Записки научных
семинаров ПОМИ
Том 402, 2012 г.

Д. В. Чистиков

ИСПОЛЬЗОВАНИЕ ЗАПРОСОВ СУЩЕСТВЕННОСТИ ДЛЯ РАСШИФРОВКИ БЕСПОВТОРНЫХ ФУНКЦИЙ

§1. Введение

В работе рассматривается запросная сложность расшифровки (точной идентификации) бесповторных булевых функций в нестандартной модели обучения (learning). Булева функция называется *бесповторной*, если ее можно выразить формулой над базисом $\{\&, \vee, \neg\}$, в которой каждая переменная встречается не более одного раза (бесповторной формулой; иногда используется термин "μ-формула"). Алгоритмы обладают возможностью выполнять стандартные запросы принадлежности (ЗП), выявляющие значение неизвестной функции в точке, и запросы двух специальных типов. Последние позволяют получать ответы на следующие вопросы о подфункциях неизвестной функции f: для некоторой подфункции f_p, порожденной частичной подстановкой констант p:

(ЗС) является ли переменная x_i существенной для f_p?
(СЗС) сколько существенных переменных у f_p?

Здесь используется стандартное определение существенности переменных для булевых функций: переменная x_i называется *существенной*, если найдется входной набор, изменение значения x_i в котором приводит к изменению значения функции. Запросы первого типа (ЗС – *запросы существенности*) имеют два параметра: p и x_i, а запросы второго типа (СЗС – *считающие запросы существенности*) – один параметр p. Нетрудно видеть, что запросы каждого из этих типов моделируются запросами другого типа.

Основные результаты работы состоят в следующем. Разработаны два алгоритма, работающие в течение полиномиального времени и точно идентифицирующие неизвестную бесповторную функцию

Ключевые слова: обучение посредством запросов, расшифровка, точная идентификация, бесповторная булева функция, существенная переменная.

Исследования поддержаны Российским фондом фундаментальных исследований, проект 09-01-00817, и грантом Президента РФ МД–757.2011.9.

n переменных. Первый алгоритм выполняет $O(n^2)$ запросов принадлежности и запросов существенности (ЗП и ЗС), второй алгоритм – $O(n \log^2 n)$ запросов принадлежности и считающих запросов существенности (ЗП и СЗС). Для каждого фиксированного n оба алгоритма можно изобразить детерминированными деревьями решений. Соответствующие мощностные нижние оценки запросной сложности (количества запросов, выполняемых в худшем случае) составляют $\Omega(n \log n)$ и $\Omega(n)$ соответственно. Второй алгоритм, таким образом, выполняет как максимум в $O(\log^2 n)$ раз большее число запросов, чем минимально необходимое. Если учитывать не количество запросов, а число бит, получаемых от оракулов, которые отвечают на эти запросы, то верхняя оценка для второго алгоритма составляет $O(n \log^3 n)$, а нижняя мощностная оценка – $\Omega(n \log n)$.

Существует несколько причин исследовать задачи обучения в данной модели. Прежде всего, запросы существенности представляются естественными для процесса обучения как такового. Представим ученика, изучающего правило, которое сопоставляет одну из двух меток каждой возможной комбинации признаков, характеризующих допустимые ситуации (примеры). В этой интерпретации запросы принадлежности – это запросы правильной классификации некоторого примера (комбинации признаков): "Если эти признаки выражены, а эти – нет, как классифицировать пример?" Запросы существенности – это вопросы следующего вида: "Пусть известна информация о наличии или отсутствии некоторых признаков; может ли значение данного признака повлиять на решение о классификации?" Поскольку правило классификации учителю (обучающему) известно, он может дать правильный ответ на такой вопрос. Заметим, что по сравнению со считающими запросами более естественным представляется вопрос о полном списке существенных признаков (а не только об их количестве).

Понятие существенной переменной играет важную роль в исследованиях, связанных с бесповторными булевыми функциями. Нетрудно заметить, к примеру, что все бесповторные функции f, существенно зависящие от двух и более переменных, обладают следующим свойством: для каждой существенной переменной x_i найдутся булева константа $\sigma \in \{0, 1\}$ и другая переменная x_j, которая является существенной для функции f, но не для подфункции $f_{x_i \leftarrow \sigma}$, получаемой подстановкой σ на место x_i. В 1963 году Б. А. Субботовская доказала

вариант обращения этого свойства: функция f бесповторна тогда и только тогда, когда все ее подфункции f_p (включая саму f) обладают этим свойством [7]. Дальнейшим развитием этого результата стал перечень всех минимальных небесповторных подфункций, полученный В. А. Стеценко [6].

В задачах обучения (связанных также с задачами тестирования) для бесповторных функций понятие существенности также оказывается очень важным. Рассмотрим обобщенные бесповторные формулы, в которых функциональные символы берутся из произвольного множества булевых функций, называемого базисом (стандартные бесповторные функции являются, таким образом, бесповторными в базисе $\{\&, \vee, \bar{}\}$). Оказывается, что для произвольного конечного базиса задача отличения конкретной (обобщенной) бесповторной функции от всех остальных таких функций (в том же базисе) решается проверкой значений на множестве входных наборов полиномиальной мощности в случае, если множество всех существенных переменных известно заранее [3, 11]. Если же такая информация недоступна, то сложность задачи становится экспоненциальной: например, чтобы отличить функцию $f(x_1, \ldots, x_n) \equiv 0$ от всех бесповторных конъюнкций букв (литералов), нужно проверить все 2^n входных наборов.

Для задач точной идентификации (расшифровки) известны сходные результаты [4, 2]. Пусть в дополнение к стандартным запросам принадлежности (запросам значения в произвольной точке, то есть на произвольном входном наборе) алгоритмы могут запрашивать, имеется ли у некоторой подфункции хотя бы одна несущественная переменная. Для стандартного базиса $\{\&, \vee, \bar{}\}$ задача точной идентификации с такими типами запросов разрешима полиномиально в том и только том случае, когда множество существенных переменных известно заранее (иначе для расшифровки требуется экспоненциальное число запросов в худшем случае).

Задачи точной идентификации для бесповторных булевых функций изучались с 1984 года, когда Л. Валиант описал алгоритм, решающий эту задачу полиномиальным числом запросов трех типов [14]. Д. Англуин, Л. Хеллерштайн и М. Карпински [10] разработали алгоритм идентификации, использующий запросы только одного из этих трех типов (а именно запросов существенной возможности – relevant possibility, позволяющих получать ответы на вопросы следующего типа:

"Содержатся ли буквы x_i^σ и x_j^τ в некоторой общей простой импликанте функции f?"), а также алгоритм, использующий запросы принадлежности и запросы эквивалентности (такой запрос фактически позволяет узнать, выразима ли f бесповторной формулой – параметром запроса; более общее исходное определение дается в работе [9]). Оба алгоритма работают полиномиальное время; первый из них выполняет $O(n^2)$ запросов существенной возможности, второй – $O(n^3)$ запросов принадлежности и n запросов эквивалентности (для частного случая монотонных бесповторных функций достаточно $O(n^2)$ запросов принадлежности, а запросы эквивалентности не нужны).

Обратим также внимание на то, что в стандартном базисе $\{\&, \vee, \overline{\ }\}$, который рассматривается в настоящей работе, бесповторная функция $f(x_1, \ldots, x_n)$ обладает несущественной переменной x_i тогда и только тогда, когда число ее *единиц* (наборов $\alpha \in \{0,1\}^n$, на которых она принимает значение 1: $f(\alpha) = 1$) четно (см., например, [4]). Это означает, что вопрос о наличии несущественной переменной эквивалентен запросу арифметической суммы $S(f)$ всех значений $f(\alpha_1, \ldots, \alpha_n)$, взятой по модулю два. Более того, число несущественных переменных f равно наибольшему целому k, для которого 2^k делит $S(f)$.

§2. Основные определения и постановки задач

Дадим вначале определение *бесповторной формулы* над базисом $\{\&, \vee, \overline{\ }\}$. Если x – булева переменная, то x является также бесповторной формулой. Если $\mathcal{F}_1$ и $\mathcal{F}_2$ – бесповторные формулы, то бесповторной формулой является запись $\overline{\mathcal{F}}_1$, а также $(\mathcal{F}_1 \,\&\, \mathcal{F}_2)$ и $(\mathcal{F}_1 \vee \mathcal{F}_2)$, если множества переменных, встречающихся в $\mathcal{F}_1$ и $\mathcal{F}_2$, не пересекаются. Всякая (корректная) формула $\mathcal{F}$ выражает булеву функцию f, определяемую стандартным образом. Функция f называется *бесповторной функцией*, если если она выразима какой-либо бесповторной формулой. По соглашению будем обыкновенно предполагать, что булевы константы 0 и 1 также являются бесповторными функциями, которые выражаются соответствующими формулами без вхождений символов переменных.

Буквы (литералы) переменной x суть $x^1 = x$ и $x^0 = \overline{x}$. Символы $\&$ и $\vee$, а также константы 0 и 1 называются (попарно) взаимно *двойственными*.

Структура бесповторных формул изображается корневыми деревьями. Деревья такого вида фактически являются схемами из функциональных элементов в базисе из отрицания, а также конъюнкций и дизъюнкций с произвольным числом входов; мы будем изображать такие деревья с корнями внизу и листьями вверху. Более формально, рассмотрим корневое дерево T, удовлетворяющее следующим условиям:

1) всякая вершина, помеченная 0 или 1, является единственной вершиной в дереве;
2) листья помечены буквами различных переменных;
3) нелистовые вершины помечены символами & и $\vee$ и имеют произвольную полустепень захода $s \geqslant 2$;
4) смежные вершины помечены различными символами.

Каждая бесповторная функция f представляется некоторым деревом такого вида. Данное представление является однозначным [5, 12], поэтому в дальнейшем под *деревом* обыкновенно понимается единственное такое дерево для (некоторой фиксированной) бесповторной функции.

В любом корневом дереве T *родитель* (непосредственный предок) некорневой вершины v обозначается символом $\mathrm{anc}(v)$. Поддерево T_v с корнем v содержит вершину v и всех ее потомков в T. Запись $w \in T_v$ означает, что w – вершина, содержащаяся в поддереве T_v. Лист u, помеченный буквой переменной x_i, будет в дальнейшем часто отождествляться с самой этой переменной; таким образом, мы зачастую будем пренебрегать разницей между переменной и ее отрицанием. Множества листьев и множества переменных будут соответственно употребляться как синонимы.

Символом $l(v)$ будем обозначать число листьев, содержащихся в T_v, то есть число листьев – потомков v (если вершина v сама является листом, то будем полагать $l(v) = 1$). Нелистовые вершины будем называть также *внутренними*, а непосредственных потомков вершины – ее *сыновьями*. Для внутренней вершины v символом $\lambda(v)$ будем иногда обозначать число листьев, являющихся сыновьями v.

Переменная x_i булевой функции f называется *существенной*, если существуют такие входные наборы α и α', различающиеся только значением x_i (в i-й компоненте), что $f(\alpha) \neq f(\alpha')$. Все остальные переменные называются *несущественными*, или фиктивными. В настоящей работе символом $R(f)$ обозначается множество всех существенных переменных функции f, а символом $r(f)$ – их количество.

