И. В. Отпущенников, А. А. Семёнов, Технология трансляции комбинаторных проблем в булевы уравнения,
ПДМ, 2011, номер 1, 96–115

МАТЕМАТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ И ПРОГРАММИРОВАНИЯ

УДК 519.7

ТЕХНОЛОГИЯ ТРАНСЛЯЦИИ КОМБИНАТОРНЫХ ПРОБЛЕМ В БУЛЕВЫ УРАВНЕНИЯ

И. В. Отпущенников, А. А. Семёнов

Институт динамики систем и теории управления СО РАН, г. Иркутск, Россия

E-mail: otilya@yandex.ru, biclop@rambler.ru

Рассматриваются проблемы сведения некоторых комбинаторных задач к задачам поиска решений булевых уравнений. Приведены теоретические результаты, являющиеся базой технологии пропозиционального кодирования алгоритмов, вычисляющих дискретные функции. Описан программный комплекс Transalg — многофункциональный транслятор комбинаторных проблем в булевы уравнения. Приведены примеры использования комплекса Transalg для сведения задач криптоанализа к SAT-задачам. Рассмотрены основы техники трансляции оптимизационных задач 0-1–ЦЛП в SAT-задачи.

Ключевые слова: *дискретные функции, булевы уравнения, криптоанализ, пропозициональное кодирование.*

Введение

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, требуется находить именно точные решения. Эта необходимость существенно сужает класс методов — методы непрерывной математики, генетические алгоритмы, разнообразные «эволюционные эвристики» в общем случае не дают точных решений.

В настоящей работе приведены основы «пропозиционального подхода» к решению комбинаторных задач из весьма широкого класса. Данный подход предполагает нахождение точных решений и включает две составляющих. Во-первых, это алгоритмы сведения комбинаторных задач к булевым уравнениям, и, во-вторых, это символьные алгоритмы поиска решений получаемых уравнений. В последние годы интерес именно к такому рассмотрению комбинаторных задач заметно усилился в связи с существенным прогрессом в алгоритмике булевых решателей (в первую очередь, SAT-задач), а также в связи с бурным развитием параллельных вычислительных технологий (булевы задачи допускают естественные формы параллелизма).

Библиография по решению булевых уравнений весьма обширна — от фундаментальной монографии С. Рудяну [1] до многочисленных в последние годы работ по SAT-задачам. Работ, специально посвященных сведению комбинаторных проблем к булевым уравнениям, сравнительно мало (можно сослаться на обзорную статью [2] и список литературы к ней). Удивительно то, что в подавляющем большинстве эти результаты имеют характер наглядных примеров и правдоподобных рассуждений — самой строгой в этом смысле продолжает оставаться процедура, фигурирующая в оригинальном и последующих доказательствах теоремы Кука [3, 4]. Следует отметить, что реализовать на основе перечисленных результатов конкретные процедуры сведения, применимые к достаточно широким классам комбинаторных задач, крайне затруднительно: сведения, описанные в [2], слишком специфичны, а известные варианты теоремы Кука доказаны в отношении машины Тьюринга — модели, которая крайне далека от современных ЭВМ в плане языка и организации вычислений.

В работе [5] описаны механизмы пропозиционального кодирования программ, вычисляющих дискретные функции на машинах с произвольным доступом к памяти (RAM). Язык данной модели очень естествен — фактически это фрагмент ассемблера современных ЭВМ. Там же в общих чертах описаны основные принципы высокоуровневой трансляции алгоритмов, вычисляющих дискретные функции, в булевы уравнения. Реализацией этих идей стал программный комплекс Transalg. В настоящей работе описаны его архитектура, функциональные возможности и применение для решения некоторых комбинаторных задач.

В п. 1 содержатся некоторые результаты работы [5]. Эти результаты являются базой для последующего материала. В п. 2 дается подробное описание архитектуры программного комплекса Transalg, а также рассматривается специальный язык программирования TA, используемый для описания алгоритмов вычисления дискретных функций. В п. 3 приводятся результаты использования комплекса Transalg для построения пропозициональных кодов некоторых криптографических алгоритмов. В п. 4 описаны основные принципы сведения задач с псевдобулевыми ограничениями к SAT-задачам.

1. Трансляция формальных программ вычисления дискретных функций в булевы уравнения

1.1. Трансляция RAM-программ

Через $\{0,1\}^n$ обозначается множество всех двоичных слов длины n, а через $\{0,1\}^*$ — множество всех двоичных слов произвольной конечной длины. Дискретными функциями называются произвольные функции вида

$$f : \{0,1\}^* \to \{0,1\}^*.$$

Далее рассмотрим такие дискретные функции, описаниями которых являются программы для детерминированной машины Тьюринга с алфавитом $\{0,1\}$. Пусть f — произвольная такая функция и M_f — вычисляющая ее ДМТ-программа. Дополнительно будем предполагать, что $\operatorname{dom} f = \{0,1\}^*$, то есть M_f останавливается на произвольном двоичном слове, и что сложность M_f растет как некоторый полином с ростом длины входа. Очевидно, что M_f задает счетное семейство функций вида

$$f_n : \{0,1\}^n \to \{0,1\}^*, \quad \operatorname{dom} f_n = \{0,1\}^n, \quad n \in \mathbb{N}.$$

Проблемой обращения произвольной функции f_n из данного семейства в точке $y \in \operatorname{range} f_n$ называется следующая задача: зная M_f, число n и $y \in \operatorname{range} f_n$, найти такой $x \in \{0,1\}^n$, 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 $y \in \operatorname{range} f_n$, означает существование такого входа $x \in \{0,1\}^n$, результат трансформации которого посредством программы M_f есть y. Фактически в этом состоит теорема Кука [3].

Как уже отмечалось, построение процедуры пропозиционального кодирования ДМТ-программ имеет чисто теоретический интерес. Гораздо более близкими к современным реальным вычислителям являются машины с произвольным доступом, впервые описанные в [6]. Далее используем упрощенную (в сравнении с исходной) форму RAM, которой тем не менее оказалось достаточно для построения теории вычислимых (рекурсивных) функций [7].

Итак, далее рассматривается двоичная (бинарная) RAM в формализме Н. Катленда [7]. Данная модель включает потенциально бесконечную вправо ленту, разбитую на ячейки, которые пронумерованы натуральными числами. В каждой ячейке может быть записан только один бит. Произвольная бинарная 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 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.

Пусть f — произвольная всюду определенная (тотальная) на $\{0,1\}^*$ полиномиально вычислимая дискретная функция, рассматриваемая как семейство $f = \{f_n\}_{n\in\mathbb{N}}$, $f_n : \{0,1\}^n \to \{0,1\}^*$, и M_f — ДМТ-программа, вычисляющая f. В соответствии с [4], значение функции сложности $\rho(n)$ программы M_f равно максимуму числа шагов ДМТ, выполняющей M_f, по всевозможным входам из $\{0,1\}^n$. Пусть R_{f_n} — произвольная программа бинарной RAM, вычисляющая функцию f_n. Поставим ей в соответствие значение функции $\vartheta(n)$, равное максимальному по всевозможным входам из $\{0,1\}^n$ числу обращений к регистрам ленты RAM в процессе выполнения R_{f_n}.

