1} &= \tilde{\beta}^r \\ \gamma^{r+1} &= \tilde{\gamma}^r \\ \delta^{r+1} &= \tilde{\delta}^r \end{aligned}$$

Здесь M_j — обозначает j -й подблок сообщения, $j = \overline{0, 15}$, а функция F_1 имеет вид $F_1(x, y, z) = (x \cdot y) \vee (\bar{x} \cdot z)$.

В ТА-программе каждая функция F_s , $s \in \{1, 2, 3, 4\}$, реализована отдельной процедурой вида (для функции F_1):

```
bit[] F(bit[] X, bit[] Y, bit[] Z){
    return (X&Y)|(!X&Z);
}
```

Преобразования

$$(\alpha^r, \beta^r, \gamma^r, \delta^r) \rightarrow (\alpha^{r+1}, \beta^{r+1}, \gamma^{r+1}, \delta^{r+1})$$

реализуются при помощи стандартных операций языка ТА. Например,

`a = b + ((a + F(b,c,d) + M[0] + 0xd76aa478) <<< 7);`

Дополнительно отметим, что обработка операции циклического сдвига не добавляет в пропозициональный код алгоритма новых переменных (это достигается за счет использования словаря термов, см. пункт 2.2.1).

Пропозициональный код алгоритма MD5, полученный с помощью программного комплекса Transalg, представляет собой систему булевых уравнений. Подставив в нее некоторое хеш-значение $h(M)$ в форме двоичного вектора, получим совместную систему (либо уравнение вида КНФ=1), из любого решения которой можно эффективно извлечь такое сообщение M' , что $h(M) = h(M')$.

3.2. Исследование свойств дискретно-автоматных динамических систем

Поведение широкого класса дискретных систем можно описать автоматными моделями. Переходы в последующие состояния в таких системах происходят в дискретные моменты времени, и довольно часто функции, задающие эти переходы, оказываются эффективно вычислимыми (за полиномиальное от объема входных данных время). Устанавливать некоторые свойства такого рода систем можно, преобразуя алгоритмы вычисления функций переходов в булевы уравнения и добавляя при необходимости к получаемым системам дополнительные ограничения. Преобразование соответствующих алгоритмов может осуществляться при помощи программного комплекса Transalg.

Обширный класс подобных проблем исследуется в современной компьютерной биологии. В работе [116] был исследован класс дискретных автоматных отображений, описывающих динамику гипотетических генных сетей, введенных в [14]. В частности, были изучены задачи поиска неподвижных точек дискретно-автоматных отображений, заданных регуляторными контурами рассматриваемых сетей. Данные задачи были сведены к проблемам поиска решений систем булевых уравнений весьма простого вида. Это оказалось возможным вследствие природы пороговых функций в узлах сети. Особо отметим, что предложенная в [116] техника применима только к поиску неподвижных точек и не позволяет находить циклы отображений длины ≥ 2 . Однако была отмечена принципиальная возможность решения этой и других (более сложных задач) при помощи пропозиционального подхода. В этом разделе описано применение программного комплекса Transalg для сведения задач поиска циклов длины ≥ 2 автоматных отображений, описанных в [14], к SAT-задачам.

Функционирование регуляторного контура генной сети определяется динамикой изменения показателей концентрации вещества (например, белка) в его вершинах в каждый такт времени в зависимости от структуры всей сети и свойств функций в ее вершинах. Такого рода переходы генной сети в последующие состояния имеют «автоматную природу» и рассматриваются как результаты действия на графе сети отображения, называемого автоматным отображением, в частности, аддитивным или мультипликативным в зависимости от выбора функций в вершинах [14, 117]. Интерес представляет изучение различных свойств таких отображений.

Пусть $G = G(U, D)$ — ориентированный связный граф без петель и кратных дуг, $|U| = n$. Каждой вершине $i \in U$ приписан целый вес $x_i \in \{0, \dots, p-1\}$, $i \in \{1, \dots, n\}$, задающий концентрацию белка в данной вершине; натуральное число p , $p \geq 2$, при этом определяет верхнюю границу

значения веса по всем вершинам графа G и называется значностью.

Графу G можно ставить в соответствие различные отображения вида

$$A_G : \Sigma_p^n \rightarrow \Sigma_p^n,$$

где через Σ_p^n обозначено множество всевозможных слов длины n над алфавитом $\Sigma_p = \{0, \dots, p-1\}$. Смысл таких отображений заключается в пересчете за один такт времени значений весов во всех вершинах графа G . Функции, определяющие для каждой вершины новое значение, могут задаваться разными способами. В работе [14] был рассмотрен класс генных сетей, отображение A_G для которых задавалось посредством следующих функций пересчета весов вершин:

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

При $X_i = \emptyset$ полагаем, что $\sum_{x_j \in X_i} x_j = 0$. Здесь $\tilde{x} = (x_1, \dots, x_n)$ — начальный набор весов вершин графа G ; через X_i обозначено множество весов тех вершин из $U \setminus \{i\}$, дуги из которых ведут в вершину i , через x_i^1 обозначено новое значение веса вершины i , $i = 1, \dots, n$, тем самым $A_G(\tilde{x}) = \tilde{x}^1$. Автоматные отображения с функциями пересчета весов вершин вида (3.1) называются p -значными аддитивными автоматами.

Задача поиска неподвижных точек произвольного автоматного отображения A_G заключается в нахождении таких векторов значений весов вершин графа G , которые данным отображением переводятся сами в себя.

В работах [14, 117] были рассмотрены различные детализации задачи о неподвижных точках, а также сформулированы другие проблемы анализа структуры отображений A_G (в том числе и аддитивных автоматов). В частности, была поставлена задача поиска циклов различной длины для

отображений A_G . Цикл длины один — это неподвижная точка. Циклом длины r , $r \geq 2$, называется такое семейство наборов

$$\tilde{x} = \tilde{x}^0, \tilde{x}^1 = A_G(\tilde{x}), \dots, \tilde{x}^{r-1} = A_G(\tilde{x}^{r-2}),$$

что $A_G(\tilde{x}^{r-1}) = \tilde{x}$.

Как было отмечено выше, задача поиска циклов отображений, заданных посредством (3.1), допускает простую сводимость к булевым уравнениям [116]. При этом полноценного преобразования алгоритма, описывающего процесс функционирования генной сети, не требуется. Однако данная техника даже для рассматриваемых простых генных сетей не дает возможности сводить к булевым уравнениям задачи поиска циклов длины ≥ 2 .