Пусть f – булева функция, а p – набор подстановок вида $x_i \leftarrow g^i$, где каждая g^i – булева функция. Символом f_p будем обозначать функцию, получаемую из f подстановкой функций g^i вместо x_i. Во избежание затруднений все переменные x_i для подстановок $x_i \leftarrow g^i$ полагаются при этом различными, причем никакая переменная x_i не может являться существенной ни для одной из функций g^j, $j \neq i$. Если все подстановки в p имеют вид $x_i \leftarrow \sigma_i$, где $\sigma_i \in \{0,1\}$, то p называется *частичной подстановкой констант*, а функция f_p – *подфункцией* функции f, порождаемой подстановкой p. Нижние индексы при символах функций будут чаще всего интерпретироваться как частичные подстановки констант, а верхние индексы – как номера функций в последовательности, например $f^n, \ldots, f^1$.

В постановках задач, рассматриваемых в настоящей работе, целью является расшифровка (точная идентификация) неизвестной бесповторной функции f известного множества переменных $\{x_1, \ldots, x_n\}$. Существенность этих переменных заранее не известна. В обеих постановках алгоритмам расшифровки (обучения, идентификации) разрешено выполнять *запросы принадлежности* – запрашивать значение f на произвольных входных наборах.

Интересующей нас характеристикой алгоритмов является *запросная сложность* – число запросов, выполняемое в худшем случае, – в противоположность временно́й сложности, равной наибольшему времени работы. Нетрудно убедиться, что оба разработанных алгоритма работают полиномиальное время и не содержат элементов непосредственного перебора.

Для установления нижних оценок сложности рассматриваемых задач расшифровки заметим, что логарифм числа бесповторных функций переменных $x_1, \ldots, x_n$ есть $\Theta(n \log n)$ (см., например, [13]). Пусть разрешены только запросы с ответами да/нет (например, запросы принадлежности и запросы существенности), тогда всякий алгоритм, точно идентифицирующий неизвестную бесповторную функцию этих переменных, должен выполнять $\Omega(n \log n)$ запросов в худшем случае: для доказательства достаточно изобразить алгоритм ориентированным от корня деревом. Если же ответы на запросы могут иметь логарифмическую длину (как в случае со считающими запросами существенности), то эта нижняя оценка опускается до $\Omega(n)$ (для числа

бит, полученных в ответах, сохраняется предыдущая оценка). Дополнительно отметим, что известное утверждение о средней глубине листа в дереве (см., например, [1, раздел 8.6]) показывает, что эти оценки верны и для сложности в среднем.

Для удобства обозначений при описании алгоритмов будем полагать, что все n входных переменных являются существенными для неизвестной функции f. Поскольку задача идентификации множества всех существенных переменных решается либо с помощью n запросов существенности (ЗС), либо с помощью $n+1$ считающего запроса существенности (СЗС), то это упрощение не изменяет верхних оценок $O(n^2)$ и $O(n\log^2 n)$ общего числа запросов, выполняемых алгоритмами.

§3. Запросы существенности: квадратичный алгоритм

В настоящем разделе рассматривается постановка задачи с доступными *запросами существенности*. Каждый такой запрос имеет два параметра: частичную подстановку констант p и переменную x_i. Запрос возвращает 1, если $x_i \in R(f_p)$, то есть если переменная x_i существенна для f_p, и 0 иначе.

Напомним, что нашей целью является точная идентификация неизвестной бесповторной функции f. Как указано выше, можно без ограничения общности предполагать, что все входные переменные являются существенными переменными f. Мы опишем алгоритм, основанный на следующем факте (см., например, [7]).

Утверждение 1. *Для любой бесповторной функции f, $r(f) \geqslant 2$, и любой ее существенной переменной x существует единственное значение $\sigma \in \{0,1\}$ такое, что $R(f_{x\leftarrow\sigma}) = R(f) \setminus \{x\}$.*

Утверждение 1 задает общую схему будущего алгоритма, состоящую в следующем. Построим последовательность подфункций f^n, $f^{n-1},\ldots,f^1$, в которой $f^n = f$, каждая f^k есть подфункция функции f^{k+1} и имеет в точности k существенных переменных (здесь $k = n-1,\ldots,1$). Иными словами, будет построена такая последовательность переменных $x_n,\ldots,x_1$, что $R(f^k) = \{x_1,\ldots,x_k\}$ и каждая функция f^{k-1} получается из f^k подстановкой некоторой булевой константы α_k на место переменной x_k. Эта конструкция описывается следующей диаграммой:

$$f^n \xrightarrow{x_n\leftarrow\alpha_n} f^{n-1} \xrightarrow{x_{n-1}\leftarrow\alpha_{n-1}} f^{n-2} \longrightarrow \cdots \longrightarrow f^2 \xrightarrow{x_2\leftarrow\alpha_2} f^1.$$

Подфункция f^1, зависящая от x_1, равна либо x_1, либо $\overline{x}_1$ и потому идентифицируется с помощью одного запроса принадлежности (значения функции в точке). Основная трудность состоит в обратных переходах, то есть в восстановлении функций f^k по их подфункциям f^{k-1}. Наш план заключается в выборе переменной x_k таким образом, что это восстановление может быть произведено без дополнительных усилий, то есть без выполнения запросов вообще.

Введем следующие обозначения. Для функции g и ее существенных переменных x и y будем записывать $x \trianglerighteq y$, если существует такая константа $\tau \in \{0,1\}$, что переменная y не является существенной для подфункции $g_{x\leftarrow\tau}$. Заметим, что $x \trianglerighteq x$ для всех (существенных) переменных x, и положим $x \triangleright y$, если и только если $x \trianglerighteq y$ и $x \neq y$. Обратим внимание на транзитивность исходного (рефлексивного) отношения $\trianglerighteq$ [7].

Утверждение 2. *Для любой булевой функции f и любых ее существенных переменных x, y, z из отношений $x \trianglerighteq y$, $y \trianglerighteq z$ следует $x \trianglerighteq z$.*

Для бесповторных функций с двумя и более существенными переменными из данного утверждения вытекает существование цикла длины два в графе отношения $\trianglerighteq$ [7].

Утверждение 3. *Для любой бесповторной функции f, $r(f) \geqslant 2$, найдутся две переменные x, y, существенные для f и такие, что $x \triangleright y$ и $y \triangleright x$.*

Вернемся теперь к подфункции f^k неизвестной функции f. Заметим, что как только найдена такая пара переменных x, y, что $x \triangleright y$ и $y \triangleright x$, наш план фактически выполнен. В самом деле, пусть σ и τ выбраны так, что y не существенна для $f^k_{x\leftarrow\sigma}$ и x не существенна для $f^k_{y\leftarrow\tau}$. Обозначим символом f^{k-1} подфункцию $f^k_{x\leftarrow\overline{\sigma}}$ и заметим, что из утверждения 1 следует равенство $R(f^{k-1}) = R(f^k) \setminus \{x\}$. В соответствии с определением существенной переменной заключаем, что справедливы тождества $f_{x\leftarrow\sigma,y\leftarrow\tau} = f_{x\leftarrow\overline{\sigma},y\leftarrow\tau} = f_{x\leftarrow\sigma,y\leftarrow\overline{\tau}}$. Поскольку $f^{k-1} = f^k_{x\leftarrow\overline{\sigma}}$, то отсюда следует, что $f^k = f^{k-1}_{y\leftarrow(x^\sigma \vee y^\tau)^\tau}$.

Таким образом, остался лишь один нерешенный вопрос: как найти подходящую пару x, y для каждой (зависящей от k переменных) функции f^k? Непосредственный перебор требует $\Theta(k^2)$ запросов в худшем случае, однако наш алгоритм будет использовать результат утверждения 2 и выполнять только $O(k)$ запросов на соответствующем шаге. Уточнение деталей проводится ниже.

Алгоритм $\mathcal{A}_1$

1. Определить все n существенных переменных.
   Положить $f^n = f$.
2. Для $k = n, \ldots, 2$:
   (a) Взять произвольную переменную $x \in R(f^k)$.
   (b) Для каждой переменной $y \in R(f^k) \setminus \{x\}$:
      (i) Для f^k, если $x \rhd y$ и $y \rhd x$, то положить $x_k = x$, $y_k = y$, выбрать такие σ_k и τ_k, что $y_k \notin R(f^k_{x_k \leftarrow \sigma_k})$ и $x_k \notin R(f^k_{y_k \leftarrow \tau_k})$, и перейти к шагу 2c.
      (ii) Для f^k, если $x \rhd y$ и $y \not\rhd x$, то положить $x = y$.
   (c) Получить $f^{k-1} = f^k_{x_k \leftarrow \overline{\sigma}_k}$.
3. Определить подфункцию f^1, зависящую от оставшейся переменной.
4. Для $k = 2, \ldots, n$: восстановить $f^k = f^{k-1}_{y_k \leftarrow (x_k^{\sigma_k} \vee y_k^{\tau_k})^{\tau_k}}$.
5. Завершить работу с ответом $f = f^n$.

Рис. 1. Алгоритм расшифровки запросами принадлежности и существенности.

Изложенные выше идеи позволяют описать алгоритм $\mathcal{A}_1$ на Рис. 1. В описании шагов 2(b)i и 2(b)ii равенства $x_k = x$, $y_k = y$ и $x = y$ задают не присваивания, а обозначения: $x_k = x$, к примеру, означает, что *имя* x_k будет в дальнейшем обозначать переменную, которая ранее называлась x. То же относится и к равенствам $f^n = f$ и $f = f^n$ на шагах 1 и 5. Равенство шага 2c фиксирует значение переменной x_k для дальнейших запросов на последующих итерациях шага 2 и на шаге 3. Равенство же в описании шага 4 указывает, как получить бесповторную формулу для f^k из известной бесповторной формулы для f^{k-1}. Отметим также, что на каждой итерации шага 2 множество $R(f^k) \setminus \{x\}$ возможных значений имени y вычисляется только один раз в ходе шага 2b, вне зависимости от возможных изменений значения имени x.

Теорема 1. *Алгоритм $\mathcal{A}_1$ правильно идентифицирует любую бесповторную функцию f и выполняет один запрос принадлежности и $O(n^2)$ запросов существенности, где n – число входных переменных f.*

Доказательство. Вначале оценим число выполняемых запросов. Заметим, что двух запросов существенности достаточно для проверки,

выполнено ли отношение $x \rhd y$ для некоторой фиксированной подфункции f^k и заданной пары существенных переменных x, y. Следовательно, всего алгоритм выполняет $O(n^2)$ запросов существенности, а запрос принадлежности нужен лишь для шага 3.

Докажем теперь корректность описанной процедуры: она, очевидно, требует дополнительного рассмотрения. Мы уже знаем, что выбор x_k, y_k на шаге 2(b)i гарантирует правильное восстановление на шаге 4. Доказательства требует тот факт, что цикл шага 2b всегда обнаруживает подходящую пару x_k, y_k и завершается на шаге 2(b)i.