Приведем два утверждения из работы [5], являющиеся теоретической базой описываемых далее процедур трансляции алгоритмов.

Лемма 1 (о моделировании). Пусть M_f — ДМТ-программа, вычисляющая тотальную дискретную функцию $f : \{0,1\}^* \to \{0,1\}^*$, 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 $\vartheta(n)$, сопоставляемая получаемому семейству RAM-программ, ограничена сверху полиномом от n.

Данный факт означает наличие эффективной процедуры перехода от ДМТ-программы, вычисляющей f, к семейству двоичных 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 практичных процедур пропозиционального кодирования алгоритмов.

Теорема 1. Пусть $f = \{f_n\}_{n\in\mathbb{N}}$ — семейство алгоритмически вычислимых за полиномиальное время дискретных функций и $\{R_{f_n}\}_{n\in\mathbb{N}}$ — семейство бинарных 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 для произвольного $y \in \operatorname{range} f_n$ система $S(f_n)|_y$ совместна. Если x^* — произвольное решение системы $S(f_n)|_y$, то из x^* за линейное от $|x^*|$ время можно выделить некоторый $x \in \{0,1\}^n$, такой, что $f_n(x) = y$.

Здесь через $S(f_n)|_y$ обозначена система булевых уравнений, которая получается из системы $S(f_n)$ в результате подстановки в нее вектора $y \in \operatorname{range} f_n$.

При доказательстве данной теоремы (см. [5]) в явном виде строится процедура, которая применима к трансляции RAM-программ, вычисляющих произвольные функции из описанного выше класса, в булевы уравнения. Процесс RAM-вычисления при этом рассматривается как последовательность переходов $K_0 \to K_1 \to \ldots \to K_e$, где K_0 — начальная, K_e — конечная, а $K_1, \ldots, K_{e-1}$ — промежуточные конфигурации RAM-вычисления. Каждой конфигурации K_i, $i \in \{0, 1, \ldots, e\}$, сопоставляется множество (массив) булевых переменных X^i, а каждому переходу — система булевых уравнений, связывающих соответствующие соседние массивы. Конъюнкция всех таких систем дает систему $S(f_n)$, кодирующую процесс выполнения программы R_{f_n} на произвольном входе из $\{0,1\}^n$.

От произвольной системы вида $S(f_n)|_y$ возможен эффективный (в общем случае за полиномиальное от n время) переход к одному уравнению вида КНФ $= 1$. Этот переход осуществляется при помощи преобразований Цейтина [8]. Между множеством решений системы $S(f_n)|_y$ и множеством решений получаемого уравнения КНФ $= 1$ существует биекция [9].

1.2. Основные принципы трансляции высокоуровневых программ в булевы уравнения

Близость синтаксиса RAM-программ к современным ассемблерным языкам позволяет рассматривать приведенные выше результаты в качестве идейной основы для разработки процедур трансляции высокоуровневых описаний алгоритмов в булевы уравнения.

Ядром пропозиционального подхода является идея представления булевыми уравнениями переходов из одной конфигурации формальной модели в другую. Для представления рабочей области современной ЭВМ следует использовать булевы массивы. Так же, как и в случае RAM, при реализации алгоритма вычисления дискретной функции на «высокоуровневой модели» каждой новой конфигурации соответствует массив, элементы которого кодируются булевыми переменными. Связь между двумя соседними конфигурациями определяется булевыми уравнениями, описывающими соответствующие зависимости между массивами. В таком представлении программа вычисления дискретной функции состоит только из команд записи битов в ячейки массивов и команд условного перехода. Только в «высокоуровневом случае» булевы переменные в новой конфигурации могут выражаться в виде суперпозиций булевых функций от

многих переменных предыдущей конфигурации. Соответственно и условия перехода могут выражаться сложными булевыми функциями.

Самым нетривиальным моментом (как и выше) является пропозициональное представление условных переходов. Далее будем рассматривать стандартную конструкцию условного оператора:

if (условие) **then** (переход)

else (переход)

При этом ситуация «Условие выполнено» эквивалентна принятию некоторой булевой функцией g значения 1, ситуация «Условие не выполнено» эквивалентно принятию булевой функцией g значения 0. Иными словами,

$$\begin{array}{l}\textbf{if}\quad g(x_1,\ldots,x_k)=1\quad\textbf{then}\quad S\\ \textbf{else}\quad T\end{array} \tag{1}$$

для некоторых булевых переменных $x_1,\ldots,x_k$. Здесь S и T — это либо снова некоторые условные переходы, либо команды, формирующие массив, задающий ту конфигурацию, в которую осуществляется переход.

Пример 1. Предположим, что рассматривается некоторая программа для вычисления функции $f_n:\{0,1\}^n\to\{0,1\}^*$. Примером условного перехода типа (1) может быть следующий фрагмент программы:

$$\begin{array}{l}\textbf{if}\quad (g(y_1^{i-1},\ldots,y_k^{i-1})=1)\quad\textbf{then}\quad (y_t^i:=(y_1^{i-1}\downarrow y_2^{i-1}),\ y_j^i:=y_j^{i-1},\ j\in\{1,\ldots,k\}\backslash\{t\})\\ \textbf{else}\quad (y_t^i:=(y_1^{i-1}\to y_2^{i-1}),\ y_j^i:=y_j^{i-1},\ j\in\{1,\ldots,k\}\backslash\{t\})\end{array}$$

То есть если $g(y_1^{i-1},\ldots,y_k^{i-1})=1$, где y_j^{i-1}, $j\in\{1,\ldots,k\}$ — компоненты массива, соответствующего $(i-1)$-й конфигурации, то в i-й конфигурации значение y_t^i совпадает со значением функции $y_1^{i-1}\downarrow y_2^{i-1}$. В противном случае, то есть если $g(y_1^{i-1},\ldots,y_k^{i-1})=0$, значение y_t^i совпадает со значением функции $y_1^{i-1}\to y_2^{i-1}$. Значения переменных y_j^i, $j\in\{1,\ldots,k\}\backslash\{t\}$, в обоих случаях совпадают со значениями переменных y_j^{i-1}.

Пример 2. Предположим, что вычисляется функция $f_4:\{0,1\}^4\to\{0,1\}^4$, заданная следующей программой:

$$\begin{array}{l}\textbf{if}\quad (x_1\cdot x_2\vee x_3=1)\quad\textbf{then}\quad (y_1:=(x_1\downarrow x_2),\ y_2:=x_2,\ y_3:=x_3,\ y_4:=x_4)\\ \textbf{else}\quad (y_1:=(x_1\to x_2),\ y_2:=x_2,\ y_3:=x_3,\ y_4:=x_4)\end{array}$$

В контексте вышесказанного данной программе сопоставляется следующая система $S(f_4)$:

$$\begin{cases}\Big(y_1\equiv\Big((x_1\downarrow x_2)(x_1\cdot x_2\vee x_3)\vee(x_1\to x_2)\overline{(x_1\cdot x_2\vee x_3)}\Big)\Big)=1,\\ (y_2\equiv x_2)=1,\\ (y_3\equiv x_3)=1,\\ (y_4\equiv x_4)=1.\end{cases}$$

От данной системы переходим к одному уравнению вида КНФ $=1$ при помощи преобразований Цейтина.