В [116] были сформулированы основные принципы, позволяющие рассматривать широкий класс задач, касающихся поведения дискретно-автоматных моделей генных сетей, с позиции пропозиционального кодирования алгоритмов, вычисляющих дискретные функции. Пусть $S(A_G, k)$ — система булевых уравнений, кодирующая k итераций, выполняемых алгоритмом, который моделирует поведение генной сети с функцией пересчета весов вершин A_G . Добавив к $S(A_G, k)$ условия вида $\tilde{x}^0 = \tilde{x}^k$, получим систему, совместную тогда и тогда, когда для рассматриваемого отображения существует цикл длины k . Из произвольного решения полученной системы можно эффективно выделить цикл длины k . В Приложении Г приведен текст ТА-программы, которая моделирует три шага функционирования генной сети на трех вершинах с функцией пересчета весов вершин (3.1) (значность сети $p = 8$).

3.3. Процедуры сведения оптимизационных задач с псевдобулевыми ограничениями к SAT-задачам

Наблюдающийся в последние годы прогресс в алгоритмике SAT-решателей стимулирует применение этих методов в решении задач дискретной оптимизации, которые на первый взгляд могут показаться далекими от проблем пропозициональной выполнимости. Одним из сравнительно новых направлений такого рода является применение SAT-подхода к задачам с т.н. «псевдобулевыми ограничениями». Данная тематика активно развивалась в 2006–2008 гг. Сейчас наблюдается заметный спад этой активности, однако продолжают проводиться соревнования [118] и даже регулярно проходят специализированные конференции, посвященные «псевдобулевым решателям». Статья [12] является, пожалуй, наиболее полным введением в проблему сведения псевдобулевых задач к SAT-задачам.

3.3.1. Сведение к SAT задач 0-1-целочисленного линейного программирования

В этом разделе приводится исходная постановка задачи 0-1-ЦЛП, а также кратко описываются базовые приемы преобразования псевдобулевых ограничений в булевы. Все примеры преобразований были выполнены с применением комплекса Transalg.

Будем рассматривать задачу 0-1-ЦЛП в стандартной постановке. Дана система ограничений-неравенств

$$A \cdot x \leq b, \tag{3.2}$$

где A — $m \times n$ -матрица с целочисленными компонентами, b — вектор длины m , состоящий из целых чисел. Предполагаем, что переменные x_i , $i \in \{1, \dots, n\}$, принимают значения в множестве целых чисел $\{0, 1\}$. Множе-

ство решений системы (3.2) называется допустимым множеством. Требуется минимизировать на допустимом множестве целевую функцию — линейную форму $\langle c, x \rangle$, где c — заданный целочисленный вектор длины n . Поскольку любой задачу ЦЛП на максимум можно эффективно свести к эквивалентной задаче на минимум, то в дальнейшем будем рассматривать только задачу минимизации.

Процесс сведения данной задачи к SAT состоит в преобразовании линейных неравенств, образующих систему (3.2), в конъюнкции дизъюнктов. Первый шаг процедуры сведения заключается в выполнении эквивалентных преобразований исходных ограничений, которые приводят к ограничениям вида

$$\langle a', x^\sigma \rangle \leq b_0, \quad (3.3)$$

где a' — вектор длины n с целыми неотрицательными компонентами, x^σ — вектор, образованный литералами над переменными из множества $X = \{x_1, \dots, x_n\}$, а b_0 — неотрицательное целое число.

Пример 3.1. Рассмотрим ограничение $3 \cdot x_1 - 2 \cdot x_2 + 5 \cdot x_3 \leq 3$. В результате эквивалентного преобразования $x_2 = 1 - \overline{x_2}$ получим ограничение $3 \cdot x_1 + 2 \cdot \overline{x_2} + 5 \cdot x_3 \leq 5$.

Таким образом, осуществляется переход от исходной задачи к задаче минимизации линейной формы $\langle c', x^\sigma \rangle$ при условии выполнения m ограничений вида (3.3).

Отметим, что неравенство

$$a_1 \cdot x_1^{\sigma_1} + \dots + a_n \cdot x_n^{\sigma_n} \leq 0,$$

в предположении, что $a_i \neq 0$, $i \in \{1, \dots, n\}$ и x_i принимают значения в $\{0, 1\}$, выполняется лишь при $x_1^{\sigma_1} = \dots = x_n^{\sigma_n} = 0$. Поэтому далее предпо-

лагается, что в ограничениях вида

$$a_1 \cdot x_1^{\sigma_1} + \dots + a_n \cdot x_n^{\sigma_n} \leq b, \quad (3.4)$$

$a_i, i \in \{1, \dots, n\}, b$ — натуральные числа.

Задачи целочисленного программирования, переменные в которых принимают значения в множестве $\{0, 1\}$, в последние годы называют также задачами с псевдобулевыми ограничениями [12]. Этим подчеркивается, что значения переменных — это не значения истинности булевых функций, а целые числа. Вообще, термины «псевдобулева задача», «псевдобулево ограничение» и т.п. более верны в отношении формулировок типа (3.4), и именно в таком контексте они используются далее.

Произвольному целочисленному вектору с компонентами из $\{0, 1\}$ естественным образом сопоставляется внешне ничем от него не отличающийся булев вектор. В этом смысле произвольному ограничению (3.4) соответствует булева функция, которая принимает значение «истина» тогда и только тогда, когда выполняется (3.4).

Пусть $(\alpha_k \dots \alpha_0)$, $\alpha_j \in \{0, 1\}$, $j \in \{0, \dots, k\}$, $k = \lfloor \log a \rfloor$, — вектор двоичного представления натурального числа a . Тогда выражению $a \cdot x^\sigma$ поставим в соответствие слово

$$A(a \cdot x^\sigma) = (A_k \dots A_0)$$

над алфавитом $\{0, x^\sigma\}$ по следующему правилу: если $\alpha_j = 0$, то $A_j = 0$, если $\alpha_j = 1$, то $A_j = x^\sigma$, $j \in \{0, \dots, k\}$.

Пример 3.2. В соответствии с описанными правилами выражению $5 \cdot \overline{x_1}$ сопоставляется слово $(\overline{x_1}0\overline{x_1})$ (число 5 в двоичном представлении задается вектором (101)).

Линейной форме $a_1 \cdot x_1^{\sigma_1} + \dots + a_n \cdot x_n^{\sigma_n}$ ставится в соответствие система булевых уравнений, связывающая компоненты слов $A(a_1 \cdot x_1^{\sigma_1}), \dots, A(a_n \cdot x_n^{\sigma_n})$.

Фактически данная система уравнений описывает процесс суммирования натуральных чисел, представленных двоичными векторами. Так, если $A = (A_k \dots A_0)$ и $B = (B_k \dots B_0)$ — построенные по описанным правилам слова, которые соответствуют выражениям $a_1 \cdot x_1^{\sigma_1}$ и $a_2 \cdot x_2^{\sigma_2}$, то связывающая компоненты слов A и B система булевых уравнений будет аналогична системе (2.3).