Чтобы доказать это, предположим противное. Пусть $y_1, \ldots, y_{k-1}$ – переменные, проверяемые на шаге 2b. Обозначим символом y_0 исходную переменную x, а символами $y_{i_0}, y_{i_1}, \ldots, y_{i_s}$ – все переменные, обозначаемые именем x во время различных итераций цикла (здесь $i_0 = 0$). Заметим, что $y_{i_0} \rhd y_{i_1} \rhd \ldots \rhd y_{i_s}$ и, кроме того, $y_{i_s} \not\rhd y_j$ при $j < i_s$ в силу утверждения 2. Следовательно, согласно утверждению 1, найдется такой номер $j > i_s$, что $y_{i_s} \rhd y_j$, но это противоречит сделанным предположениям. Теорема доказана. □

Замечание. Полезно заметить, что единственный запрос принадлежности используется алгоритмом для того, чтобы выбрать верное из соотношений $f^1 = x_1$ и $f^1 = \overline{x}_1$ (в предположении $R(f) = \{x_1, \ldots, x_n\}$). Поскольку после этого формула для f получается из формулы для f^1 подстановками вида $y_k \leftarrow (x_k^{\sigma_k} \vee y_k^{\tau_k})^{\tau_k}$, то различные ответы на этот запрос принадлежности ведут к паре бесповторных функций, которые являются отрицаниями друг друга. Это означает, что описанному алгоритму запрос принадлежности как таковой не требуется. Вместо этого запроса алгоритму можно подать на вход произвольную корректную пару вида $\langle \alpha, f(\alpha) \rangle$, где α – любой входной набор из $\{0,1\}^n$. Этап восстановления нового алгоритма будет давать две функции – догадки об f, одна из которых несовместна с данным примером. Другая догадка совпадает с f и является ответом алгоритма.

§4. Считающие запросы: обзор алгоритма

В настоящем разделе рассматривается постановка задачи, в которой доступны *считающие запросы существенности*. Единственный параметр такого запроса – частичная подстановка констант p. Ответом является значение $r(f_p)$, то есть количество переменных, существенных для подфункции f_p.

Высокоуровневая схема алгоритма идентификации будет такой же, как в предыдущем разделе. Мы по-прежнему будем строить такую последовательность подфункций f^k, $k = n, \ldots, 1$, неизвестной функции f, что каждая подфункция f^k имеет ровно k существенных переменных и получается из подфункции f^{k+1} подстановкой константы на место некоторой переменной x_{k+1}.

Тем не менее, наша стратегия будет отличаться от примененной в разделе 3. Напомним, что для подфункции f^k мы использовали $O(k)$ запросов при нахождении подходящей переменной x_k с соответствующей булевой константой. Это обнуляло стоимость шага восстановления: подфункция f^k получалась из f^{k-1} без выполнения дополнительных запросов. К сожалению, это давало нам оценку в $\Theta(n^2)$ запросов для функции n переменных в худшем случае. В настоящем разделе мы будем довольствоваться любой существенной переменной в качестве x_k и выполнять $O(1)$ запросов для нахождения соответствующей константы. Таким образом, общая сложность алгоритма будет определяться количеством запросов, требуемых для восстановления всех функций f^k по их предшественникам f^{k-1}. Весь процесс будет основан на следующей лемме, которая неявно используется в работе [4]:

Лемма 1. *Пусть g – бесповторная функция, имеющая $k \geqslant 3$ существенных переменных. Пусть $x \in R(g)$ и $\sigma \in \{0,1\}$ выбраны таким образом, что $0 < r(g_{x \leftarrow \overline{\sigma}}) < r(g_{x \leftarrow \sigma})$. Тогда дерево T функции g можно получить из дерева T' функции $g_{x \leftarrow \sigma}$ с помощью одной из следующих операций (см. Рис. 2):*

1) *присоединение нового листа x к некоторой внутренней вершине v;*
2) *замена некоторого листа y на новую внутреннюю вершину u, помеченную символом, двойственным к пометке родителя v листа y, и присоединение листьев с буквами переменных x и y в качестве сыновей вершины u;*
3) *разрезание внутренней вершины v на две вершины v_0 и v_1 и вставка новой вершины u между ними: если вершина v помечена символом $\vee$, а в ее сыновьях реализуются функции $g_1, \ldots, g_s$, то T получается заменой T'_v на поддерево, реализующее функцию*

$$((g_{i_1} \vee \ldots \vee g_{i_t}) \,\&\, x^\sigma) \vee g_{i_{t+1}} \vee \ldots \vee g_{i_s},$$

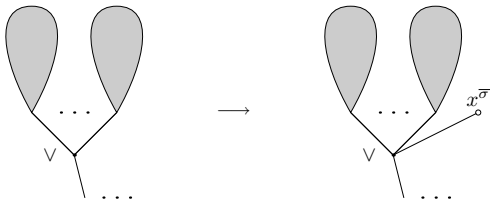
где $\{i_1, \ldots, i_s\} = \{1, \ldots, s\}$, $s \geqslant 3$ *и* $t \geqslant 2$ (*если* v *помечена символом* &, *то все символы в формуле заменяются двойственными*).

Доказательство. Согласно утверждению 1, имеем $R(g) \setminus R(g_{x \leftarrow \sigma}) = \{x\}$. Пусть дерево T функции g известно, тогда рассмотрим родителя w вершины x в T. Пусть d – число сыновей (непосредственных потомков) вершины w. Если $d \geqslant 3$, то подстановка σ на место x приводит к удалению одного из этих сыновей, остальная же часть дерева при этом не меняется. Это соответствует пункту 1 в приведенном списке. Если $d = 2$, то у вершины w есть, кроме x, ровно один сын. Если этот сын – тоже лист, то ситуация описана в пункте 2. В противном случае этот сын помечен тем же символом, что и родитель вершины w, и эти вершины склеены в одну вершину в T'. Это соответствует пункту 3. Отметим, что вершина w не может быть корнем T, так как $r(g_{x \leftarrow \overline{\sigma}}) > 0$. Лемма доказана. □

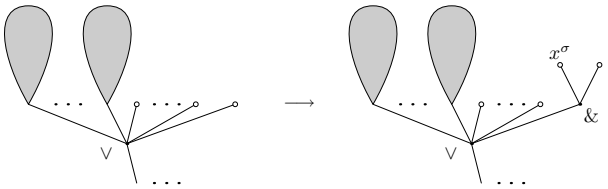
Отметим, что случай $r(g_{x \leftarrow \overline{\sigma}}) = 0$ является простым и разбирается отдельно.

Пусть $f^{k-1} = f^k_{x \leftarrow \sigma}$, где f^k, x и σ удовлетворяют условиям леммы 1. Пусть T' – известное дерево функции f^{k-1}, а T – неизвестное дерево f^k. Разобьем задачу восстановления T по T' на две части. Вначале найдем такую вершину $a \in T'$, что $a = \mathrm{anc}(\mathrm{anc}(x))$ в T. Затем идентифицируем множество переменных, существенных для $f^k_{x \leftarrow \sigma}$, но не для $f^k_{x \leftarrow \overline{\sigma}}$. После этого в дерево T' легко вставляется лист, помеченный подходящей буквой переменной x. Полученное таким образом дерево будет являться деревом функции f^k, что и требуется.

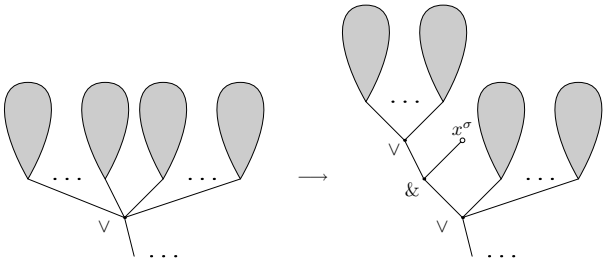
Представим теперь набросок нашего алгоритма $\mathcal{A}_2$ точной идентификации с помощью запросов принадлежности и считающих запросов существенности. Основные шаги алгоритма перечислены на Рис. 3. Не будем пока останавливаться на деталях реализации, сколь бы важны они ни были, и опустим строгое описание всех действий. Необходимые пояснения для шагов 2, 3, 4a и 4d будут даны в доказательстве теоремы 2 в настоящем разделе; реализация же шагов 4b и 4c откладывается до разделов 5 и 6 соответственно. Наша конечная цель – доказать, что все шаги можно реализовать таким образом, что общая запросная сложность будет составлять $O(n \log^2 n)$.



a) Присоединение нового листа к существующей вершине.



b) Добавление новой внутренней вершины с двумя потомками.



c) Разрезание существующей вершины.

Рис. 2. Преобразования деревьев (в двойственном случае символы $\vee$ и $\&$ меняются местами, а константа σ инвертируется).

Нетрудно заметить, что основная сложность зависит в первую очередь от реализации шагов 4b и 4c, поскольку все остальные вычисления, очевидно, требуют значительно меньшего числа запросов. Докажем, что эти (основные) шаги допускают реализацию вспомогательными процедурами с подходящей запросной сложностью. Разобьем данное утверждение на два.

Алгоритм $\mathcal{A}_2$

1. Определить все существенные переменные.
   Обозначить их $x_1, \ldots, x_n$.
2. Построить такую последовательность констант $\sigma_n, \ldots, \sigma_2$ из $\{0,1\}$, что для функций $f^n, \ldots, f^1$, определяемых равенствами $f^n = f$ и $f^{k-1} = f^k_{x_k \leftarrow \sigma_k}$, были выполнены соотношения $R(f^k) = \{x_1, \ldots, x_k\}$.
3. Выбрать верное из равенств $f^1 = x_1$ и $f^1 = \overline{x}_1$.
4. Для $k = 2, \ldots, n$ восстановить f^k из ее подфункции f^{k-1}:
   (a) Если $r(f^k_{x_k \leftarrow \overline{\sigma}_k}) = 0$, то выбрать $\circ \in \{\&, \vee\}$ и $\tau \in \{0,1\}$ так, что $f^k = f^{k-1} \circ x_k^\tau$, восстановить дерево функции f^k и перейти к следующему k.
   (b) Найти такую вершину a в известном дереве функции f^{k-1}, что $a = \mathrm{anc}(\mathrm{anc}(x_k))$ в неизвестном дереве функции f^k.
   (c) Определить множество переменных $R(f^k_{x_k \leftarrow \sigma_k}) \backslash R(f^k_{x_k \leftarrow \overline{\sigma}_k})$.
   (d) Преобразовать дерево функции f^{k-1} в дерево функции f^k и перейти, если возможно, к следующему k.
5. Завершить работу с $f = f^n$.

Рис. 3. Алгоритм расшифровки запросами принадлежности и считающими запросами существенности.