Вложенные условные операторы разбираются в соответствии с теми же принципами, что и для RAM-программ.

Пример 3. Рассмотрим программу

$$\begin{array}{l}\textbf{if}\quad g(x_{i_1},\ldots,x_{i_k})=1\\ \quad\textbf{then}\quad S\\ \textbf{else if}\quad h(x_{j_1},\ldots,x_{j_l})=1\\ \qquad\textbf{then}\quad T\\ \quad\textbf{else}\quad U\end{array} \tag{2}$$

В соответствии с описанными ранее принципами в процессе трансляции данного оператора объявляется массив булевых переменных $y_1, \ldots, y_r$, для значений которых имеются три альтернативы, определяемые блоками команд S, T и U. Предположим, что значения переменных y_i, $i \in \{1, \ldots, r\}$, в блоке S определяются функциями $\lambda_i^S(x_1, \ldots, x_n)$, в блоке T — функциями $\mu_i^T(x_1, \ldots, x_n)$, в блоке U — функциями $\nu_i^U(x_1, \ldots, x_n)$. Сказанное означает, что результатом трансляции программы (2) является следующая система булевых уравнений:

$$\begin{cases} \Big(y_1 \equiv g(\ldots)\lambda_1^S(\ldots) \vee \overline{g(\ldots)}h(\ldots)\mu_1^T(\ldots) \vee \overline{g(\ldots)h(\ldots)}\nu_1^U(\ldots)\Big) = 1, \\ \qquad \ldots \\ \Big(y_r \equiv g(\ldots)\lambda_r^S(\ldots) \vee \overline{g(\ldots)}h(\ldots)\mu_r^T(\ldots) \vee \overline{g(\ldots)h(\ldots)}\nu_r^U(\ldots)\Big) = 1. \end{cases}$$

Рассмотренные принципы кодирования применимы к произвольным программам, вычисляющим всюду определенные дискретные функции, написанным на высокоуровневых языках программирования. При этом само вычисление интерпретируется последовательностью переписывающихся булевых массивов — элементы последующих массивов задаются булевыми функциями от элементов предыдущих массивов, а формулам, реализующим эти функции, сопоставляются булевы уравнения, образующие системы типа $S(f_n)$.

2. Архитектура и функциональные возможности программного комплекса Transalg

Размерности систем булевых уравнений, которые кодируют комбинаторные задачи, возникающие в практических приложениях, таковы (иногда это десятки мегабайт), что процесс их построения требует автоматизации. Далее описаны основные элементы программного комплекса (транслятора) Transalg, который стал практической реализацией перечисленных выше идей. Транслятор Transalg получает на входе программу вычисления некоторой дискретной функции, записанную на специальном процедурном языке программирования (язык ТА). Результатом трансляции ТА-программы является система булевых уравнений, кодирующая процесс вычисления рассматриваемой функции (система $S(f_n)$).

2.1. Базовые механизмы трансляции ТА-программ

Схематично процесс трансляции ТА-программ представлен на рис. 1.

Фазы анализа текста ТА-программы, построения дерева синтаксического разбора и обход полученного дерева с целью интерпретации реализованы стандартным способом (см., например, [10]). Нетривиальным моментом трансляции является процедура интерпретации конструкций языка, поскольку именно она отвечает за построение системы булевых уравнений, кодирующей процесс выполнения алгоритма. На этом этапе возникают многочисленные локальные проблемы, от решения которых может существенно зависеть как объем кода (в смысле общего числа задействованных в нем булевых переменных и количества уравнений), так и его структура.

Язык ТА представляет собой процедурный язык программирования 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 (объявление/определение переменной или массива переменных; определение именованных констант; оператор присваивания; составной опе-

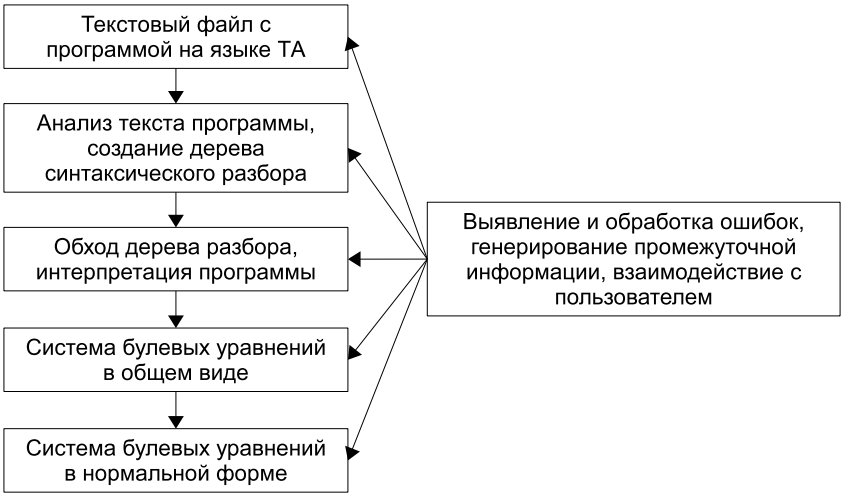

Рис. 1. Общая схема работы комплекса Transalg

ратор; условный переход; цикл; определение пользовательской функции; возврат из функции; вызов функции).

Переменные целочисленных типов хранят служебные параметры транслируемой программы. Например, это могут быть длины входного и выходного слов, количество итераций в циклах, целочисленные константы, используемые при вычислении дискретной функции.

Тип данных `bit` используется для объявления булевых переменных, кодирующих входную/выходную информацию транслируемой программы, а также информацию, возникающую в процессе работы этой программы. Речь идет о переменных, играющих ту же роль, что и переменные, кодирующие содержимое конфигураций $K_0, K_1, \ldots, K_e$ RAM (см. п. 1). Кроме этого, тип `bit` могут иметь переменные, используемые в тексте программы в качестве вспомогательных для хранения результатов промежуточных вычислений.

Действия с памятью, которые выполняются в любом современном вычислительном устройстве, аналогичны действиям с регистрами RAM. Далее будем рассматривать вычисление, которое осуществляет транслируемая программа, как последовательность изменений данных в памяти вычислительного устройства в моменты времени $0, 1, \ldots, e$. В каждый момент времени i, $i \in \{0, \ldots, e\}$, данные в памяти кодируются булевыми переменными, образующими множество X^i. Таким образом, множество X^0 образовано булевыми переменными, кодирующими входные данные, а множество X^e — переменными, кодирующими выходные данные рассматриваемого дискретного преобразования.

Особо подчеркнем, что переменные транслируемой ТА-программы и переменные пропозиционального кода этой программы представляют разные сущности. Переменные, фигурирующие в тексте транслируемой ТА-программы (далее «переменные программы»), понимаются в традиционном смысле — это идентификаторы областей памяти. Переменные, попадающие в пропозициональный код (далее «переменные кода»), понимаются как символы некоторого конечного алфавита. По своему смыслу переменные кода — это переменные итоговой системы булевых уравнений. Тем не менее эти два вида переменных тесно связаны. Каждой переменной программы соответствует специальная структура данных `var_object`. Эта структура хранит информацию о связи переменной кода с переменной программы на некотором шаге вычисления. При ин-

терпретации инструкций, содержащих переменные программы, транслятор проверяет соответствующие структуры `var_object` на наличие в них связи с переменными кода.