Предположим, что построена система булевых уравнений, которая связывает компоненты слов $A(a_1 \cdot x_1^{\sigma_1}), \dots, A(a_n \cdot x_n^{\sigma_n})$. К данной системе добавим уравнения, которые кодируют тот факт, что результат целочисленного сложения $a_1 \cdot x_1^{\sigma_1} + \dots + a_n \cdot x_n^{\sigma_n}$ не превосходит натурального числа b . Итоговую систему булевых уравнений обозначим через Λ .

Применяя преобразования Цейтина, комплекс Transalg осуществляет переход от систем типа Λ к уравнениям вида КНФ=1. Для минимизации получаемых КНФ используется утилита Espresso [103]. Итогом описанных действий является такая КНФ $C(A, b)$, что между множеством решений системы (3.2) и множеством решений уравнения $C(A, b) = 1$ существует биекция, и от любого решения этого уравнения можно эффективно перейти к решению исходной системы целочисленных неравенств.

Пусть допустимое множество для исходной задачи 0-1-ЦЛП не пусто, $x^0 = (\alpha_1^0, \dots, \alpha_n^0)$ — некоторая его точка и $R = \sum_{i=1}^n c_i \cdot \alpha_i^0$ — значение целевой функции в данной точке. Чтобы попытаться улучшить значение целевой функции, можно добавить к системе ограничений (3.2) дополнительное условие в форме линейного неравенства. Это могут быть неравенства вида $\langle c, x \rangle \leq R - 1$, $\langle c, x \rangle \leq \lfloor \frac{R}{2} \rfloor$ и т.п. Вид этих ограничений зависит от выбранной схемы поиска. В соответствии с описанной выше техникой от произвольного такого неравенства возможен переход к уравнению вида КНФ=1. Таким образом, оптимизационный вариант задачи 0-1-ЦЛП может быть сведен к решению серии SAT-задач, причем если R — некоторое

«начальное приближение», то использование схемы с дихотомией гарантирует нахождение оптимального значения целевой функции $\langle c, x \rangle$ на допустимом множестве, определяемом системой ограничений (3.2), за $O(\log R)$ итераций.

Таблица 3.4. Сравнение результатов сведения псевдобулевых задач к SAT-задачам.

Задача	Transalg		MiniSat+	
	Переменные	Дизъюнкты	Переменные	Дизъюнкты
A05100	2469	19577	3568	17608
D10200	11987	103072	17777	99316
D20100	14847	128078	17905	102600
D10400	24110	197863	34380	191468
D20200	30012	259923	35739	206813
D20400	60243	519960	69789	404014
D15900	81804	705882	110942	651436

Описанные процедуры сведения задач 0-1-ЦЛП к SAT-задачам реализованы в виде отдельных модулей комплекса Transalg. При сведении задачи 0-1-ЦЛП к SAT входным текстом является не программа на языке ТА, а текстовый файл в LP-формате, широко используемом решателями задач целочисленного линейного программирования [119]. Для работы с данным форматом были разработаны и включены в программный комплекс Transalg специализированные модули разбора и преобразования. В таблице 3.4 приведено сравнение результатов преобразования 0-1-ЦЛП задач в SAT (исходные задачи были взяты из библиотеки [120]). Сравниваются КНФ, получаемые комплексом Transalg, и КНФ, полученные известной программой MiniSat+ (см. [121]).

3.3.2. Сведение к SAT квадратичных задач о назначениях (QAP)

Следует отметить, что на текущий момент SAT-подход проигрывает по эффективности коммерческим решателям 0-1-ЦЛП задач на целом ряде практически значимых тестов из специализированных библиотек целочисленной оптимизации (см., например, [120]). Однако в дискретной оптимизации существуют задачи, на которых традиционные методы, показывающие хорошие результаты на 0-1-ЦЛП задачах, неэффективны. Одной из таких задач является квадратичная задача о назначениях (QAP) [40], имеющая многочисленные применения на практике (логистика, планирование производства, теория расписаний, статистический анализ данных и др.). Далее приведена краткая постановка данной задачи.

Даны некоторые n «объектов», которые размещаются по n «позициям». Дана $n \times n$ -матрица $D = \|d_{ij}\|$ неотрицательных целых чисел, определяющая расстояния между позициями. Также дана $n \times n$ -матрица $F = \|f_{ij}\|$ неотрицательных целых чисел, определяющая «интенсивность потока» между объектами. Предположим, что объекты и позиции пронумерованы натуральными числами от 1 до n и начальное размещение задается тождественной подстановкой:

$$\begin{pmatrix} 1 & 2 & \dots & n \\ 1 & 2 & \dots & n \end{pmatrix}.$$

Рассматривается функция

$$\sum_{i=1}^n \sum_{j=1}^n f_{ij} \cdot d_{p(i)p(j)}, \quad (3.5)$$

где p — произвольная подстановка на n -элементном множестве. Требуется найти подстановку $p \in S_n$, доставляющую минимум функции (3.5).

Для преобразования QAP-задач в SAT был использован программный комплекс Transalg. На начальном шаге преобразования строится $n \times n$ -матрица $X = \|x_{ij}\|$ (т. н. «матрица размещений»), элементами которой являются булевы переменные. Смысл данной матрицы в том, что она кодирует произвольное размещение объектов по позициям, соответствующее «неопределенной» перестановке p : булева переменная x_{ij} принимает значение «1» тогда и только тогда, когда в результате перестановки p объект с номером i переместился в позицию с номером j . Исходя из этого, можно записать следующую систему булевых ограничений на значения переменных x_{ij} , $i, j \in \{1, \dots, n\}$:

$$\left\{ \begin{array}{l} \bigvee_{k=1}^n x_{1k} \quad \& \quad \bigwedge_{j \in \{1, \dots, n\} \setminus \{k\}} \bar{x}_{1j} = 1 \\ \dots \\ \bigvee_{k=1}^n x_{nk} \quad \& \quad \bigwedge_{j \in \{1, \dots, n\} \setminus \{k\}} \bar{x}_{nj} = 1 \\ \bigvee_{k=1}^n x_{k1} \quad \& \quad \bigwedge_{i \in \{1, \dots, n\} \setminus \{k\}} \bar{x}_{i1} = 1 \\ \dots \\ \bigvee_{k=1}^n x_{kn} \quad \& \quad \bigwedge_{i \in \{1, \dots, n\} \setminus \{k\}} \bar{x}_{in} = 1 \end{array} \right.$$

Следующий шаг преобразования заключается в построении целевой функции в явном виде, т.е. в виде формулы над переменными x_{ij} , $i, j \in \{1, \dots, n\}$. Для этого просматривается матрица F и определяются пары объектов с номерами i_1 и i_2 , между которыми интенсивность потока положительна. Для каждой такой пары объектов в матрице X рассматриваются строки с номерами i_1 и i_2 , и формируется следующий фрагмент целевой функции (в предположении, что матрица D симметрична):

$$\begin{aligned} h_{(i_1 i_2)} = & f_{i_1 i_2} \cdot d_{12} \cdot (x_{i_1 1} \cdot x_{i_2 2} \vee x_{i_1 2} \cdot x_{i_2 1}) + \dots \\ & \dots + f_{i_1 i_2} \cdot d_{(n-1)n} \cdot (x_{i_1 (n-1)} \cdot x_{i_2 n} \vee x_{i_1 n} \cdot x_{i_2 (n-1)}) . \end{aligned}$$

Формула, выражающая целевую функцию, имеет следующий вид:

$$h = \sum_{i_1, i_2: f_{i_1 i_2} > 0} h_{(i_1 i_2)}.$$

Выражение h преобразуется в систему булевых уравнений с использованием структур данных и алгоритмов комплекса Transalg.