Лемма 2. *Вычисления на шаге 4b алгоритма $\mathcal{A}_2$ могут быть произведены вспомогательной процедурой, выполняющей самое большее $\log_2 k + O(1)$ считающих запросов существенности.*

Лемма 3. *Вычисления на шаге 4c алгоритма $\mathcal{A}_2$ могут быть произведены вспомогательной процедурой, выполняющей считающие запросы существенности и обладающей следующим свойством: если алгоритм $\mathcal{A}_2$ применен к бесповторной функции f с n существенными переменными, то общее число запросов, выполняемых этой процедурой, есть $O(n \log^2 n)$.*

Доказательства этих двух лемм приводятся ниже. Раздел 5 посвящен доказательству леммы 2, а раздел 6 – доказательству леммы 3.

Теперь доказательство анонсированной оценки сложности фактически сведено к доказательству лемм 2 и 3. Приводимое ниже доказательство теоремы 2 восполняет недостающие детали.

Теорема 2. *Алгоритм $\mathcal{A}_2$ правильно идентифицирует любую бесповторную функцию f и выполняет один запрос принадлежности и $O(n\log^2 n)$ считающих запросов существенности, где n – число входных переменных f.*

Доказательство. Заметим вначале, что $y \in R(f)$ тогда и только тогда, когда $r(f) \neq r(f_{y\leftarrow 0})$, поэтому шаг 1 реализуется с помощью $n+1$ запроса. Шаг 2 выполняется в соответствии с утверждением 1, требует самое большее $2(n-1)$ запросов и выявляет значения $r(f^k_{x_k\leftarrow 0})$, $r(f^k_{x_k\leftarrow 1})$ для $k = n, \ldots, 2$. На шаге 3 выполняется один запрос принадлежности.

Рассмотрим итерации шага 4. Предположим вначале, что значение $r(f^k_{x_k\leftarrow\overline{\sigma}_k}) = 0$. Докажем, что одного считающего запроса существенности достаточно для восстановления f^k. В самом деле, заметим, что подфункция $f^k_{x_k\leftarrow\overline{\sigma}_k}$ равна некоторой булевой константе β. Возьмем произвольные значения $\alpha_1, \ldots, \alpha_{k-1} \in \{0,1\}$ и построим частичную подстановку констант $p' = \{x_1 \leftarrow \alpha_1,\ x_2 \leftarrow \alpha_2,\ \ldots\ ,\ x_{k-1} \leftarrow \alpha_{k-1}\}$. Значение $\beta' = f^{k-1}_{p'}$ уже известно, причем $\beta = \beta'$ тогда и только тогда, когда $r(f^k_{p'}) = 0$. (Заметим, что мы могли бы использовать и запрос принадлежности для выявления значения β.) Если $\beta = 0$, то $f^k = f^{k-1}\ \&\ x_k^{\sigma_k}$, в противном случае $f^k = f^{k-1} \vee x_k^{\overline{\sigma}_k}$.

Теперь допустим, что $r(f^k_{x_k\leftarrow\overline{\sigma}_k}) > 0$. Согласно леммам 2 и 3, шаги 4b и 4c обнаруживают вершину $a = \mathrm{anc}(\mathrm{anc}(x_k))$ и множество переменных, существенных для $f^k_{x_k\leftarrow\sigma_k}$, но не для $f^k_{x_k\leftarrow\overline{\sigma}_k}$. После этого на шаге 4d дерево функции f^{k-1} преобразуется в дерево функции f^k в соответствии с леммой 1. Буква переменной x_k выбирается при этом так, чтобы все переменные функции f^k, кроме x_k, были существенными для подфункции $f^k_{x_k\leftarrow\sigma_k}$ (аналогично случаю $r(f^k_{x_k\leftarrow\overline{\sigma}_k}) = 0$, хотя в данном случае дополнительных запросов уже не требуется).

Остается определить общее число выполняемых алгоритмом запросов. Имеем:

$$O(n) + \sum_{k=2}^{n}(\log_2 k + O(1)) + O(n\log^2 n) = O(n\log^2 n),$$

что и завершает доказательство. □

Замечание. Алгоритм $\mathcal{A}_2$, подобно алгоритму $\mathcal{A}_1$, не использует возможности запроса принадлежности в полной мере в том смысле, что

этот запрос можно заменить любой корректной парой $\langle \alpha, f(\alpha) \rangle$, подаваемой на вход. Лемма 1 свидетельствует о том, что структуру дерева f можно восстановить, используя лишь последовательные разности $R(f^k_{x_k \leftarrow \sigma_k}) \setminus R(f^k_{x_k \leftarrow \overline{\sigma}_k})$, с единственным исключением для случая $r(f^k_{x_k \leftarrow \overline{\sigma}_k}) = 0$, где один считающий запрос существенности попрежнему предоставляет всю нужную информацию. Можно доказать по индукции, что без запроса принадлежности алгоритм может вычислять две функции – догадки об f, являющиеся отрицаниями друг друга. Этот эффект связан со следующим наблюдением: любые типы запросов, зависящие только от информации о существенности входных переменных, могут различать булевы значения, но не определять, какое из них является нулем, а какое – единицей.

§5. Доказательство леммы 2

Настоящий раздел посвящен доказательству леммы 2. Вначале вводятся нужные обозначения, формулируется утверждение леммы и описываются основные идеи доказательства. После этого доказательство приводится полностью.

Итак, пусть f^{k-1} – известная бесповторная функция с $R(f^{k-1}) = \{x_1, x_2, \ldots, x_{k-1}\}$. Пусть f^k – неизвестная бесповторная функция, удовлетворяющая соотношениям $R(f^k) = R(f^{k-1}) \cup \{x_k\}$ и $f^k_{x_k \leftarrow \sigma_k} = f^{k-1}$, где σ_k – константа из $\{0, 1\}$. Обозначим символом T' известное дерево f^{k-1}, а символом T – неизвестное дерево f^k.

Согласно описанию алгоритма $\mathcal{A}_2$, будем предполагать, что выполняется неравенство $r(f^k_{x_k \leftarrow \overline{\sigma}_k}) > 0$. Из него, в частности, следует, что лист дерева T, помеченный буквой переменной x_k, не смежен с корнем T. Другими словами, в дереве T существует такая вершина a, что $a = \mathrm{anc}(\mathrm{anc}(x_k))$. Напомним, что T получается из T' посредством одной из операций, описанных в лемме 1, так что a является также вершиной дерева T'. Цель настоящего раздела – доказать, что эта вершина $a \in T'$ допускает идентификацию с помощью $\log_2 k + O(1)$ считающих запросов существенности.

Поскольку на протяжении всего доказательства значение k остается неизменным, мы будем использовать вспомогательные обозначения, чтобы избежать ненужных индексов. Положим по определению $x = x_k$ и $\sigma = \sigma_k$. Обозначим также $g = f^k$ и $g' = f^k_{x \leftarrow \sigma} = f^{k-1}$, тогда T – дерево g, а T' – дерево g'. Кроме того, положим $g'' = f^k_{x \leftarrow \overline{\sigma}}$ и определим

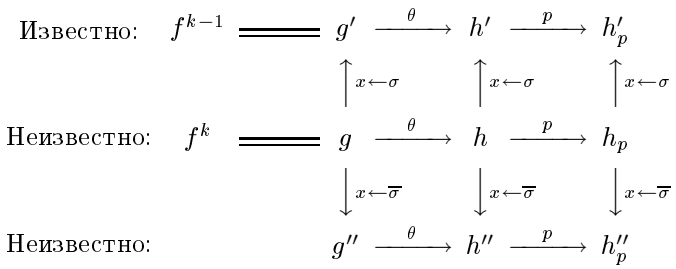


Рис. 4. Подфункции и подстановки в доказательстве леммы 2.

$m = r(g') - r(g'')$. Отметим, что связи между данными подфункциями функции f отражены в двух левых столбцах диаграммы на Рис. 4.

Основная идея доказательства состоит в нахождении какой-либо переменной y, принадлежащей разности множеств $R(g') \setminus R(g'')$. На этом этапе точная идентификация всего множества таких переменных не требуется, поскольку главная цель данного шага – определение вершины a. Тем не менее, переменные, обладающие описанным свойством, являются ключом к нахождению a.

Поиск переменной $y \in R(g') \setminus R(g'')$ непосредственным перебором связан с подстановкой констант на места отдельных переменных g'' и анализом числа переменных, существенных для получаемых таким образом подфункций. К сожалению, такой подход использует $\Theta(k)$ запросов в худшем случае, что дает только $O(n^2)$ как верхнюю оценку общей сложности алгоритма. Применяемая в настоящем разделе идея заключается в подстановке констант на места нескольких переменных сразу. Пусть p – частичная подстановка констант на места некоторых s переменных. Заметим, что если все эти переменные существенны для функции g'', то значение $r(g''_p)$ не превосходит $r(g'') - s$. Допустим теперь, что по меньшей мере одна из этих переменных не является существенной для g''. Тогда этот факт заведомо выявляется, если $r(g''_p) > r(g'') - s$. При аккуратном построении подстановки p последнее неравенство будет выполнено всегда, когда p подставляет константу на место некоторой переменной $y \in R(g') \setminus R(g'')$. После этого естественно использовать двоичный поиск для окончательной идентификации переменной y и определения вершины a.

Начнем выполнять описанный план. Нам понадобится вспомогательная конструкция, которая окажется полезной для дальнейших рассуждений. Используем известное значение разности $m = r(g') - r(g'')$, чтобы свести преобразования деревьев из леммы 1 к манипуляциям с некоторым заранее очерченным множеством поддеревьев. Напомним, что число листьев дерева, являющихся потомками вершины v, обозначается символом $l(v)$. Рассмотрим множество

$$N = \{\, u \in T' \mid u \neq v_0,\ l(u) \leqslant m,\ l(\mathrm{anc}(u)) > m \,\},$$

где v_0 – корень T'. Следующее утверждение ограничивает возможные преобразования деревьев, переводящие T' в T, условием сохранения всех поддеревьев T'_u, где $u \in N$.

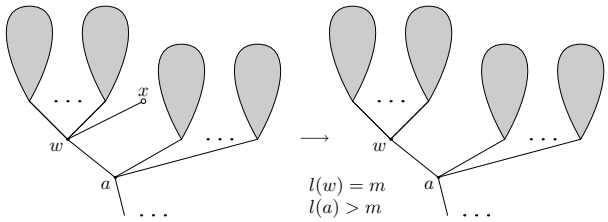
Утверждение 4. *Все переменные множества $R(g') \setminus R(g'')$ содержатся в поддеревьях T'_u, где $u \in N$. Кроме того, все поддеревья T'_u, $u \in N$, содержащие переменные из $R(g') \setminus R(g'')$, не содержат переменных из $R(g'')$ и имеют такие корни u, что $a = \mathrm{anc}(u)$.*

Доказательство. Пусть w – внутренняя вершина дерева T, смежная с x. Обозначим символом l_w число листьев в поддереве T_w, тогда справедливо равенство $m = l_w - 1$. Заметим, что вершина $a = \mathrm{anc}(w) \in T$ обладает хотя бы одним сыном помимо w, поэтому для дерева T' выполняется неравенство $l(a) > l_w - 1 = m$. Более того, в дереве T' все переменные множества $R(g') \setminus R(g'')$ лежат в поддеревьях T'_u, где вершины u – сыновья a (два из возможных случаев изображены на Рис. 5). Все эти поддеревья T_u не изменены в T по сравнению с T' и не содержат переменных из $R(g'')$. Общее число листьев в этих поддеревьях равно m, поэтому $l(u) \leqslant m$. Утверждение 4 доказано. □

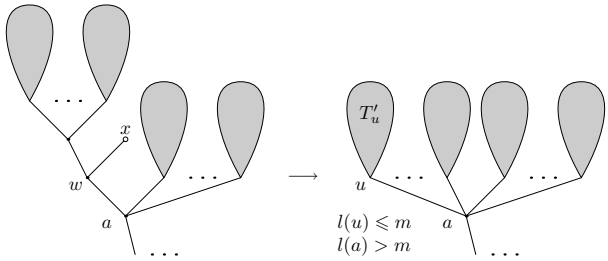
Замечание. Возможная интерпретация утверждения 4 состоит в следующем: как только обнаружена переменная $y \in R(g') \setminus R(g'')$, задача определения вершины a решается указанием на родителя единственной вершины $u \in N$, для которой $y \in T'_u$. Причина этого заключается в том, что для любой переменной $x_i \in R(g')$ существует только одна вершина u на пути из x_i в v_0, для которой $l(u) \leqslant m$, но $l(\mathrm{anc}(u)) > m$.