Переменные программы требуется различать по семантике. Для этой цели в языке при объявлении переменных типа `bit` используются специальные атрибуты. Атрибут `_in` сообщает транслятору, что данная переменная программы связана (посредством структуры `var_object`) с переменной кода, которая входит в множество X^0. Эта связь осуществляется транслятором на начальном шаге (шаг инициализации). Атрибут `_out` используется для выделения переменных программы, связанных посредством `var_object` с переменными кода из множества X^e. Переменные с атрибутами `_in` или `_out` должны иметь глобальную область видимости, поскольку они сообщают транслятору информацию о входах и выходах кодируемого дискретного преобразования.

Кроме перечисленных, в тексте ТА-программы могут встречаться переменные, необходимые для хранения результатов промежуточных вычислений. Структура `var_object` таких переменных не связывает их с переменными кода. Далее такие переменные называются фиктивными. Проиллюстрируем все сказанное на следующем примере.

Пример 4. Рассмотрим ТА-программу, которая реализует (моделирует) изображённый на рис. 2 РСЛОС — регистр сдвига с линейной обратной связью [11], заданной полиномом над GF(2) $P(x) = x^{19} + x^{18} + x^{17} + x^{14} + 1$.

```
1  _in bit reg[19];
2  _out bit output[100];
3  bit shiftReg(){
4      bit x = reg[18];
5      bit y = reg[18]^reg[17]^reg[16]^reg[13];
6      for(int j = 18; j > 0; j = j - 1){
7          reg[j] = reg[j - 1];
8      }
9      reg[0] = y;
10     return x;
11 }
12 void main(){
13     for(int i = 0; i < 100; i = i + 1){
14         output[i] = shiftReg();
15     }
16 }
```

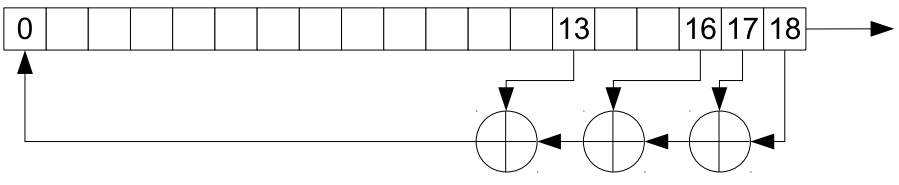

Рис. 2. РСЛОС с полиномом обратной связи $P(x) = x^{19} + x^{18} + x^{17} + x^{14} + 1$

Массив булевых переменных `reg` описывает в каждый фиксированный момент времени 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 $X^0 = \{x_1, \ldots, x_{19}\}$.

Представленная программа реализует 100 тактов работы регистра сдвига. В теле основной функции `main()` организован цикл, в котором вызывается функция сдвига регистра `shiftReg()`. Значения, возвращаемые функцией `shiftReg()`, определяют биты выходного слова, которое представлено в программе массивом `output`.

Сдвиг регистра обновляет значения всех элементов массива `reg`. На первом шаге данная операция приводит к вводу новых переменных кода, образующих множество $X^1 = \{x_{20}, \ldots, x_{38}\}$. Эти переменные связываются через структуры `var_object` с переменными программы `reg[0], ..., reg[18]`. Данная связь интерпретируется следующей системой булевых уравнений:

$$\begin{cases} (x_{20} \equiv x_{19} \oplus x_{18} \oplus x_{17} \oplus x_{14}) = 1, \\ (x_{21} \equiv x_1) = 1, \\ (x_{22} \equiv x_2) = 1, \\ \quad \ldots \\ (x_{38} \equiv x_{18}) = 1. \end{cases} \tag{3}$$

Переменная x_{19} кодирует первый бит ключевого потока, полученный в результате первого сдвига рассматриваемого РСЛОС. Отметим, что локальные переменные x и y функции `shiftReg()` необходимы лишь для корректной организации вычислений и не связаны с переменными кода программы. В контексте вышесказанного x и y — фиктивные переменные.

Можно заметить, что пропозициональный код, представляемый системой (3), является избыточным, поскольку переменные x_1 и x_{21} кодируют одну и ту же информацию. Аналогично для переменных x_2 и x_{22},..., x_{18} и x_{38}. Избежать подобных ситуаций позволяет описываемая далее техника, использующая при интерпретации инструкций ТА-программы специальный словарь термов S. Данный словарь образован термами над переменными кода транслируемой ТА-программы. Словарь S является динамически расширяемым. В начальном состоянии в S находятся только переменные множества X^0 (то есть переменные, кодирующие входную информацию). В дальнейшем каждый новый терм, попадающий в словарь, является результатом интерпретации транслятором некоторой операции присваивания следующего вида:

$$z = \Phi(z_{j_1}, \ldots, z_{j_r}).$$

Здесь z — переменная программы, которая не является фиктивной, то есть связана через структуру `var_object` с некоторой переменной кода, а $\Phi(z_{j_1}, \ldots, z_{j_r})$ — терм над переменными программы. Транслятор обрабатывает терм $\Phi(z_{j_1}, \ldots, z_{j_r})$ и на его основе формирует терм $\varphi(x_{j_1}, \ldots, x_{j_r})$ над переменными кода. Затем транслятор проверяет словарь S на наличие в нем построенного терма. Если такой терм в словаре не найден, транслятор создает новую переменную кода $\tilde{x}$, терм $\varphi(x_{j_1}, \ldots, x_{j_r})$ добавляет в словарь S, а имеющаяся система булевых уравнений дополняется новым уравнением вида

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

Если терм $\varphi(x_{j_1}, \ldots, x_{j_r})$ содержится в S, это означает, что кодируемая этим термом информация уже учтена в пропозициональном коде транслируемой программы, то есть в системе булевых уравнений присутствует уравнение вида

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

В этом случае транслятор связывает переменную программы z с переменной кода x' при помощи структуры `var_object`. Именно этот прием позволяет избегать ввода переменных кода, кодирующих одну и ту же информацию. Поясним сказанное на примере.

Пример 5. Рассмотрим процесс трансляции ТА-программы из примера 4. На начальном шаге трансляции в словарь термов включаются переменные кода, образующие множество $X^0 = \{x_1, \ldots, x_{19}\}$. При этом связи между переменными программы, объединенными в массив `reg`, и переменными кода из множества X^0 выглядят следующим образом (рис. 3):

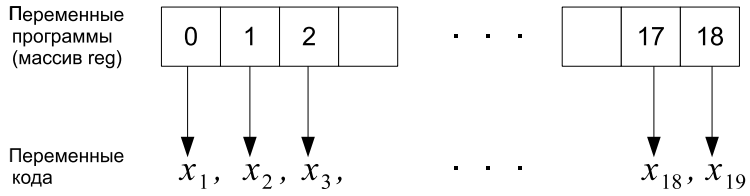

Рис. 3. Связь переменных программы с переменными кода до сдвига регистра

Интерпретируя сдвиг регистра, транслятор добавляет в словарь S терм $\varphi = x_{19} \oplus \oplus x_{18} \oplus x_{17} \oplus x_{14}$, кодирующий функцию обратной связи, создает новую переменную кода x_{20} и добавляет в систему булевых уравнений уравнение

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

Кроме этого, транслятор обновляет связи между элементами массива `reg` и переменными кода (рис. 4).