Следует отметить, что целевая функция в задаче QAP нелинейна по переменным из множества X . Современные коммерческие решатели ЦЛП-задач, комбинирующие методы ветвей, границ и отсечений с методами решения задач линейного программирования над полем рациональных чисел, с такими постановками, как правило, не работают. Сказанное означает перспективность представленной технологии преобразования псевдобулевых задач в булевы уравнения в исследовании задач дискретного программирования, в том числе с нелинейными ограничениями и невыпуклыми целевыми функциями.

Основные выводы по третьей главе. Описанный во второй главе комплекс Transalg применялся для преобразования в булевы уравнения ТА-программ, описывающих функционирование разнообразных дискретных систем. В большинстве случаев результатом такого преобразования является пропозициональный код алгоритма вычисления дискретной функции в виде уравнения КНФ=1. В получаемых КНФ входные и выходные данные рассматриваемой функции представлены булевыми переменными. Такие КНФ называются шаблонными (шаблонами). SAT-задача, кодирующая проблему обращения функции в конкретной точке $y \in \text{range } f_n$ получается в результате приписывания (по конъюнкции) к шаблонной КНФ однолитеральных дизъюнктов, кодирующих слово y . Получаемая КНФ подается на вход SAT-решателю. Из ее выполняющего набора эффективно извлекается такое слово $x \in \{0, 1\}^n$, что $f_n(x) = y$.

В результате применения комплекса Transalg была построена библио-

тека шаблонов (в виде КНФ, представленных в формате DIMACS [104]), кодирующих задачи обращения ряда криптографических функций (алгоритм DES, генератор A5/1, суммирующий генератор, генератор Гиффорда, генератор Геффе, алгоритм хеширования MD5). В построенную библиотеку шаблонов также включены КНФ, кодирующие задачи 0-1–ЦЛП, QAP и задачи поиска циклов для некоторых дискретно-автоматных динамических моделей генных сетей.

Заключение

Основная цель диссертации заключалась в разработке и программной реализации общего подхода к преобразованию алгоритмов вычисления дискретных функций в булевы уравнений.

Далее перечислены основные результаты диссертации:

1. Разработана общая методология преобразования высокоуровневых описаний алгоритмов вычисления дискретных функций в булевы уравнения; сформулированы общие принципы и описаны базовые механизмы таких преобразований.
2. Разработан проблемно-ориентированный язык процедурного типа, предназначенный для описания алгоритмов вычисления всюду определенных дискретных функций и обладающий семантикой, обеспечивающей корректное построение булевых уравнений.
3. Разработаны и реализованы структуры данных и алгоритмы, позволяющие автоматизировать преобразования в булевы уравнения процедур вычисления дискретных функций, описанных на специально созданном языке.
4. Построен программный комплекс, интегрирующий все разработанные алгоритмы и структуры данных и позволяющий, помимо собственно преобразования, оптимизировать получаемые в результате булевы структуры, а также осуществлять проверку их корректности.
5. С использованием построенного программного комплекса создан набор шаблонов, хранящих пропозициональные коды (в форме систем булевых уравнений) ряда функциональных примитивов, широко используемых при описании поведения дискретных систем.

Литература

- [1] Prestwich S.D. CNF encodings. In Handbook of Satisfiability (editors: A. Biere, M. Heule, H. van Maaren, T. Walsh). 2009. Vol. 6. pp. 75–97.
- [2] Marques-Silva J., Lynce I. Towards robust cnf encodings of cardinality constraints // In Thirteenth International Conference on Principles and Practice of Constraint Programming. Springer. Lecture Notes in Computer Science. 2007. Vol. 4741. pp. 483–497.
- [3] Massacci F., Marraro L. Logical Cryptanalysis as a SAT Problem // Journal of Automated Reasoning. 2000. Vol. 24, no. 1–2. pp. 165–203.
- [4] Massacci F., Marraro L. Logical Cryptanalysis as a SAT Problem: the Encoding of the Data Encryption Standard. // Preprint. Dipartimento di Informatica e Sistemistica, Universita di Roma "La Sapienza". 1999.
- [5] Gent I.P., Lynce I. A sat encoding for the social golfer problem // In Workshop on Modelling and Solving Problems with Constraints, IJCAI'05. 2005.
- [6] Gent I.P., Nightingale P. A new encoding of alldifferent into sat // In Third International Workshop on Modelling and Reformulating Constraint Satisfaction Problems, CP'04. 2004.
- [7] Опарин Г. А., Богданова В. Г., Сидоров И. А. Интеллектуальный решатель задач в булевых ограничениях в распределенной вычислительной среде // Информационные и математические технологии в науке и управлении: — Иркутск: ИСЭМ РАН. 2007. С. 32–40.
- [8] Опарин Г. А., Новопашин А. П. Булево моделирование планирования

действий в распределенных вычислительных системах // Известия РАН. Теория и системы управления. 2004. № 5. С. 105–108.

- [9] Oparin G. A., Novopashin A. P. Boolean models and planning methods for parallel abstract programs // Automation and Remote Control. 2008. Vol. 69, no. 8. pp. 1423–1432.
- [10] Опарин Г. А., Новопашин А. П. Булевы модели синтеза параллельных планов решения вычислительных задач // Вестник НГУ. Серия: Информационные технологии. 2008. Т. 6, № 1. С. 53–59.
- [11] Gu J., Purdom P., Franco J., Wah B. W. Algorithms for the satisfiability problem. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2000.
- [12] Een N., Sorrensson N. Translating Pseudo-Boolean Constraints into SAT // Journal on Satisfiability, Boolean Modeling and Computation. 2006. Vol. 2. pp. 1–25.
- [13] Касьянов В. Н., Поттосин И. В. Методы построения трансляторов. Новосибирск: «Наука», 1986. С. 344.
- [14] 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.
- [15] 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.