Согласно лемме 1, дерево T получается из T' либо присоединением нового листа к такой вершине $u \in N$, что $l(u) = m$ (если $m = 1$, это означает введение новой внутренней вершины), либо разрезанием одной из вершин – родителей элементов N, то есть вершин множества

$$A = \{\, \mathrm{anc}(u) \in T' \mid u \in N \,\},$$



a) Вершина w в T имеет полустепень захода 3 или более.



b) Вершина w в T имеет полустепень захода 2.

Рис. 5. Нахождение неизменяемых поддеревьев.

и выделением некоторого подмножества множества N. Пусть теперь u – некоторая нелистовая вершина, содержащаяся в N. Соответствующее поддерево T'_u представляет бесповторную функцию g'_u, имеющую как минимум две существенные переменные. Рассмотрим произвольный лист в T'_u. Пусть этот лист помечен буквой x_i^τ, где $\tau \in \{0, 1\}$. Поскольку переменная x_i существенна для g'_u, найдется такая частичная подстановка θ_u констант на места остальных переменных g'_u, что $(g'_u)_{\theta_u} = x_i^\tau$. Пусть теперь частичная подстановка констант θ есть композиция таких частичных подстановок θ_u для всех нелистовых вершин u из N (см. Рис. 6; если все элементы N – листья, то θ – тождественная подстановка).

Данная подстановка θ переводит функции g, g' и g'' в h, h' и h'' соответственно (см. левые горизонтальные стрелки на Рис. 4). Функции h и h'' по-прежнему неизвестны. Обозначим символом T'_1 дерево функции h', получаемое из T' заменой поддеревьев с корнями из N на отдельные листья. Множество N переводится подстановкой θ в множество N_θ, состоящее из (некоторых) листьев дерева T'_1.

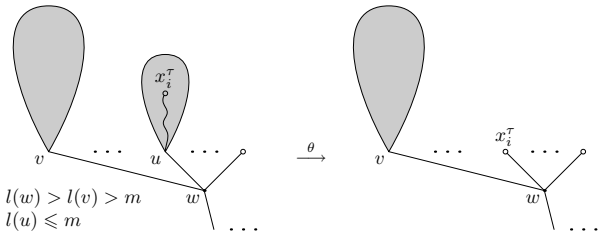


Рис. 6. Частичная подстановка констант θ.

Положим $m_1 = r(h') - r(h'') > 0$. (Здесь для выявления значения $r(h'')$ требуется один считающий запрос существенности.) Для каждой вершины v дерева T_1' обозначим символом $\lambda(v)$ число листьев T_1', которые являются сыновьями вершины v. Пусть

$$A_1 = \{\, v \in A \mid \lambda(v) \geqslant m_1 + \delta(v) \,\},$$

где $\delta(v) = 0$, если вершина v дерева T_1' обладает хотя бы одним сыном – не листом, и $\delta(v) = 1$, если все сыновья v – листья. Рассмотрим множество

$$N_1 = \{\, u \in N_\theta \mid \mathrm{anc}(u) \in A_1 \,\}$$

и докажем, что все существенные переменные h', не существенные для h'', содержатся в N_1.

Утверждение 5. *Для любой переменной $y \in R(h') \setminus R(h'')$ справедливо включение $y \in N_1$ и соотношение $\mathrm{anc}(y) = a \in A_1$.*

Доказательство. Вначале заметим, что $R(h')\setminus R(h'') \subseteq R(g')\setminus R(g'')$. Согласно утверждению 4, все существенные переменные функции g', не существенные для g'', содержатся в поддеревьях T_u', где $u \in N$. Поскольку T_1' получается из T' заменой этих поддеревьев отдельными листьями, то все переменные множества $R(h') \setminus R(h'')$ принадлежат множеству N_θ. Кроме того, из утверждения 4 следует также, что $\mathrm{anc}(u)$ – одна и та же вершина $a \in A$ для всех таких поддеревьев T_u'. Эта вершина a в T_1' должна иметь по меньшей мере m_1 сыновей, являющихся листьями, поэтому остается проверить справедливость неравенства $\lambda(a) \geqslant m_1 + \delta(a)$. В самом деле, если все сыновья a – листья, то их число должно быть больше либо равно $m_1 + 1$, поскольку в противном случае дерево функции h могло бы быть получено из T_1' присоединением нового листа к a, причем то же было бы справедливо

для T и T' (что невозможно, так как некоторые сыновья вершины a в T' принадлежат N и, следовательно, $l(a) > m$ в T', откуда сразу $a \notin N$). Утверждение 5 доказано. □

Утверждение 5 показывает, что дерево функции h'' получается из дерева T_1' удалением некоторых листьев, принадлежащих N_1, и, возможно, "подавлением" получающейся вершины с полустепенью захода 1 и последующей склейкой двух смежных вершин с одинаковыми пометками.

Замечание. Общий подход к поиску вершины a можно реализовать и без явного построения подстановки θ. Представляется, однако, что описанное преобразование делает рассуждения более прозрачными, поскольку основная задача сводится таким образом к своему частному случаю.

Итак, вспомогательные построения завершены; перейдем теперь к реализации описанного выше подхода к задаче идентификации вершины a. Введем следующее вспомогательное определение. Назовем частичную подстановку констант $p = \{x_{i_1} \leftarrow \alpha_1, \ \dots \ , \ x_{i_s} \leftarrow \alpha_s\}$ *слабой*, если выполнены следующие условия:

1) все листья, помеченные буквами переменных $x_{i_1}, \dots, x_{i_s}$, смежны в дереве функции h' с внутренними вершинами из множества A_1;
2) для каждого j все существенные переменные функции h', кроме x_{i_j}, являются существенными и для подфункции $h'_{x_{i_j} \leftarrow \alpha_j}$;
3) какова бы ни была внутренняя вершина v дерева T_1', если все ее сыновья – листья, то самое большее $(\lambda(v) - m_1)$ из них помечены буквами переменных из множества $\{x_{i_1}, \dots, x_{i_s}\}$.

Понятие слабой частичной подстановки выражает нашу идею отделения случая, когда константы подставляются на места лишь существенных переменных, от случая, когда одной из несущественных переменных присваивается некоторое фиксированное значение. Формализация дается следующим утверждением:

Утверждение 6. *Если слабая частичная подстановка констант p задает значение некоторой переменной y, не являющейся существенной для функции h'', то все существенные переменные h'', за исключением тех, значения которых задает p, являются существенными и для подфункции h''_p.*

Доказательство. Заметим вначале, что основные (хотя и не все) подфункции f, используемые в настоящем доказательстве, можно найти на Рис. 4. Если $y \in R(h') \setminus R(h'')$, то в дереве T_1' функции h' справедливо соотношение $\mathrm{anc}(y) = a$. Для произвольной вершины v дерева T_1' обозначим символом h_v' функцию, реализуемую поддеревом $(T_1')_v$, а символом h_v'' – функцию, реализуемую соответствующим поддеревом дерева h''. (Отметим, что хотя такие нижние индексы можно по-прежнему интерпретировать как частичные подстановки, это значение оказывается здесь второстепенным.)

В силу условия 2 в определении слабой частичной подстановки констант ни одна из подстановок вида $x_{i_j} \leftarrow \alpha_j$ в p не может сделать другие переменные h' несущественными. Вспомним, что дерево h'' получается из T_1' удалением некоторых листьев. Нетрудно убедиться, что это удаление также можно представить композицией ξ некоторых m_1 подстановок вида $x_i \leftarrow \tau_i$, обладающих тем же свойством (для этой частичной подстановки констант ξ справедливо соотношение $(h')_\xi = h''$). Следовательно, достаточно доказать, что в условиях утверждения для каждой внутренней (нелистовой) вершины $v \in T_1'$ соответствующая функция h_v' переводится композицией ξp в неконстантную подфункцию $(h_v')_{\xi p} = (h_v'')_p$. Докажем это, опираясь на тот факт, что функция является неконстантной тогда и только тогда, когда у нее есть хотя бы одна существенная переменная.

Рассмотрим вначале такие вершины $v \in T_1'$, что вершина a не является ни вершиной v, ни ее потомком. Воспользуемся индукцией по глубине $(T_1')_v$ (наибольшему числу ребер на кратчайших путях из вершины v в ее потомки). Если все сыновья вершины v – листья, то требуемое непосредственно вытекает из условия 3 в определении слабой частичной подстановки констант: p не задает значение хотя бы одной из соответствующих переменных. Если же среди сыновей v есть некоторая нелистовая вершина w, то требуемое следует из индуктивного предположения для w и условия 2 в определении слабой частичной подстановки констант.

Рассмотрим теперь вершину a. Если у нее есть хотя бы одна вершина-сын v, не являющаяся листом, то все переменные, существенные для $(h_v'')_p$, существенны также и для $(h_a'')_p$. Предположим теперь, что множество W всех сыновей вершины a состоит только из листьев. Вспомним, что их число есть $\lambda(a)$, причем самое большее $(\lambda(a) - m_1)$

из них соответствуют переменным, значения которых задает p. Поскольку ξ задает значения m_1 переменных, буквами которых помечены листья из W, и хотя бы одна из переменных получает значение как посредством p, так и посредством ξ, то как минимум один лист из множества W помечен буквой переменной, значение которой не задает ни p, ни ξ. Эта переменная является, таким образом, существенной для $(h''_a)_p$.