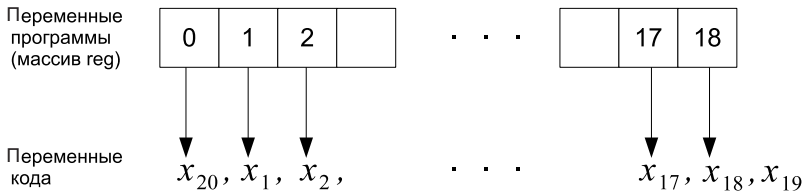

Рис. 4. Связь переменных программы с переменными кода после сдвига регистра

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

2.2. Трансляция условных переходов

В программном комплексе Transalg интерпретация условного оператора начинается с анализа условного выражения. Условный оператор — это оператор вида

if $\Phi(z_{j_1}, \ldots, z_{j_r}) = 1$ **then** Ветвь 1;

else Ветвь 2;

где терм условного выражения $\Phi(z_{j_1}, \ldots, z_{j_r})$ — терм над переменными программы. Если в терм условного выражения входят переменные программы, связанные посредством структур `var_object` с переменными кода, то транслятор переходит в режим ветвления. Первое действие транслятора в этом режиме заключается в интерпретации терма условного выражения Φ, результатом чего является терм φ над переменными кода.

Ветвью условного оператора может быть произвольный блок операторов. Интерпретация ветви — это последовательная интерпретация всех операторов соответствующего блока. Особо отметим, что одна и та же переменная программы может фигурировать в различных ветвях условного оператора.

Рассмотрим условный оператор, имеющий две ветви условного перехода. Пусть в обеих ветвях данного оператора выполняется некоторая операция присваивания, и z — переменная программы, являющаяся левым операндом этой операции. Пусть Δ_1 и Δ_2 — термы над переменными программы, являющиеся правыми операндами данной операции присваивания соответственно в 1-й и 2-й ветвях. В этой ситуации транслятор связывает переменную z с новой переменной кода $\tilde{x}$, в словарь S добавляется терм

$$\varphi \cdot \delta_1 \vee \bar{\varphi} \cdot \delta_2,$$

а в систему булевых уравнений — уравнение

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

Здесь δ_1 и δ_2 — термы над переменными кода, полученные в результате интерпретации термов Δ_1 и Δ_2 соответственно. Для хранения пар вида (z, δ_1) и (z, δ_2) используются отдельные списки L_1 и L_2.

Далее рассматривается конструкция из нескольких вложенных условных операторов:

if $\Phi_1(\ldots) = 1$ **then** Ветвь 1;
else if $\Phi_2(\ldots) = 1$ **then** Ветвь 2;
...
else if $\Phi_n(\ldots) = 1$ **then** Ветвь n;
else Ветвь $n + 1$;

В соответствии со сказанным выше, каждому терму Φ_i, $i = 1, \ldots, n$, сопоставляется терм φ_i над множеством переменных кода. Каждой ветви с номером $i \in \{1, \ldots, n+1\}$ ставится в соответствие список L_i, образованный парами вида (z, δ_i). Пусть z — переменная программы, выступающая в качестве левого операнда операций присваивания в ветвях с номерами от 1 до $n+1$ рассматриваемого условного оператора. Тогда транслятор связывает переменную z с новой переменной кода $\tilde{x}$, в словарь S добавляется терм

$$\varphi_1 \cdot \delta_1 \vee \bar{\varphi}_1 \cdot \varphi_2 \cdot \delta_2 \vee \ldots \vee \bar{\varphi}_1 \cdot \bar{\varphi}_2 \cdot \ldots \cdot \bar{\varphi}_{n-1} \cdot \varphi_n \cdot \delta_n \vee \bar{\varphi}_1 \cdot \ldots \cdot \bar{\varphi}_n \cdot \delta_{n+1},$$

а в систему булевых уравнений — уравнение

$$(\tilde{x} \equiv \varphi_1 \cdot \delta_1 \vee \bar{\varphi}_1 \cdot \varphi_2 \cdot \delta_2 \vee \ldots \vee \bar{\varphi}_1 \cdot \bar{\varphi}_2 \cdot \ldots \cdot \bar{\varphi}_{n-1} \cdot \varphi_n \cdot \delta_n \vee \bar{\varphi}_1 \cdot \ldots \cdot \bar{\varphi}_n \cdot \delta_{n+1}) = 1.$$

Проиллюстрируем сказанное на примере.

Пример 6. Рассматриваются два РСЛОС, обозначаемые через `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 5, поэтому определения функций `shiftRegA()` и `shiftRegB()` в следующей ТА-программе опущены.

```
 1 _in bit regA[19];
 2 _in bit regB[17];
 3 _out bit output[100];
 4 bit shiftRegA(){...}
 5 bit shiftRegB(){...}
 6 bit conditionA(){return regA[18] ^ regB[16];}
 7 bit conditionB(){return regA[18] & regB[16];}
 8 void main(){
 9     for(int i = 0; i < 100; i = i + 1){
10         if(conditionA())
11             shiftRegA();
12         else if(conditionB())
13             shiftRegB();
14         else{
15             shiftRegA();
16             shiftRegB();
17         }
18         output[i] = regA[18] ^ regB[16];
19     }
20 }
```

На начальном шаге транслятор формирует множество переменных кода

$$X^0 = \{\underbrace{x_1, \ldots, x_{19}}_{\texttt{regA}}, \underbrace{x_{20}, \ldots, x_{36}}_{\texttt{regB}}\}.$$

Данные переменные кодируют входную информацию рассматриваемой дискретной функции.

При интерпретации условных выражений транслятор обнаруживает (анализируя структуры `var_object`), что входящие в эти выражения переменные программы `regA[18]` и `regB[16]` связаны с переменными кода. В результате транслятор переходит в режим ветвления. Интерпретация условных выражений в операторе ветвления дает термы над переменными кода

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

В ходе интерпретации ветвей условного перехода формируются списки L_i, $i \in \{1, 2, 3\}$. Обозначим через δ_1 и δ_2 термы, полученные в результате интерпретации термов программы, выражающих функции обратной связи регистров `regA` и `regB`. В соответствии со сказанным выше, для рассматриваемого оператора ветвления транслятор формирует структуру данных, изображенную на рис. 5 (связи переменных программы с переменными кода обновляются в соответствии с описанной ранее техникой).