- [16] MiniSat. URL: <http://minisat.se/MiniSat.html> (дата обращения: 10.05.2011).
- [17] Отпущенников И. В., Семенов А. А. Технология трансляции комбинаторных проблем в булевы уравнения // Прикладная дискретная математика. 2011. № 1. С. 96–115.
- [18] Заикин О. С., Отпущенников И. В., Семенов А. А. Параллельные алгоритмы решения проблемы выполнимости в применении к оптимизационным задачам с булевыми ограничениями // Вычислительные методы и программирование: Новые вычислительные технологии (Электронный журнал). 2011. Т. 12. С. 205–212.
- [19] Отпущенников И. В., Семенов А. А. Преобразования алгоритмов вычисления дискретных функций в булевы уравнения // Известия ИГУ. Серия: математика. 2011. Т. 4, № 1. С. 83–96.
- [20] Семенов А. А., Отпущенников И. В., Кочемазов С. Е. Пропозициональный подход в задачах тестирования дискретных автоматов // Современные технологии. Системный анализ. Моделирование. 2009. № 4. С. 48–56.
- [21] Семенов А. А., Отпущенников И. В. Об алгоритмах обращения дискретных функций из одного класса // Прикладные алгоритмы в дискретном анализе. Серия: дискретный анализ и информатика; вып. 2. Изд-во ИГУ. 2008. С. 127–156.
- [22] Отпущенников И. В., Семенов А. А. Программная трансляция алгоритмов в пропозициональную логику применительно к комбинаторным задачам // Прикладная дискретная математика. Приложение. 2010. № 3. С. 81–82.

- [23] Семенов А. А., Заикин О. С., Отпущенников И. В., Буров П. С. О некоторых особенностях задач обращения дискретных функций // Труды XIV Байкальской Международной школы-семинара «Методы оптимизации и их приложения». — Иркутск: ИСЭМ СО РАН. 2008. Т. 1. С. 498–505.
- [24] Заикин О. С., Отпущенников И. В., Семенов А. А. Параллельные алгоритмы решения SAT в применении к оптимизационным задачам с булевыми ограничениями // Труды V Международной конференции «Параллельные вычислительные технологии» (ПАВТ 2011). Москва, МГУ. 2011. С. 501–508.
- [25] Отпущенников И. В., Семенов А. А. Инструментальное средство трансляции алгоритмов вычисления дискретных функций в выражения исчисления высказываний Transalg 1.0. Свидетельство о государственной регистрации программы для ЭВМ № 2011611151 (03.02.2011).
- [26] Яблонский С. В. Введение в дискретную математику. Москва: «Наука», 1986. С. 384.
- [27] Мендельсон Э. Введение в математическую логику. Москва: «Наука», 1971. С. 320.
- [28] Катленд Н. Вычислимость. Введение в теорию рекурсивных функций. Москва: «Мир», 1983. С. 256.
- [29] Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. Москва: «Мир», 1982. С. 416.
- [30] Пападимитриу Х., Стайглиц К. Комбинаторная оптимизация. Алгоритмы и сложность. Москва: «Мир», 1985. С. 510.

- [31] Васильев Ю. Л., Ветухновский Ф. Я., Глаголев В. В. и др. Дискретная математика и математические вопросы кибернетики. Москва: «Наука», 1974. Т. 1.
- [32] Нигматуллин Р. Г. Сложность булевых функций. Москва: «Наука», 1991. С. 240.
- [33] Wegener I. The complexity of Boolean function. Stuttgart: J. Wiley and Sons ltd, 1987.
- [34] Закревский А. Д., Поттосин Ю. В., Черемисинова Л. Д. Логические основы проектирования дискретных устройств. Москва: ФИЗМАТЛИТ, 2007. С. 590.
- [35] Кларк Э. М., Грамберг О., Пелед Д. Верификация моделей программ: Model Checking. М., МЦНМО, 2002.
- [36] Гаранина Н. О., Шилов Н. В. Верификация комбинированных логик знаний, действий и времени в моделях // Системная информатика. 2005. Т. 10. С. 114–173.
- [37] Amman P., Ritchey R. Using Model Checking to Analyze Network Vulnerabilities // Proc. Of the 2000 IEEE Symposium on Security and Privacy. 2000. pp. 156–165.
- [38] Sheyer O., Jha S., Wing J., et al. Automated Generation and Analysis of Attack Graphs // 2002 IEEE Symposium on Security and Privacy. 2002.
- [39] Nemhauser G., Wolsey L. Integer and Combinatorial Optimization. John Wiley and Sons Ltd., 1998.
- [40] Cela E. The Quadratic Assignment Problem. Theory and Algorithms. Kluwer Academic Publishers, 1998. P. 287.

- [41] Schrijver A. Combinatorial Optimization. Springer Verlag, Berlin, 2003.
- [42] 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.
- [43] Shepherdson J. C., Sturgis H. E. Computability of recursive functions // J. Assoc. Comp. Machinery. 1963. Vol. 10. pp. 217–255.
- [44] 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.
- [45] Menezes A., van Oorshot P., Vanstone S. Handbook of Applied Cryptography. CRC Press, 1996. P. 657.
- [46] Schneier B. Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C. John Wiley and Sons, 1996. P. 758.
- [47] Лидл Р., Нидеррайтер Г. Конечные поля (в двух томах). Москва: «Мир», 1988.
- [48] Biryukov A., Shamir A., Wagner D. Real Time Cryptanalysis of A5/1 on a PC. Fast Software Encryption Workshop 2000, April 10-12. New-York City, 2000.
- [49] Семенов А. А. О преобразованиях Цейтина в логических уравнениях // Прикладная дискретная математика. 2009. № 4. С. 28–50.
- [50] Цейтин Г. С. О сложности вывода в исчислении высказываний // Записки научных семинаров ЛОМИ АН СССР. 1968. Т. 8. С. 234–259.

- [51] Tseitin G. On the complexity of derivation in propositional calculus // Automation of reasoning. 1983. Vol. 2. pp. 466–483.
- [52] Hachtel G.D., Somenzi F. Logic synthesis and verification algorithms. Kluwer Ac. Publ., 2002.
- [53] 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.
- [54] Berkeley Logic Synthesis and Verification Group, ABC: A System for Sequential Synthesis and Verification. URL: <http://www.eecs.berkeley.edu/alanmi/abc/> (дата обращения: 10.05.2011).
- [55] 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.
- [56] Advanced Formal Verification // Ed. by R. Drechsler. Springer, 2005. P. 280.
- [57] Ganai M., Gupta A. SAT-based scalable formal verification solutions. Springer, 2007. P. 326.
- [58] Boole G. An investigation of the laws of thought on which are founded the mathematical theories of logic and probabilities. London, 1854.
- [59] Rudeanu S. Boolean functions and equations. Amsterdam-London: North-Holland Publ. Comp., 1974. P. 442.
- [60] Akers S. B. On a theory of Boolean functions // J. Soc. Ind. Appl. Math. 1959. Vol. 7, no. 4. pp. 487–498.