Такие же рассуждения, как и в индукционном переходе выше, показывают, что для всех вершин v, потомками которых является a, получаемая подфункция $(h'_v)_{\xi p} = (h''_v)_p$ обладает хотя бы одной существенной переменной. Следовательно, согласно определению ξ и условию 2 из определения слабой частичной подстановки констант, все переменные, существенные для h', за исключением тех, значения которых задаются подстановкой ξ либо p, существенны также и для h''_p. Это завершает доказательство утверждения 6. □

Последнее вспомогательное утверждение настоящего раздела показывает, что константного числа слабых частичных подстановок достаточно для нахождения переменной из $R(h') \setminus R(h'')$:

Утверждение 7. *Пусть известны бесповторная функция g' и значения величин $r(g'')$ и $r(h'')$. Тогда найдутся две слабых частичных подстановки констант p_1, p_2, обладающие следующим свойством: для любой бесповторной функции g, согласованной с g' и значениями $r(g'')$, $r(h'')$, хотя бы одна из этих двух подстановок задает значение некоторой переменной, не существенной для функции h''.*

Доказательство. Рассмотрим для каждой вершины $v \in A_1$ множество $W(v)$ всех сыновей v, принадлежащих N_1. Если у v есть хотя бы один сын, не являющийся листом, то пусть подстановка p_1 задает значения α_j всех переменных x_{i_j} из $W(v)$. Эти значения выбираются в соответствии с пометкой v так, чтобы выполнялись равенства $R(h'_{x_{i_j} \leftarrow \alpha_j}) = R(h') \setminus \{x_{i_j}\}$ (см. также утверждение 1 в разделе 3). Если же все сыновья v – листья, то пусть каждая из подстановок p_1 и p_2 задает значения $\lambda(v) - m_1 \geqslant 1$ переменных из $W(v)$ таким образом, чтобы общее число переменных, фиксируемых композицией $p_1 p_2$, было наибольшим из возможных. Вспомним, что если $v = a$, то в точности $\lambda(v) - m_1$ переменных из множества $W(v)$ являются существенными для h'', поэтому в каждом подмножестве $W(v)$ размера $\lambda(v) - m_1 + 1$ найдется переменная, не являющаяся существенной для h''. Заметим,

что $(\lambda(v) - m_1 + 1)/(\lambda(v) - m_1) \leqslant 2$. Следовательно, если все переменные из $W(v)$, значения которых задает хотя бы одна из подстановок p_1 и p_2, существенны для h'', то существенными для h'' являются и все остальные переменные из $W(v)$. Утверждение 7 доказано. $\square$

Утверждения 6 и 7 фактически дают алгоритм определения вершины a. Вначале в соответствии с приведенным выше описанием строится частичная подстановка θ и определяются множества N_1 и A_1 (для вычисления значения $m_1 = r(h') - r(h'')$ используется один считающий запрос существенности). После этих подготовительных действий алгоритм строит частичные подстановки констант p_1, p_2 в соответствии с утверждением 7, а затем выполняет считающие запросы существенности для обеих порождаемых подфункций h''_{p_i}.

Обозначим символом s_i количество переменных, значения которых задает подстановка p_i. Если хотя бы одна из этих переменных не является существенной для h'', то, в силу утверждения 6, каждая существенная переменная h'' либо является существенной и для h''_{p_i}, либо получает значение посредством p_i. Таким образом, в этом случае справедливо неравенство $r(h''_{p_i}) + s_i > r(h'')$. С другой стороны, если все переменные, значения которых задает p_i, существенны для h'', то справедливо неравенство $r(h''_{p_i}) + s_i \leqslant r(h'')$. Следовательно, сравнение сумм $r(h''_{p_i}) + s_i$ со значениями $r(h'')$ всегда выявляет подстановку p_i, существование которой гарантируется утверждением 7.

Оставшаяся часть доказательства весьма прямолинейна и состоит в следующем. Поскольку все подмножества (поднаборы) слабой частичной подстановки констант также являются слабыми, то простой бинарный поиск позволяет идентифицировать вершину a с помощью не более чем $\log_2 k$ запросов. Общее число запросов, выполняемых на шаге 4b, не превосходит $\log_2 k + O(1)$. Лемма 2 доказана.

§6. Доказательство леммы 3

Настоящий раздел посвящен доказательству леммы 3. Напомним, что для каждого фиксированного $k = 2, \ldots, n$ имеется известная бесповторная функция f^{k-1} с $R(f^{k-1}) = \{x_1, \ldots, x_{k-1}\}$. Целью является восстановление по ней неизвестной бесповторной функции f^k, для которой $R(f^k) = R(f^{k-1}) \cup \{x_k\}$ и $f^k_{x_k \leftarrow \sigma_k} = f^{k-1}$, где σ_k – известная константа. С учетом предыдущего шага алгоритма известной считается также такая вершина a в известном дереве T' функции f^{k-1}, что

$a = \mathrm{anc}(\mathrm{anc}(x_k))$ в неизвестном дереве T функции f^k (из существования данной вершины вытекает неравенство $k \geqslant 3$).

Напомним, что восстановление f^k проводится в соответствии с леммой 1. Согласно этой лемме, дерево T' может быть получено из дерева T посредством одной из трех явно описанных операций (строго говоря, существует также особый вырожденный случай, но он будет обсуждаться отдельно). Более того, лемма 1 показывает, что для восстановления f^k достаточно лишь множества $R(f^k_{x_k \leftarrow \sigma_k}) \setminus R(f^k_{x_k \leftarrow \overline{\sigma}_k})$ (помимо T'). Доказываемая в настоящем разделе лемма 3 утверждает, что это множество допускает эффективную идентификацию в предположении, что дерево T' известно, а вершина $a = \mathrm{anc}(\mathrm{anc}(x_k))$ найдена заранее. Более строго, для любой бесповторной функции f с n существенными переменными общее число запросов, достаточное для определения множеств $R(f^k_{x_k \leftarrow \sigma_k}) \setminus R(f^k_{x_k \leftarrow \overline{\sigma}_k})$, $k = 3, \ldots, n$, составляет $O(n \log^2 n)$.

Опишем алгоритм, определяющий каждое из этих множеств. Следуя рассуждениям предыдущего раздела, сведем данную задачу к частному случаю с дополнительными ограничениями. Заметим, что все переменные из разности $R(f^k_{x_k \leftarrow \sigma_k}) \setminus R(f^k_{x_k \leftarrow \overline{\sigma}_k})$ содержатся в некоторых из поддеревьев T'_u, где вершины u – сыновья вершины a. Используя одну частичную подстановку констант θ, обеспечим выполнение следующего свойства: все такие поддеревья – отдельные листья (иными словами, дерево неизвестной подфункции $(f^k_{x_k \leftarrow \overline{\sigma}_k})_\theta$ можно получить из дерева подфункции $(f^{k-1})_\theta$ изъятием нескольких листьев с общим родителем a).

Не будем повторять явное построение такой частичной подстановки θ, описанное в разделе 5. Тем не менее, мы будем использовать ранее введенные обозначения: $h = f^k_\theta$, $h' = f^{k-1}_\theta$ и $h'' = (f^k_{x_k \leftarrow \overline{\sigma}_k})_\theta$. Отметим, что величина $m_1 = r(h') - r(h'')$ всегда положительна, а ее значение определяется с помощью одного считающего запроса существенности. Символ T'_1, как и ранее, обозначает дерево функции h'.

Итак, редуцированная постановка задачи состоит в следующем. По входу T'_1 и $a \in T'_1$ необходимо определить множество листьев $u \in T'_1$, смежных с a и помеченных буквами переменных, несущественных для функции h''.

Техника идентификации этого множества близка технике предыдущего раздела. Пусть W – множество всех листьев, смежных с a в T'_1. Все, что делает алгоритм, – это выбирает подмножества множества

W, подставляет подходящие константы на места соответствующих переменных и выполняет запросы для выявления числа существенных переменных порождаемых подфункций. Вкратце, для каждого подмножества $U \subseteq W$ алгоритм может определить число переменных из U, существенных для h'', с помощью одного считающего запроса существенности. После этого остается применить вариант двоичного поиска для окончательной идентификации множества $R(h') \setminus R(h'')$.

Следует, однако, отметить, что для определения количества переменных из U, существенных для h'', не удается непосредственно использовать утверждение 6 из раздела 5. Дело в том, что некоторые нужные на данном шаге частичные подстановки констант p на места переменных U не являются слабыми в смысле упомянутого раздела. Тем не менее, тот факт, что все переменные из U имеют общего родителя в T'_1, позволяет обойти эту трудность:

Утверждение 8. *Пусть $U = \{u_1, \ldots, u_s\}$ – подмножество W и подстановка $p = \{u_1 \leftarrow \alpha_1, \ \ldots \ , \ u_s \leftarrow \alpha_s\}$ присваивает такие значения переменным из U, что $R(h'_{u_i \leftarrow \alpha_i}) = R(h') \setminus \{u_i\}$ при $i = 1, \ldots, s$. Тогда величина*

$$q(U) = |U| - \min\{\, r(h'') - r(h''_p),\ |W| - m_1 \,\},$$

где $m_1 = r(h') - r(h'')$, равна количеству листьев из U, помеченных переменными, несущественными для h''.

Доказательство. Рассмотрим множество $Q = R(h') \setminus R(h'')$ и заметим, что $m_1 = |Q|$. Если $Q \cup U \neq W$, то $r(h'') - r(h''_p) < |W| - m_1$ и $q(U) = |U| - (r(h'') - r(h''_p))$, что составляет в точности число переменных из U, несущественных для h''. Если же $Q \cup U = W$, то $r(h'') - r(h''_p) \geqslant |W| - m_1$. В этом случае $q(U) = |U| - (|W| - m_1) = |U| + |Q| - |Q \cup U| = |Q \cap U|$. Утверждение 8 доказано. □

Таким образом, задача идентификации множества $R(h') \setminus R(h'')$ может быть поручена процедуре $\mathcal{S}$, определенной на Рис. 7 и вызываемой с оракулом для определения значений $q(U)$. После ее работы множество $R(f^k_{x_k \leftarrow \sigma_k}) \setminus R(f^k_{x_k \leftarrow \overline{\sigma}_k})$ восстанавливается без выполнения дополнительных запросов, поскольку произвольная переменная $y \in R(f^{k-1})$ является существенной для $f^k_{x_k \leftarrow \overline{\sigma}_k}$ тогда и только тогда, когда она лежит в таком поддереве T'_u, что $u \in R(h'')$.

Докажем теперь корректность описанного выше алгоритма и анонсированную верхнюю оценку числа запросов, выполняемых на этом

Процедура $\mathcal{S}$

Параметры: множество W и целое число m.
Оракул: функция мощности пересечения $q(\cdot)$.

1. Если $m = 0$, то возвратить $\varnothing$. Если $m = |W|$, то возвратить W.
2. В противном случае выбрать произвольное $U \subseteq W$ такое, что $|U| = \lfloor |W|/2 \rfloor$.
3. Выполнить запрос к оракулу для выявления значения $q_1 = q(U)$.
4. Выполнить два рекурсивных вызова для вычисления $R = \mathcal{S}(U, q_1) \cup \mathcal{S}(W \setminus U, m - q_1)$.
5. Возвратить R.

Рис. 7. Процедура идентификации подмножества конечного множества.

шаге (4c на Рис. 3 для алгоритма $\mathcal{A}_2$). Везде далее символом $\log c$ будем обозначать логарифм числа c по основанию 2. Докажем следующее утверждение, справедливое для каждого отдельного вызова процедуры $\mathcal{S}$.