Рассмотренная схема дает эффективную процедуру связывания переменных программы с переменными кода при трансляции операторов условного перехода. Например, переменная `regA[0]` в результате трансляции будет связана с новой переменной кода x_{37}, в словарь S будет добавлен терм

$$\varphi_1 \cdot \delta_1 \vee \bar{\varphi}_1 \cdot \varphi_2 \cdot x_1 \vee \bar{\varphi}_1 \cdot \bar{\varphi}_2 \cdot \delta_1,$$

а в систему булевых уравнений — уравнение

$$(x_{37} \equiv \varphi_1 \cdot \delta_1 \vee \bar{\varphi}_1 \cdot \varphi_2 \cdot x_1 \vee \bar{\varphi}_1 \cdot \bar{\varphi}_2 \cdot \delta_1) = 1.$$

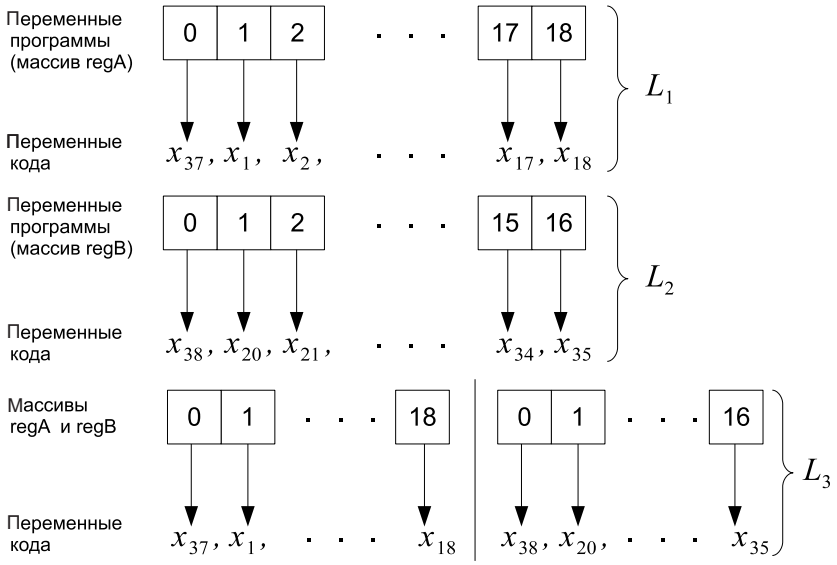

Рис. 5. Вспомогательные структуры данных, представляющие условный переход

2.3. Кодирование целочисленных операций

В комплексе Transalg предусмотрена возможность пропозиционального кодирования целочисленных операций. При трансляции целые числа представляются булевыми массивами. Однако в тексте ТА-программы присутствуют лишь операции над буквенными обозначениями чисел, что позволяет приблизить синтаксис выражений языка ТА к синтаксису выражений, общепринятому в процедурном программировании.

Пример 7. Рассмотрим следующую ТА-программу (n — произвольная натуральная константа):

```
1 _in bit a[n];
2 _in bit b[n];
3 _out bit c[n+1];
4 void main(){
5     c = a + b;
6 }
```

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. Для представления целых чисел a, b и c объявляются булевы массивы $a = \{a_0, \ldots, a_{n-1}\}$, $b = \{b_0, \ldots, b_{n-1}\}$ и $c = \{c_0, \ldots, c_n\}$. Далее приведен результат трансляции данной программы в предположении, что для сложения двух неотрицательных целых чисел используется алгоритм «столбик»:

$$\begin{cases} (c_0 \equiv a_0 \oplus b_0) = 1, \\ (p_0 \equiv a_0 \cdot b_0) = 1, \\ (p_j \equiv \mathrm{maj}(a_j,\, b_j,\, p_{j-1})) = 1, \quad j = \overline{1,\, n-1}, \\ (c_i \equiv a_i \oplus b_i \oplus p_{i-1}) = 1, \quad i = \overline{1,\, n-1}, \\ (c_n \equiv p_{n-1}) = 1. \end{cases} \tag{4}$$

Запись $\mathrm{maj}(x, y, z)$ обозначает терм $x \cdot y \vee x \cdot z \vee y \cdot z$. Для кодирования битов переноса транслятор вводит новые переменные кода p_i, $i = \overline{0,\, n-1}$. В трансляторе Transalg

реализована также возможность кодирования операций умножения, вычитания и сравнения пар целых чисел.

Как было отмечено в п. 1.1, для использования эффективных символьных решателей предпочтительно от полученной в результате трансляции алгоритма системы булевых уравнений перейти к некоторой нормальной форме. Наилучшие результаты на обширных классах практических задач показывают SAT-решатели, то есть решатели булевых уравнений вида КНФ $= 1$. В трансляторе Transalg переход от систем булевых уравнений описанного типа к уравнениям вида КНФ $= 1$ осуществляется при помощи преобразований Цейтина [8], дополненных процедурами минимизации булевых функций в классе КНФ. Кроме этого, возможен вывод пропозиционального кода алгоритма в форме систем полиномиальных уравнений над GF(2). Предусмотрена также возможность построения И-НЕ-графа, фактически представляющего алгоритм вычисления рассматриваемой функции в форме схемы из функциональных элементов над базисом $\{\&, \neg\}$.

3. Применение комплекса Transalg для пропозиционального кодирования алгоритмов вычисления некоторых криптографических функций

Одной из наиболее наглядных областей применения описанной техники трансляции ТА-программ является криптография. Далее разбираются примеры построения пропозициональных кодов некоторых криптографических алгоритмов. Для ряда криптосистем данный подход позволил успешно решить задачи криптоанализа.

В работе [12] 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 [13].

Логический криптоанализ оказался эффективным в применении к некоторым генераторам ключевого потока [14, 15]. Быстрые генераторы поточного шифрования — это эффективно вычислимые дискретные функции, преобразующие двоичные последовательности конечной длины (инициализирующие последовательности) в бесконечные периодические двоичные последовательности (ключевой поток). Задача криптоанализа генератора ключевого потока заключается в нахождении инициализирующей последовательности по известному фрагменту ключевого потока и алгоритму функционирования генератора.

Программный комплекс Transalg по известному алгоритму генерации ключевого потока позволяет построить систему булевых уравнений, кодирующих процесс порождения произвольного фрагмента ключевого потока. Подстановка в полученную систему анализируемого фрагмента ключевого потока дает систему булевых уравнений, из решения которой можно эффективно выделить искомый секретный ключ (инициализирующую последовательность).

3.1. Кодирование генераторов поточного шифрования

Рассмотрим задачу пропозиционального кодирования генераторов поточного шифрования на примере суммирующего генератора Р. Рюппеля [16] (см. также [11]). Данный генератор состоит из $R \geqslant 2$ РСЛОС и специального регистра, называемого сумматором. 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 нату-

рального числа C_0 (соответствующие биты образуют часть секретного ключа). В каждый такт с номером τ, $\tau \in \{1, 2, \ldots\}$, 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, $\tau \in \{1, 2, \ldots\}$, является младший бит двоичного представления числа S_τ. Значением сумматора в такте с номером τ, $\tau \in \{1, 2, \ldots\}$, является двоичное представление числа $\lfloor S_\tau/2 \rfloor$.