- [61] Buttner W., Simonis H. Embedding Boolean expressions into logic programming // Journal of Symbolic Computation. 1987. Vol. 4. pp. 191–205.
- [62] Brown F. M. Boolean Reasoning: The Logic Of Boolean Equations. Dover Publications, 2003. P. 304.
- [63] Бохманн Д., Постхоф Х. Двоичные динамические системы. Москва: «Энергоатомиздат», 1986. С. 401.
- [64] Семенов А. А., Буранов Е. В. Погружение задачи криптоанализа симметричных шифров в пропозициональную логику // Вычислительные технологии (совместный выпуск с журналом «Региональный вестник Востока»). 2003. Т. 8. С. 118–126.
- [65] Buchberger B. Grobner-Bases: An Algorithmic Method in Polynomial Ideal Theory // Multidimensional Systems Theory. 1985. Vol. 6. pp. 184–232.
- [66] Кокс Д., Литтл Д., О’Ши Д. Идеалы, многообразия и алгоритмы. Москва: «Мир», 2000. С. 687.
- [67] Агибалов Г. П. Логические уравнения в криптоанализе генераторов ключевого потока // Вестник Томского гос. ун-та. Приложение. 2003. № 6. С. 31–41.
- [68] Тимошевская Н. Е. О линеаризационно эквивалентных покрытиях // Вестник Томского гос. ун-та. Приложение. 2005. № 14. С. 84–91.
- [69] Тимошевская Н. Е. Задача о кратчайшем линеаризационном множестве // Вестник Томского гос. ун-та. Приложение. 2005. № 14. С. 79–83.
- [70] Семенов А. А. О сложности обращения дискретных функций из одного

класса // Дискретный анализ и исследование операций. 2004. Т. 11, № 4. С. 44–55.

- [71] 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.
- [72] Bryant R. E. Graph-Based Algorithms for Boolean Function Manipulation // IEEE Transactions on Computers. 1986. Vol. 35, no. 8. pp. 677–691.
- [73] Bryant R. E. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams // ACM Computing Surveys. 1992. Vol. 24, no. 3. pp. 293–318.
- [74] Meinel C., Theobald T. Algorithms and Data Structures in VLSI-Design: OBDD-Foundations and Applications. Springer-Verlag, 1998.
- [75] Davis M., Logemann G., Loveland D. A machine program for theorem-proving // Communications of the ACM. 1962. Vol. 5, no. 7. pp. 394–397.
- [76] 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.
- [77] 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.
- [78] Goldberg E., Novikov Y. Berkmin: The Fast and Robust SAT Solver // Automation and Test in Europe (DATE). 2002. pp. 142–149.
- [79] Prestwich S. D. Sat problems with chains of dependent variables // Discrete Applied Mathematics. 2002. Vol. 3037. pp. 1–2.

- [80] Crawford J.M., Baker A.B. Experimental results on the application of satisfiability algorithms to scheduling problems // In Twelfth National Conference on Artificial Intelligence. AAAI Press. 1994. Vol. 2. pp. 1092–1097.
- [81] Ernst M., Millstein T., Weld D. Automatic sat-compilation of planning problems // In Fifteenth International Joint Conference on Artificial Intelligence. 1997. pp. 1169–1176.
- [82] Опарин Г. А., Богданова В. Г. РЕБУС — интеллектуальный решатель комбинаторных задач в булевых ограничениях // Вестник НГУ. Серия: Информационные технологии. 2008. Т. 6, № 1. С. 60–68.
- [83] Gent I. P. Arc consistency in sat. IOS Press, 2002. pp. 121–125.
- [84] Kasif S. On the parallel complexity of discrete relaxation in constraint satisfaction networks // Artificial Intelligence. 1990. Vol. 45. pp. 275–286.
- [85] Walsh T. Sat vs CSP // Lecture Notes in Computer Science. 2000. Vol. 1894. pp. 441–456.
- [86] Bessiere C., Hebrard E., Walsh T. Local consistencies in sat // Lecture Notes in Computer Science. 2003. Vol. 2919. pp. 229–314.
- [87] Bacchus F. Gac via unit propagation // Lecture Notes in Computer Science. 2007. Vol. 4741. pp. 133–147.
- [88] Сергиенко А. М. VHDL для проектирования вычислительных устройств. Изд-во ТИД «ДС», 2003. С. 208.
- [89] Поляков А. К. Языки VHDL и Verilog в проектировании цифровой аппаратуры. Москва: СОЛОН — Пресс, 2003. С. 320.

- [90] King J. C. Symbolic execution and program testing // Communications of the ACM. 1976. Vol. 19, no. 7. pp. 385–394.
- [91] Hansen T., Schachte P., Sondergaard H. State joining and splitting for the symbolic execution of binaries // Springer-Verlag Berlin Heidelberg. 2009. pp. 76–92.
- [92] Abstract Interpretation, MIT course. URL: <http://web.mit.edu/afs/athena.mit.edu/course/16/16.399/www> (дата обращения: 10.05.2011).
- [93] Bagnara R., Hill P. M., Pescetti A., Zaffanella E. On the design of generic static analyzers for imperative languages. 2007. arXiv:cs/0703116v2 [cs.PL].
- [94] Wichmann B., Canning A., Clutterbuck D. L. et al. Industrial Perspective on Static Analysis // Software Engineering Journal. 1995. Vol. 10, no. 2. pp. 69–75.
- [95] Ayewah N., Hovemeyer D., Morgenthaler J. D. et al. Using Static Analysis to Find Bugs // IEEE Software. 2008. Vol. 25, no. 5. pp. 22–29.
- [96] Ершов А. П. Проблемно-ориентированный язык программирования // Математическая энциклопедия. 1977–1985.
- [97] Ахо А., Сети Р., Ульман Д. Компиляторы. Принципы, технологии, инструменты. Москва, СПб, Киев: «Вильямс», 2001. С. 768.
- [98] Ахо А., Ульман Д. Теория синтаксического анализа, перевода и компиляции. Москва: «Мир», 1977.
- [99] Карпов Ю. Г. Теория и технология программирования. Основы построения трансляторов. СПб: БХВ-Петербург, 2005. С. 272.
- [100] Керниган Б. У., Ритчи Д. М. Язык программирования С. М.: «Вильямс», 2006. С. 304.