Утверждение 9. *Пусть W – конечное множество и $Q \subseteq W$, $|Q| = m$. Пусть $q(U) = |Q \cap U|$ для всех $U \subseteq W$. Тогда процедура $\mathcal{S}$ на Рис. 7, вызванная с параметрами W, m и $q(\cdot)$, всегда завершает работу, возвращает Q и выполняет при этом не более* $\min\{m, |W| - m\}\lceil \log |W| \rceil$ *запросов.*

Доказательство. $\mathcal{S}$ есть реализация варианта бинарного поиска на множестве W. При $|W| \leqslant 1$ алгоритм всегда завершает работу без рекурсивных вызовов и возвращает Q. В иных случаях мощность каждого из множеств U и $W \setminus U$ строго меньше мощности $|W|$. Таким образом, из описания $\mathcal{S}$ следует, что для любого конечного множества W процедура $\mathcal{S}$ завершает работу и возвращает Q.

Для доказательства оценки числа выполняемых запросов к оракулу обозначим символом w мощность множества W. Доказательство проведем индукцией по глубине рекурсивных вызовов $\mathcal{S}$. При $m = 0$ и $m = w$ доказываемое неравенство очевидно. Отметим, что в случае $w = 1$ также не выполняется ни одного запроса. Предположим теперь, что $w \geqslant 2$ и $0 < m < w$. Положим по определению $w_1 = \lfloor w/2 \rfloor$ и

$w_2 = \lceil w/2 \rceil$. Кроме того, пусть $q_2 = m - q_1$ и $m_i = \min\{q_i, w_i - q_i\}$, $i = 1, 2$. Докажем, что справедливо следующее неравенство:

$$1 + m_1 \lceil \log w_1 \rceil + m_2 \lceil \log w_2 \rceil \leqslant \min\{m, w - m\} \lceil \log w \rceil.$$

Во-первых, заметим, что $\lceil \log w_1 \rceil \leqslant \lceil \log(w/2) \rceil = \lceil \log w \rceil - 1$. Во-вторых, если $\lceil \log w_2 \rceil > \lceil \log w \rceil - 1$, то $\lceil \log(2\lceil w/2 \rceil) \rceil > \lceil \log w \rceil$, откуда следует, что w не может быть четным. Пусть $w = 2l + 1$, тогда $\lceil \log(2l+2) \rceil > \lceil \log(2l+1) \rceil$. Следовательно, $2l+1$ есть степень двойки. Единственно возможный случай – $w = 1$, но это противоречит исходным предположениям. Следовательно, $\lceil \log w_2 \rceil \leqslant \lceil \log w \rceil - 1$. Таким образом, имеем

$$\begin{aligned} 1 + m_1 \lceil \log w_1 \rceil + m_2 \lceil \log w_2 \rceil &\leqslant 1 + m_1(\lceil \log w \rceil - 1) + m_2(\lceil \log w \rceil - 1) \\ &= (m_1 + m_2)\lceil \log w \rceil - (m_1 + m_2 - 1). \end{aligned}$$

Теперь достаточно доказать неравенство $m_1 + m_2 \leqslant \min\{m, w - m\}$. В самом деле, из этого неравенства и неравенства $m_1 + m_2 - 1 \geqslant 0$ непосредственно вытекает искомое неравенство. (Отметим, что если $m_1 + m_2 < 1$, то $m_1 = m_2 = 0$ и мы доказываем очевидно справедливое неравенство $1 \leqslant \min\{w - m, m\} \lceil \log w \rceil$.) Поскольку $m_i \leqslant q_i$ при $i = 1, 2$, то $m_1 + m_2 \leqslant q_1 + q_2 = m$. Точно так же из неравенств $m_i \leqslant w_i - q_i$ получаем $m_1 + m_2 \leqslant w_1 + w_2 - m = w - m$. Таким образом, $m_1 + m_2 \leqslant \leqslant \min\{m, w - m\}$. Утверждение 9 доказано. □

Утверждение 9 обосновывает корректность нашего алгоритма идентификации множества $R(f^k_{x_k \leftarrow \sigma_k}) \setminus R(f^k_{x_k \leftarrow \overline{\sigma}_k})$, а также демонстрирует, что каждый отдельный вызов процедуры $\mathcal{S}$ может требовать относительно большого числа запросов. В самом деле, если $|W| = \Theta(k)$ и $m_1 = \Theta(k)$, то полученная нами верхняя оценка может иметь вид $\Theta(k \log k)$. Более того, ее нельзя улучшить более чем на полилогарифмический множитель, поскольку число всевозможных подмножеств размера $\lfloor k/2 \rfloor$ множества из k элементов составляет $2^{k(1-o(1))}$.

По этой причине мы будем оценивать общее количество запросов, выполняемых в течение всех таких вызовов. Поскольку параметры каждого следующего вызова в определенной степени зависят от результатов предыдущих, нам понадобится специальный способ учета этой зависимости.

Будем пользоваться следующими обозначениями, связанными с ориентированными графами. Дугу в таком графе, ведущую из вершины u в вершину v, будем обозначать символом $u \to v$; будем также

говорить, что вершина v *следует* за вершиной u. Ориентированные маршруты будем записывать как $v_1 \to v_2 \to \ldots \to v_s$, где $v_1, \ldots, v_s$ – последовательные вершины маршрута. Вершины с полустепенью исхода 0 будем, как обычно, называть *стоками*. *Ветвь* – это маршрут из вершины с полустепенью захода 0 в какой-либо сток.

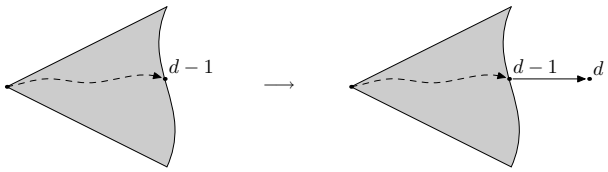
Напомним, что в силу леммы 1 на каждой итерации шага 4 в алгоритме $\mathcal{A}_2$ в основном случае $r(f^k_{x_k \leftarrow \overline{\sigma}_k}) > 0$ существуют три типа операций, преобразующих дерево функции f^{k-1} в дерево функции f^k. В особом случае $r(f^k_{x_k \leftarrow \overline{\sigma}_k}) = 0$ такое преобразование состоит либо в присоединении вершины с буквой x_k к корню дерева (случай 1 в лемме 1), либо во введении нового корня с ровно двумя сыновьями.

Построим последовательность помеченных ориентированных графов $G_2, G_3, \ldots, G_n$ следующим образом. Пусть граф G_2 содержит ровно одну вершину, помеченную значением 1. Пусть для $k = 3, \ldots, n$ граф G_k получается из графа G_{k-1} одним из следующих способов в соответствии с преобразованием деревьев, которое выполняет алгоритм $\mathcal{A}_2$ при восстановлении f_k (см. Рис. 8):

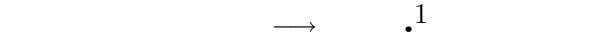
(a) Пусть T получается присоединением листа x_k к существующей вершине v (случай 1 в лемме 1 либо $r(f^k_{x_k \leftarrow \overline{\sigma}_k}) = 0$, если v – корень T'). Пусть $d+1$ есть число вершин – сыновей v в T. Возьмем какой-нибудь сток w с пометкой $d-1$ в G_{k-1} и добавим дугу из w в новую вершину, пометив эту вершину значением d.

(b) Пусть T получается из T' введением нового корня (в этом случае справедливо равенство $r(f^k_{x_k \leftarrow \overline{\sigma}_k}) = 0$) либо вставкой новой внутренней вершины с двумя потомками (случай 2 в лемме 1). В этом случае добавим к графу G_{k-1} новую изолированную вершину, пометив ее значением 1.

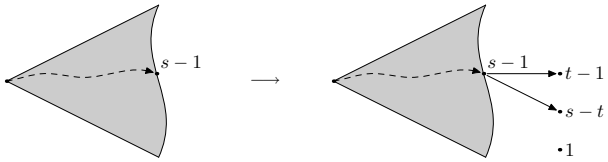
(c) Пусть, наконец, дерево T получается из T' разрезанием внутренней вершины v (случай 3 в лемме 1). Как и ранее, предположим, что v имеет s сыновей в T' и что это разрезание "выделяет" t из них. В этом случае возьмем какой-нибудь сток w с пометкой $(s-1)$ в графе G_{k-1}, добавим дуги из w в две новые вершины и пометим эти вершины значениями $(t-1)$ и $(s-t)$ соответственно. Кроме того, добавим к графу новую изолированную вершину с пометкой 1.



a) В случае присоединения нового листа к существующей вершине.



b) В случае введения нового корня либо добавления новой внутренней вершины с двумя потомками.



c) В случае разрезания существующей вершины.

Рис. 8. Построение графов $G_2, \ldots, G_n$ (показаны только преобразуемые компоненты слабой связности).

Нетрудно проверить, что описанная процедура построения последовательности $G_2, \ldots, G_n$ корректна: пометки стоков G_k суть уменьшенные на 1 количества сыновей внутренних вершин T на шаге 4 алгоритма $\mathcal{A}_2$ для данного k. Положим $L_k = 0$, если для итерации с данным значением k выполняется операция (a) либо (b), и $L_k = \min\{l', l''\}$, если выполняется операция (c) и вводятся две дуги из вершины с пометкой $(l' + l'')$ в вершины с пометками l' и l''.

Утверждение 10. *Для любого $k \geqslant 3$ количество запросов, выполняемых на итерации k шага 4 алгоритма $\mathcal{A}_2$, не превосходит* $(L_k + 1) \cdot \cdot \lceil \log(k-1) \rceil$.

Доказательство. Если $r(f^k_{x_k \leftarrow \overline{\sigma}_k}) = 0$, то процедура $\mathcal{S}$ не вызывается. Пусть $r(f^k_{x_k \leftarrow \overline{\sigma}_k}) \neq 0$, тогда процедура $\mathcal{S}$ должна идентифицировать подмножество размера m_1 в множестве вершин (листьев в T'_1),

смежных с a. Пусть w – количество таких вершин. В силу утверждения 9 процедура $\mathcal{S}$ выполняет не более $\min\{m_1, w-m_1\}\lceil\log w\rceil$ запросов. Если $m_1 = 1$, то для графа G_{k-1} выполняется операция (a) или (b). В этом случае $L_k = 0$ и доказываемая оценка очевидна. Если $m_1 \geqslant 2$, то выполняется операция (c), величины l' и l'' суть $m_1 - 1$ и $w - m_1$, поэтому $L_k + 1 = \min\{m_1, w - m_1 + 1\}$. Утверждение 10 доказано. $\square$

Определим для графа G_n величину

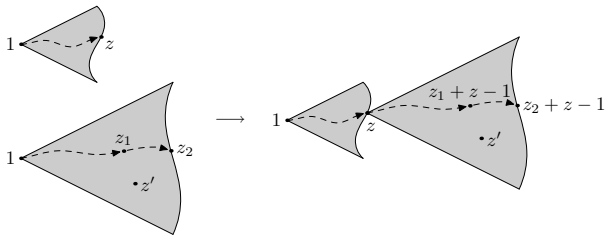
$$L(G_n) = \sum_{k=3}^{n} L_k \ = \sum_{\substack{v \to v_1 \\ v \to v_2}} \min\{L(v_1), L(v_2)\},$$

где $L(v)$ есть пометка вершины v в графе G_n и сумма в последнем выражении берется по всем вершинам $v \in G_n$ с полустепенью исхода 2 (для каждой такой вершины v через v_1 и v_2 обозначены вершины, следующие за v).