Рассмотрим суммирующий генератор, состоящий из трех РСЛОС ($R = 3$), функции обратной связи которых задаются следующими полиномами над GF(2):

$$P_1(x) = x^{19} + x^{18} + x^{17} + x^{14} + 1;\ P_2(x) = x^{22} + x^{21} + 1;\ P_3(x) = x^{23} + x^{22} + x^{21} + x^8 + 1.$$

Запишем программу для комплекса Transalg, вычисляющую 180 бит ключевого потока для заданного генератора суммирования (функции `shiftRegA()`, `shiftRegB()` и `shiftRegC()` определяются по аналогии с функцией `shiftReg()` из примера 5):

```
1  _in bit regA[19];
2  _in bit regB[22];
3  _in bit regC[23];
4  _in bit summator[2];
5  _out bit output[180];
6  void main(){
7      for(int i = 0; i < 180; i = i + 1){
8          summator = shiftRegA() + shiftRegB()
9                                  + shiftRegC() + summator;
10         output[i] = summator[0];
11         summator = summator >> 1;
12     }
13 }
```

Объявления булевых массивов с атрибутами `_in` и `_out` позволяют транслятору создать множество переменных $X^0 = \{x_1, \ldots, x_{66}\}$, кодирующих секретный ключ, и множество переменных $X^e = \{x_{i_1}, \ldots, x_{i_{180}}\}$, кодирующих 180-битовый начальный фрагмент ключевого потока. Пропозициональное кодирование процедур сдвига регистров осуществляется по схеме, описанной в п. 2.

Результатом трансляции первого такта работы суммирующего генератора является следующая система булевых уравнений:

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

Здесь переменные y_1, y_2, y_3 — это новые переменные кода, связанные 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 пять новых булевых уравнений и пять дополнительных переменных.

Результатом трансляции 180 тактов работы данного генератора является система из 900 булевых уравнений от 966 булевых переменных. Эта система посредством преобразований Цейтина сводится к уравнению вида КНФ = 1. Результирующая КНФ состоит из 11532 дизъюнктов над множеством из 966 булевых переменных.

3.2. Кодирование криптоалгоритма DES

Алгоритм DES является симметричным блочным алгоритмом шифрования, построенным на основе сети Фейстеля, с 56-битовым секретным ключом. Данный алгоритм фигурирует в массе источников (см., например, [11, 17]), поэтому его описание здесь не приводится. Первой работой, в которой был приведен пропозициональный код DES, является препринт [18]. Журнальный вариант данной работы появился годом позже [13]. Эти статьи (наряду с [12]) следует считать основополагающими работами по логическому криптоанализу.

Одними из базовых примитивов шифра DES являются перестановки. Перестановки в DES задаются таблицами натуральных чисел, которые не являются секретными. Произвольная перестановка применяется к некоторому множеству бит обрабатываемого слова. При пропозициональном кодировании операции перестановки транслятор Transalg не создает новых переменных кода — как видно из п. 2.1, транслятору достаточно обновить связи переменных программы с уже существующими элементами словаря термов S. Данный факт означает, что операции перестановки не вносят в пропозициональный код алгоритма шифрования новой информации, никак не усложняя, таким образом, задачу логического криптоанализа.

В результате применения транслятора Transalg к алгоритму DES был получен пропозициональный код данного алгоритма (в форме КНФ), существенно более экономный, чем код, приведенный в [13, 18] (см. табл. 1).

Таблица 1

Кодирование процесса шифрования алгоритмом DES одного блока открытого текста длиной 64 бита

Программный комплекс Transalg				F. Massacci, L. Marraro, [13]	
Без минимизации		Минимизация (Espresso, [19])			
Переменные	Дизъюнкты	Переменные	Дизъюнкты	Переменные	Дизъюнкты
1912	37888	1912	26400	10336	61935

4. Процедуры сведения задач 0-1-ЦЛП 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 «псевдобулевыми ограничениями». Данная тематика активно развивалась в 2006–2008 гг. Сейчас наблюдается заметный спад этой активности, однако продолжают проводиться соревнования [20] и даже регулярно проходят специализированные конференции, посвященные «псевдобулевым решателям». Работа [21] является, пожалуй, наиболее полным введением в проблему трансляции псевдобулевых задач в SAT.

Приведём исходную постановку задачи 0-1-целочисленного линейного программирования (0-1-ЦЛП), а также кратко опишем базовые принципы трансляции псевдобулевых ограничений в булевы.

Рассмотрим задачу 0-1-ЦЛП в стандартной постановке: дана система ограничений-неравенств

$$A \cdot x \leqslant b, \tag{5}$$

где A — $(m \times n)$-матрица с целочисленными компонентами; b — вектор длины m, состоящий из целых чисел. Предполагается, что переменные x_i, $i \in \{1, \ldots, n\}$, принимают значения в множестве целых чисел $\{0, 1\}$. Множество решений (5) называется допустимым множеством. Распознавательный вариант задачи 0-1-ЦЛП подразумевает ответ на вопрос «Верно ли, что допустимое множество не пусто?» В оптимизационном варианте требуется минимизировать на допустимом множестве целевую функцию — линейную форму $\langle c, x\rangle$, где c — заданный целочисленный вектор длины n.

Процесс сведения данной задачи к SAT состоит в преобразовании линейных неравенств, образующих систему (5), в конъюнкции дизъюнктов. При этом можно использовать эквивалентные преобразования исходных ограничений, приводящие к ограничениям вида

$$\langle a', x^\sigma\rangle \leqslant b_0, \tag{6}$$

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

Пример 8. Рассмотрим ограничение $3x_1 - 2x_2 + 5x_3 \leqslant 3$. Результатом замены $x_2 = 1 - \bar{x}_2$ является ограничение $3x_1 + 2\bar{x}_2 + 5x_3 \leqslant 5$.

Сказанное означает, что от исходной задачи возможен эффективный переход к задаче минимизации линейной формы $\langle c', x^\sigma\rangle$ при условии выполнения m ограничений вида (6).

Задачи 0-1-ЦЛП в последние годы называют также задачами с псевдобулевыми ограничениями. Этим подчеркивается, что переменные задачи принимают значения в $\{0, 1\}$, однако это не значения истинности, а целые числа. Вообще, термины «псевдобулева задача», «псевдобулево ограничение» и т. п. более правомерны в отношении формулировок вида (6). Именно в таком контексте они и используются далее.

Очевидно, что неравенство

$$a_1 x_1^{\sigma_1} + \ldots + a_n x_n^{\sigma_n} \leqslant 0$$

в предположении, что $a_i \neq 0$, $i \in \{1, \ldots, n\}$, а x_i принимают значения в $\{0, 1\}$, выполняется лишь при $x_1^{\sigma_1} = \ldots = x_n^{\sigma_n} = 0$. Поэтому далее рассматриваем псевдобулевы ограничения вида

$$a_1 x_1^{\sigma_1} + \ldots + a_n x_n^{\sigma_n} \leqslant b, \tag{7}$$

полагая, что a_i, $i \in \{1, \ldots, n\}$, b — натуральные числа.

Произвольному целочисленному вектору с компонентами из $\{0, 1\}$ естественным образом сопоставляется внешне ничем от него не отличающийся булев вектор. В этом смысле произвольному ограничению (7) соответствует булева функция, которая принимает значение «истина» тогда и только тогда, когда выполняется (7).

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

$$A(ax^\sigma) = (A_k \ldots A_0)$$

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

Пример 9. В соответствии с описанными правилами выражению $5\bar{x}_1$ сопоставляется слово $(\bar{x}_1 0 \bar{x}_1)$ (число 5 в двоичном представлении задается вектором (101)).