- [101] Лавров С. С. Программирование. Математические основы, средства, теория. СПб: БХВ-Петербург, 2001. С. 314.
- [102] Себеста Р. У. Основные концепции языков программирования. М.: «Вильямс», 2001. С. 672.
- [103] Espresso. URL: <http://embedded.eecs.berkeley.edu/pubs/downloads/espresso> (дата обращения: 10.05.2011).
- [104] DIMACS. URL: <http://www.satalia.com/technology/cnf/> (дата обращения: 10.05.2011).
- [105] Biere A. The AIGER And-Inverter Graph (AIG) Format Version 20070427 // Johannes Kepler University. 2007. URL: <http://fmv.jku.at/aiger/> (дата обращения: 10.05.2011).
- [106] Cook S., Mitchel D. G. Finding hard instances of the satisfiability problem: A survey // DIMACS Series in Discrete Mathematics and Theoretical Computer Science. 1997. Vol. 35. pp. 1–17.
- [107] Заикин О. С., Семенов А. А. Технология крупноблочного параллелизма в SAT-задачах // Проблемы управления. 2008. № 1. С. 43–50.
- [108] Посыпкин М. А., Заикин О. С., Беспалов Д. В., Семенов А. А. Решение задач криптоанализа поточных шифров в распределенных вычислительных средах // Труды ИСА РАН. 2009. Т. 46. С. 119–137.
- [109] Rueppel R. A. Correlation immunity and the summation combiner // In Lecture Notes in Computer Science 218; Advances in Cryptology: Proc. Crypto'85, H. C. Williams Ed., Santa Barbara, CA, Aug. 18–22. 1985. Vol. 2. pp. 260–272.

- [110] Gifford D.K., Lucassen J.M., Berlin S.T. The Application of Digital Broadcast Communication to Large Scale Information Systems // IEEE Journal on Selected Areas in Communications. 1985. Vol. 3, no. 3. pp. 457–467.
- [111] Поточные шифры. Результаты зарубежной открытой криптологии. Москва: «Мир», 1997. С. 389.
- [112] Cain T.R., Sherman A.T. How to break Gifford's Cipher // CRYPTOLOGIA. 1997. Vol. 21, no. 3. pp. 237–286.
- [113] Biham E., Shamir A. Differential Cryptanalysis of DES-like Cryptosystems // Advances in Cryptology- CRYPTO 90 Proceeding, Springer-Verlag. 1991. pp. 2–21.
- [114] Guneysu T., Kasper T., Novotny M. et al. Cryptanalysis with COPACOBANA // IEEE Transactions on computers. 2008. Vol. 57, no. 11. pp. 1498–1513.
- [115] Rivest L.R. The MD5 Message Digest Algorithm, RFC 1321. 1992. URL: <http://rfc.com.ru/rfc1321.htm> (дата обращения: 10.05.2011).
- [116] Евдокимов А.А., Кочемазов С.Е., Семенов А.А. Применение символьных вычислений к исследованию дискретных моделей некоторых классов генных сетей // Вычислительные технологии. 2011. Т. 16, № 1. С. 30–47.
- [117] 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.

- [118] The International Pseudo-Boolean Competition. URL: <http://www.cril.univ-artois.fr/PB10> (дата обращения: 10.05.2011).
- [119] CPLEX Solver. URL: <http://www.aimms.com/features/solvers/cplex> (дата обращения: 10.05.2011).
- [120] MIPLIB — Mixed Integer Problem Library. URL: <http://miplib.zib.de> (дата обращения: 10.05.2011).
- [121] MiniSat+. URL: <http://minisat.se/MiniSat+.html> (дата обращения: 10.05.2011).

Приложение А

Грамматика языка ТА

```
//программа есть набор объявлений и определений языка
<program> ::= <definition_list>
<definition_list> ::= <definition> <definition_list> | <empty>

//объявление: переменной, массива или функции
<definition> ::= <var_def> ";" | <array_def> ";" | <function_def>

//объявление переменной начинается с типа данных,
//затем следует идентификатор и инициализирующее выражение
//опционально используются атрибуты __in, __out или __mem
<attr> ::= "__in" | "__out" | "__mem"
<var_def> ::= [<attr>] <data_type> <id>["=" <expression>]

//объявление массива;
//после идентификатора перечисляются измерения массива
<array_def> ::= [<attr>] <data_type> <id> "["<const>"]"
               {"["<const>"]"}["=" <array_init>]

//многомерные массивы инициализируются таблицами
<table> ::= "{" <table> {, <table>} {"}

//типы данных языка ТА
<data_type> ::= "void"|"bool"|"int"|"bit"

//определение функции (по аналогии с языком С)
<function_def> ::= <data_type> <id>("<param_list>")<block>
<param_list> ::= <data_type> <id> {, <data_type> <id>} | <empty>

//составной оператор языка ТА
//с каждым составным оператором связана локальная область видимости
<block> ::= "{" <stmt_list> {"}

//последовательность инструкций, разделенных символов ";"
<stmt_list> ::= <stmt> ";" {<stmt> ";" } | <empty>
```

```

//Инструкции языка ТА:
//1. Объявление/определение переменной или массива переменных;
//2. Объявление именованной константы;
//3. Оператор присваивания;
//4. Составной оператор;
//5. Оператор цикла;
//6. Оператор условного перехода;
//7. Оператор вызова функции;
//8. Оператор возврата из функции.
<stmt> ::= <var_def> | <array_def> | <assignment> |
          <func_call> | <for_loop> | <if_stmt> |
          <return_stmt> | <block>

//оператор вызова функции
<func_call> ::= <id> "(" <params> ")"
<params> ::= <empty> | <expression> {"," <expression>}

//оператор цикла
<for_loop> ::= "for" "(" <stmt> ";" <expression> ";" <stmt> ")"
              <stmt>

//оператор условного перехода
<if_stmt> ::= "if" "(" <expression> ")" <stmt>
             {"else" "if" "(" <expression> ")" <stmt>}
             ["else" <stmt>]

//оператор возврата из функции
<return_stmt> ::= "return" <expression>

//оператор присваивания
//слева - переменная или элемент массива; справа - выражение
<assignment> ::= <lValue> "=" <expression>
<lValue> ::= <id> {"[" <expression> "]" }

//выражение
<expression> ::= <operand> {<operation> <operand>}
//операнд

```

```

<operand> ::= <id> [{" "<expression>"}] | "("<param_list>")" |
               <const> | ["!" | "-" ]<operand> | "("<expression>")"

//операции языка TA
<operation> ::= "+" | "-" | "*" | "/" | "&" | "^" | "|" | "!" | "||" | "<" | ">" |
               "<<" | ">>" | "<<*" | ">>*" | ">" | ">=" | "<" | "<=" |
               "==" | "!=" | "="

//целочисленная константа
<const> ::= ["0x"] <digit> {digit}
<digit> ::= [0-9]

//идентификатор - любая комбинация букв и цифр,
//начинающаяся буквы или символа подчеркивания
<id> ::= <letter> {<letter> | <digit>}
<letter> ::= [a-z_A-Z]