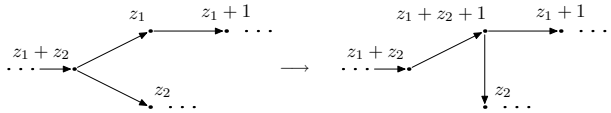
Утверждение 11. $L(G_n) \leqslant n \log n$.

Доказательство. Заметим, что граф G_n представляет собой лес, состоящий из корневых деревьев, ориентированных от корней к листьям. Если G_n состоит более чем из одного дерева, то преобразуем его в дерево G'_n последовательным отождествлением корней отдельных деревьев с листьями других деревьев (см. Рис. 9a). Если при таком отождествлении пометка листа u есть $z > 1$, тогда выбирается какая-либо произвольная ветвь $u \to u_1 \to u_2 \to \ldots \to u_s$ из u в сток u_s в получающемся графе и пометки всех ее вершин u_i увеличиваются на величину $(z-1)$. В результате пометка любой вершины v с полустепенью исхода 2 в дереве G'_n равна сумме пометок вершин, следующих за v. Кроме того, пометка любой вершины u с полустепенью исхода 1 на единицу меньше пометки вершины, следующей за u. Ясно, что $L(G_n) \leqslant L(G'_n)$.

Предположим теперь, что в дереве G'_n существует вершина v_0 с полустепенью исхода 2 и такие дуги $v_0 \to v_1$, $v_0 \to v_2$, что за вершиной v_1 следует лишь одна вершина v'_1. Пусть пометки v_1 и v_2 суть z_1 и z_2, тогда вершина v_0 помечена $z_1 + z_2$, а вершина v'_1 помечена $z_1 + z_2 + 1$. Преобразуем дерево, заменив дугу $v_0 \to v_2$ на дугу $v_1 \to v_2$ (см. Рис. 9b).Для сохранения прежних свойств пометок заменим пометку вершины v_1 на $z_1 + z_2 + 1$. Заметим, что такое преобразование либо вовсе не изменяет величины $L(\cdot)$, либо увеличивает ее на 1.



a) Отождествление корня с листом.



b) Перебрасывание дуги.

Рис. 9. Преобразования графов в доказательстве утверждения 11.

Будем выполнять такие преобразования до тех пор, пока это возможно. Полученное в результате дерево G_n'' будет состоять из простого пути по вершинам полустепени исхода 1 из корня G_n'' в корень u некоторого строго двоичного поддерева (не содержащего вершин полустепени исхода 1). Согласно приведенным выше рассуждениям, справедливо неравенство $L(G_n') \leqslant L(G_n'')$. Оценим сверху величину $L(G_n'')$.

Пусть s – пометка вершины u. Докажем, что $L(G_n'') \leqslant s \log s$. Проведем доказательство методом математической индукции по s. В случае $s = 1$ оценка очевидна. Если $s = 2$, то $L(G_n'') \leqslant 1 \leqslant 2 \log 2$. Пусть $s \geqslant 3$, в G_n'' присутствуют дуги $u \to u_1$, $u \to u_2$ и пометки вершин u_1, u_2 суть x и $s - x$. Не ограничивая общности рассуждений, предположим, что $x \leqslant s/2$. Положим по определению $\phi(x) = x \log x$ и $\Phi(x) = x + \phi(x) + \phi(s - x)$. По предположению индукции справедливо неравенство $L(G_n'') \leqslant \Phi(x)$. Покажем, что $\Phi(x) \leqslant s \log s$ для всех $x \in [1; s/2]$. Рассмотрим производные $\phi'(x) = (\ln x + 1)/\ln 2$ и

$$\Phi'(x) = 1 + \frac{(\ln x + 1) - (\ln(s - x) + 1)}{\ln 2} = 1 + \log \frac{x}{s - x} = \log \frac{2x}{s - x}.$$

Так как $x \leqslant s/2$, то неравенство $\Phi'(x) \geqslant 0$ равносильно неравенству $x \geqslant s/3$. Это означает, что достаточно рассмотреть значения $x = 1$ и

$x = s/2$. Имеем:

$$\Phi(1) = 1 + 0 + (s-1)\log(s-1) \leqslant \log s + (s-1)\log s = s\log s,$$
$$\Phi(s/2) = \frac{s}{2} + 2 \cdot \frac{s}{2}\log\frac{s}{2} = \frac{s}{2} + s\log s - s \leqslant s\log s.$$

Утверждение 11 доказано. □

В силу утверждений 10 и 11 число запросов, выполняемых алгоритмом $\mathcal{A}_2$ на шаге 4c, не превосходит величины

$$\begin{aligned}\sum_{k=3}^{n}(L_k+1)\lceil\log(k-1)\rceil &\leqslant \left(\sum_{k=3}^{n} L_k + n\right)\cdot(\log n + 1)\\ &\leqslant (n\log n + n)\cdot(\log n + 1)\\ &= O(n\log^2 n).\end{aligned}$$

Лемма 3 полностью доказана.

§7. Заключение

В статье разработаны два алгоритма расшифровки (точной идентификации) неизвестной бесповторной функции. Первый алгоритм $\mathcal{A}_1$ выполняет $O(n^2)$ запросов типа да/нет. Второй алгоритм $\mathcal{A}_2$ выполняет $O(n\log^2 n)$ запросов с ответами логарифмической и константной длины. (На самом деле количество используемых каждым из алгоритмов запросов принадлежности равно одному, причем этот запрос можно заменить любой корректной парой $\langle\alpha, f(\alpha)\rangle$, подаваемой на вход.) Запросная сложность второго алгоритма близка к мощностной нижней оценке, составляющей $\Omega(n\log n)$ бит. Второй алгоритм фактически выполняет запросы специальной характеристики арифметической суммы $S(f_p)$ всех значений подфункции f_p для произвольной частичной подстановки констант p – наибольшего целого k, для которого 2^k делит $S(f_p)$.

Само значение $S(f_p)$, как видно, оказывается ключевым для процесса точной идентификации, выполняемой алгоритмом. Для другой постановки задачи, когда разрешены только запросы четности $S(f_p)$ (суммы по модулю 2) [4], известный алгоритм имеет запросную сложность $n^2(1-o(1))$. Если дополнительно разрешены запросы второго младшего бита $S(f_p)$ [8], то это значение можно понизить до $3n^2/4 \cdot (1-o(1))$, что по-прежнему составляет $\Theta(n^2)$, в то время как в настоящей работе получена оценка $O(n\log^3 n)$.

Автор признателен Андрею Анатольевичу Вороненко, который поставил исходную задачу расшифровки с запросами к оракулу, возвращающему значения $S(f_p)$. Кроме того, автор благодарит своих коллег Максима Александровича Башова и Владимира Владимировича Лысикова за полезные и вдохновляющие обсуждения.

Литература

1. А. В. Ахо, Д. Э. Хопкрофт, Д. Д. Ульман, *Структуры данных и алгоритмы.* М.: Вильямс, 2007. 400 с.
2. А. А. Вороненко, *О задачах глобального тестирования (расшифровки) бесповторных булевых функций.* – Синтаксис и семантика логических систем: материалы 3-й Российской школы-семинара. Иркутск: Изд-во Восточно-Сибирской государственной академии образования, 2010. 17–22.
3. А. А. Вороненко, *О проверяющих тестах для бесповторных функций.* – Математические вопросы кибернетики. Вып. 11. М., Физматлит (2002), 163–176.
4. А. А. Вороненко, Д. В. Чистиков, *Расшифровка бесповторных функций оракулом – счетчиком четности.* — Прикладная математика и информатика. Вып. 34, М., МАКС Пресс (2010), 93–106. (A. A. Voronenko, D. V. Chistikov, *Learning read-once functions using subcube parity queries.* — Computational Mathematics and Modeling, **22**, No. 1 (2011), 81–91.)
5. В. А. Гурвич, *О бесповторных булевых функциях.* — Усп. матем. наук, **32**, No. 1 (1977), 183–184.
6. В. А. Стеценко, *О предплохих базисах в P_2.* — Математические вопросы кибернетики. Вып. 4. М., Физматлит (1992), 139–177. (V. A. Stetsenko, *On almost bad Boolean bases.* — Theor. Comput. Sci., **136**, No. 2 (1994), 419–469.)
7. Б. А. Субботовская, *О сравнении базисов при реализации функций алгебры логики формулами.* — Докл. АН СССР, **149**, No. 4 (1963), 784–787.
8. Д. В. Чистиков, *Расшифровка бесповторных функций по двум младшим битам числа единиц подфункций.* — Синтаксис и семантика логических систем: материалы 3-й Российской школы-семинара. Иркутск: Изд-во Восточно-Сибирской государственной академии образования (2010), 114–118.
9. D. Angluin, *Queries and concept learning.* — Machine Learning, **2** (1987), 319–342.
10. D. Angluin, L. Hellerstein, M. Karpinski, *Learning read-once formulas with queries.* — J. ACM, **40** (1993), 185–210.
11. D. V. Chistikov, *Checking tests for read-once functions over arbitrary bases.* — Proc. CSR 2012, Lecture Notes in Computer Science, **7353** (2012), 52–63.
12. M. Karchmer, N. Linial, I. Newman, M. Saks, A. Widgerson, *Combinatorial characterization of read-once formulae.* — Discrete Mathematics, **114**, No. 1–3 (1993), 275–282.
13. P. Savicky, A. R. Woods, *The number of Boolean functions computed by formulas of a given size.* — Random Structures and Algorithms: Proc. of the Eighth International Conference, **13**, Nos. 3–4 (1998), 349–382.
14. L. G. Valiant, *A theory of the learnable.* — Communications of the ACM, **27** (1984), 1134–1142.

Chistikov D. V. Using relevance queries for identification of read-once functions.

A Boolean function is called read-once if it can be expressed by a formula over $\{\&, \vee, \overline{\ }\}$ where no variable appears more than once. The problem of identifying an unknown read-once function f depending on a known set of variables $x_1, \ldots, x_n$ by making queries is considered. Algorithms are allowed to perform standard membership queries and queries of two special types, allowing to reveal the relevance of variables to projections of f. Two exact identification algorithms are developed: one makes $O(n^2)$ yes–no queries, and the other makes $O(n \log^2 n)$ queries with logarithmically long answers. Information-theoretic lower bound on the number of bits transferred from oracles to identification algorithms in the worst case is $\Omega(n \log n)$.

Факультет ВМК
МГУ имени М. В. Ломоносова
Ленинские горы
Москва 119991 ГСП-1, Россия
E-mail: dch@cs.msu.ru

Поступило 19 декабря 2011 г.