Линейной форме $a_1 x_1^{\sigma_1} + \ldots + a_n x_n^{\sigma_n}$ ставится в соответствие система булевых уравнений, связывающая компоненты слов $A(a_1 x_1^{\sigma_1}), \ldots, A(a_n x_n^{\sigma_n})$. Фактически данная система уравнений описывает процесс суммирования натуральных чисел, представленных двоичными векторами. Так, если $A = (A_k \ldots A_0)$ и $B = (B_k \ldots B_0)$ — построенные по описанным правилам слова, которые соответствуют выражениям $a_1 x_1^{\sigma_1}$ и $a_2 x_2^{\sigma_2}$, то связывающая компоненты слов A и B система булевых уравнений будет аналогична системе (4).

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

От систем типа $S_b(a_1 x_1^{\sigma_1} + \ldots + a_n x_n^{\sigma_n})$ переходим к уравнениям вида КНФ $= 1$ при помощи преобразований Цейтина. Итогом описанных действий является такая КНФ $C(A, b)$, что между множеством решений системы (5) 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, $x^0 = (\alpha_1^0, \ldots, \alpha_n^0)$ — некоторая его точка и $R = \sum\limits_{i=1}^{n} c_i \alpha_i^0$ — значение целевой функции в данной точке. Чтобы попытаться улучшить значение целевой функции, можно добавить к системе ограничений (5) условие в форме линейного неравенства. Это могут быть неравенства вида $\langle c, x \rangle \leqslant R - 1$, $\langle c, x \rangle \leqslant \lfloor R/2 \rfloor$ и т. п., все зависит от выбранной схемы поиска. От произвольного такого неравенства возможен переход к уравнению вида КНФ $= 1$ в соответствии с описанной выше техникой. Таким образом, оптимизационный вариант задачи 0-1-ЦЛП может быть сведен к решению серии SAT-задач. При этом если R — некоторое начальное приближение, то использование дихотомии гарантирует нахождение оптимального значения целевой функции $\langle x, c \rangle$ на допустимом множестве, определяемом системой ограничений (5), за $\mathrm{O}(\log R)$ итераций.

Описанные преобразования псевдобулевых ограничений в булевы были реализованы в виде дополнительного модуля к транслятору Transalg. В табл. 2 приведено сравнение результатов трансляции псевдобулевых задач (эти задачи были взяты из библиотеки [22]) в SAT-задачи. Сравниваются КНФ, полученные транслятором Transalg, и КНФ, полученные известной программой трансляции псевдобулевых задач MiniSat+ (см. [23]).

Таблица 2

Сравнение результатов трансляции псевдобулевых задач в SAT

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

Заключение

Описанная технология пропозиционального кодирования алгоритмов может применяться при исследовании разнообразных систем, поведение которых описывается полиномиально вычислимыми дискретными функциями. Сказанное касается, прежде всего, дискретно-автоматных динамических систем — переходы в последующие состояния в таких системах происходят в дискретные моменты времени, и довольно часто функции, задающие эти переходы, оказываются эффективно вычислимыми. Устанавливать некоторые свойства такого рода систем можно, транслируя алгоритмы вычисления функций переходов в булевы уравнения и добавляя при необходимости к получаемым системам дополнительные ограничения. Трансляция соответствующих алгоритмов может осуществляться при помощи программного комплекса Transalg. Предполагаем, что данный подход окажется полезным в исследовании динамических свойств дискретных моделей генных сетей [24], а также при решении задач из области «Bounded Model Checking» [25].

ЛИТЕРАТУРА

1. *Rudeanu S.* Boolean functions and equations. Amsterdam; London: North-Holland Publishing Company, 1974. 442 p.
2. *Prestwich S.* CNF encodings. In Handbook of Satisfiability / eds. A. Biere, M. Heule, H. van Maaren, T. Walsh. IOS Press, 2009. P. 75–97.
3. *Cook S. A.* The complexity of theorem-proving procedures // Proc. 3rd Ann. ACM Symp. on Theory of Computing (STOC 71). ACM. 1971. P. 151–159.
4. *Garey M. R. and Johnson S.* Computers and intractability: A guide to the theory of NP-completeness. W. H. Freeman, 1979. 340 p.
5. *Семёнов А. А.* Трансляция алгоритмов вычисления дискретных функций в выражения пропозициональной логики // Прикладные алгоритмы в дискретном анализе. Сер. Дискретный анализ и информатика. 2008. Вып. 2. С. 70–98.
6. *Shepherdson J. C. and Sturgis H. E.* Computability of recursive functions // J. Assoc. Comp. Machinery. 1963. V. 10. P. 217–255.
7. *Катленд Н.* Вычислимость. Введение в теорию рекурсивных функций. М.: Мир, 1983. 256 с.
8. *Цейтин Г. С.* О сложности вывода в исчислении высказываний // Записки научных семинаров ЛОМИ АН СССР. 1968. Т. 8. С. 234–259.

9. *Семёнов А. А.* О преобразованиях Цейтина в логических уравнениях // Прикладная дискретная математика. 2009. № 4. С. 28–50.
10. *Ахо А., Сети Р., Ульман Дж.* Компиляторы. Принципы, технологии, инструменты. М.; СПб.; Киев: Вильямс, 2001. 768 с.
11. *Menezes A., Oorschot P., and Vanstone S.* Handbook of Applied Cryptography. CRC Press, 1996. 657 p.
12. *Cook S. A. and Mitchel G.* Finding hard instances of the satisfiability problem: A survey // DIMACS Ser. Discr. Mathem. Theoret. Comp. Scie. 1997. V. 35. P. 1–17.
13. *Massacci F. and Marraro L.* Logical Cryptanalysis as a SAT Problem // J. Autom. Reas. 2000. V. 24. No. 1–2. P. 165–203.
14. *Семёнов 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.
15. *Посыпкин М. А., Заикин О. С., Беспалов Д. В., Семёнов А. А.* Решение задач криптоанализа поточных шифров в распределенных вычислительных средах // Труды ИСА РАН. 2009. Т. 46. С. 119–137.
16. *Rueppel R.* Correlation immunity and the summation generator // LNCS. 1986. V. 218. P. 260–272.
17. *Schneier B.* Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C. John Wiley and Sons, 1996. 758 p.
18. *Massacci F. and Marraro L.* Logical Cryptoanalysis as a SAT Problem: the Encoding of the Data Encryption Standard. Preprint. Dipartimento di Imformatica e Sistemistica, Universita di Roma "La Sapienza", 1999.
19. `http://embedded.eecs.berkeley.edu/pubs/downloads/espresso`
20. `http://www.cril.univ-artois.fr/PB10/`
21. *Een N. and Sorensson N.* Translating Pseudo-Boolean Constraints into SAT // J. Satisfiab., Bool. Model. Comp. 2006. V. 2. P. 1–25.
22. `http://miplib.zib.de` MIPLIB — Mixed Integer Problem Library.
23. `http://minisat.se/MiniSat+.html`
24. Системная компьютерная биология / под ред. Н. А. Колчанова, С. С. Гончарова, В. А. Лихошвая, В. А. Иванисенко. Новосибирск: Изд-во СО РАН, 2008. 767 с.
25. *Ganai M. and Gupta A.* SAT-based scalable formal verification solutions. Springer, 2007. 326 p.