```

Приложение Б

ТА-программа, вычисляющая 100 бит ключевого потока генератора А5/1

```
__in bit regA[19];
__in bit regB[22];
__in bit regC[23];
__out bit result[100];
bit shift_rslosA(){
    bit x = regA[18];
    bit y = regA[18]^regA[17]^regA[16]^regA[13];
    for(int j = 18; j > 0; j=j-1){
        regA[j] = regA[j-1];
    }
    regA[0] = y;
    return x;
}
bit shift_rslosB(){
    bit x = regB[21];
    bit y = regB[21]^regB[20];
    for(int j = 21; j > 0; j=j-1){
        regB[j] = regB[j-1];
    }
    regB[0] = y;
    return x;
}
bit shift_rslosC(){
    bit x = regC[22];
    bit y = regC[22]^regC[21]^regC[20]^regC[7];
    for(int j = 22; j > 0; j=j-1){
        regC[j] = regC[j-1];
    }
    regC[0] = y;
    return x;
}
bit majority(bit A, bit B, bit C){
```

```

    return A&B|A&C|B&C;
}
void main(){
    int midA = 8;
    int midB = 10;
    int midC = 10;
    bit maj;
    for(int i = 0; i < 100; i=i+1){
        maj = majority(regA[midA],regB[midB],regC[midC]);
        if(!(maj^regA[midA])) shift_rslosA();
        if(!(maj^regB[midB])) shift_rslosB();
        if(!(maj^regC[midC])) shift_rslosC();
        result[i] = regA[18]^regB[21]^regC[22];
    }
}

```

Приложение В

ТА-программа, реализующая шифрование 64 бит открытого текста алгоритмом DES

```
__in bit data[64];
__in bit key[2][28];
__out bit code[64];

int key_shift[16] = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};

int compress_replace[48] =
    {14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10, 23, 19, 11, 4,
     26, 8, 16, 7, 27, 20, 13, 2, 41, 52, 31, 37, 47, 55, 30, 40,
     51, 45, 33, 48, 44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32};

int exp_replace[48] =
    {32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8, 9, 10, 11,
     12, 13, 12, 13, 14, 15, 16, 17, 16, 17, 18, 19, 20, 21, 20, 21,
     22, 23, 24, 25, 24, 25, 26, 27, 28, 29, 28, 29, 30, 31, 32, 1};

//S-блоки
int SBox[8][4][16] = {
    //1 S-box
    {{14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7},
     {0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8},
     {4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0},
     {15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13}},
    //2 S-box
    {{15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10},
     {3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5},
     {0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15},
     {13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9}},
    //3 S-box
    {{10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8},
     {13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1},
     {13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7},
```

```

    { 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12}},
//4 S-box
{{ 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15},
 {13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9},
 {10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4},
 { 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14}},
//5 S-box
{{ 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9},
 {14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6},
 { 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14},
 {11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3}},
//6 S-box
{{12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11},
 {10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8},
 { 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6},
 { 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13}},
//7 S-box
{{ 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1},
 {13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6},
 { 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2},
 { 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12}},
//8 S-box
{{13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7},
 { 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2},
 { 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8},
 { 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11}}
};

```

```

int P[32] =
    {16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23, 26, 5, 18, 31, 10,
     2, 8, 24, 14, 32, 27, 3, 9, 19, 13, 30, 6, 22, 11, 4, 25};

```

```

void main(){
    //переменные для результатов промежуточных вычислений
    bit lData[32];    //левая половина блока
    bit rData[32];    //правая половина блока
    bit crKey[48];    //перестановка ключа со сжатием
    bit erData[8][6]; //перестановка данных с расширением

```

```

bit sOut[8][4];    //выход S-блоков
bit spOut[32];     //выход P-блока
bit Key[56];       //ключ рауда
//разбиваем блок данных на 2 половины
lData = data;
rData = data >> 32; //сдвиг в сторону младшего бита
//16 этапов алгоритма DES
for(int i = 0; i < 16; i = i + 1){
    //формирование ключа
    key[0] = key[0] <<< key_shift[i];
    key[1] = key[1] <<< key_shift[i];
    Key[56] = key[0] || key[1];
    //перестановка со сжатием
    for(int j = 0; j < 48; j = j + 1)
        crKey[j] = Key[compress_replace[j]];
    //перестановка с расширением
    for(int j = 0; j < 8; j = j + 1)
        for(int s = 0; s < 6; s = s + 1)
            erData[j][s] = rData[exp_replace[8*j + s]]
    erData = erData ^ crKey; //48 новых переменных кода
    //подстановка в S-блоках (32 новых переменных кода)
    for(int j = 0; j < 8; j = j + 1)
        MakeSBox(SBox[j], erData[j], sOut[j]);
    //перестановка в P-блоке
    for(int j = 0; j < 32; j = j + 1)
        spOut[j] = sOut[P[j]];
    spOut = spOut ^ lData; //32 новых переменных кода
    lData = rData; //новый левый блок данных
    rData = spOut; //новый правый блок данных
}
code = lData || rData;
}

```

Приложение Г

ТА-программа, моделирующая функционирование генной сети

```
__in bit v1[3];
__in bit v2[3];
__in bit v3[3];
__out bit v1e[3];
__out bit v2e[3];
__out bit v3e[3];
// первый параметр номер шага, второй параметр индекс компонента
bit v1t[c1+1][3]; bit v2t[c1+1][3]; bit v3t [c1+1][3];
void main(){
    //инициализация
    v1t[0]=v1; v2t[0]=v2; v3t[0]=v3;
    //моделирование динамики функционирования генной сети
    for (int i = 0; i < 3; i = i + 1){
        if ((v2t[i]+v3t[i]==0)&(v1t[i]<p-1)) v1t[i+1]=v1t[i]+1;
        else if ((v2t[i]+v3t[i]>0)&(v1t[i]>0)) v1t[i+1]=v1t[i]-1;
        else v1t[i+1]=v1t[i];
        if ((v1t[i]+v3t[i]==0)&(v2t[i]<p-1)) v2t[i+1]=v2t[i]+1;
        else if ((v1t[i]+v3t[i]>0)&(v2t[i]>0)) v2t[i+1]=v2t[i]-1;
        else v2t[i+1]=v2t[i];
        if ((v1t[i]+v2t[i]==0)&(v3t[i]<p-1)) v3t[i+1]=v3t[i]+1;
        else if ((v1t[i]+v2t[i]>0)&(v3t[i]>0)) v3t[i+1]=v3t[i]-1;
        else v3t[i+1]=v3t[i];
    }
    //заключительное состояние генной сети
    v1e = v1t[c1+1];
    v2e = v2t[c1+1];
    v3e = v3t[c1+1];
}
```