Г. А. ЧИСТЯКОВ

ОСНОВЫ ТЕОРИИ АЛГОРИТМОВ

Учебное пособие

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных систем
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

Г. А. ЧИСТЯКОВ

ОСНОВЫ ТЕОРИИ АЛГОРИТМОВ

Учебное пособие

Киров
2018

УДК 510.51(07)
Ч-689

Рекомендовано к изданию методическим советом факультета автоматики и вычислительной техники ВятГУ

Допущено редакционно-издательской комиссией методического совета ВятГУ в качестве учебного пособия для студентов направлений 09.03.01 «Информатика и вычислительная техника» и 09.03.03 «Прикладная информатика» всех профилей подготовки, всех форм обучения

Рецензенты:
канд. техн. наук, доцент кафедры САУ ВятГУ,
В. И. Семеновых

канд. техн. наук, заместитель управляющего по операционным вопросам филиала банка ВТБ (ПАО) в г. Кирове
О. В. Рыков

Чистяков, Г. А.
Ч-689 Основы теории алгоритмов : учебное пособие / Г. А. Чистяков. – Киров : ВятГУ, 2018. – 96 с.

Учебное пособие предназначено для студентов, изучающих дисциплину «Математическая логика и теория алгоритмов».

УДК 510.51(07)

Содержание

Введение

Теория алгоритмов является одной из наиболее интересных дисциплин в сфере computer science, существующей на стыке информатики и математики. В современных реалиях трудно представить себе высококвалифицированного специалиста в области вычислительной техники, не знакомого с такими терминами как сложность, вычислимость, неразрешимость; не способного оперировать инструментарием абстрактных алгоритмических систем и схем алгоритмов. Все эти понятия, являющиеся предметом изучения теории алгоритмов, — необходимый фундамент для освоения целого ряда профессиональных дисциплин.

В данной работе, подготовленной по материалам лекционного курса дисциплины «Математическая логика и теория алгоритмов», читаемого студентам направления 09.03.01 — Информатика и вычислительная техника на кафедре ЭВМ Вятского государственного университета, рассматриваются базовые вопросы и проблемы теории алгоритмов.

Структурно пособие разделено на четыре части.

В первой из них кратко рассматривается история возникновения дисциплины, осуществляется экскурс в ее проблематику, а также вводится ряд основных понятий и определений.

Вторая, наиболее объемная, часть позволяет проследить путь становления дисциплины через призму развития оказавших на нее значительное влияние алгоритмических систем.

В третьей части пособия подробно рассматривается проблема существования класса алгоритмически неразрешимых задач.

Заключительная, четвертая, часть посвящена аспектам конструирования и оперирования схемами алгоритмов и в определенной степени является «переходным мостиком» между теорией алгоритмов и математической логикой.

При написании работы частично использовались материалы профессора кафедры ЭВМ Вятского государственного университета В.Д. Матвеева.

Пособие предназначено для студентов младших курсов, заинтересованных в получении навыков программирования и изучении принципов функционирования вычислительной техника.

1 Введение в теорию алгоритмов

1.1 История термина «алгоритм»

Понятие алгоритма является одним из фундаментальных в программировании и теории обработки информации. Считается, что слово «алгоритм» произошло от имени выдающегося математика средних веков аль-Хорезми (Абу Абдаллах Мухаммеад ибн Муса аль-Хорезми), жившего ориентировочно в 783-850 гг. Будучи одним из ведущих ученых «дома Мудрости», прообраза академии наук, арабского халифата в Багдаде, он, среди прочих трудов по истории, географии, астрономии, опубликовал около 825 г. трактат «Книга об индийской арифметике», где обобщил правила десятичных операций сложения, вычитания, умножения, деления. Подлинник данного труда не сохранился, однако существует его перевод на латинский язык — «Dixit Algorizmi». В дальнейшем все десятичные вычисления стали называть «алгоризм» или «алгорисмус». Затем слово трансформировалось в «алгорифм», а во второй половине XX века — в «алгоритм».

Таким образом, в XX веке под алгоритмом стали понимать конечную совокупность точно сформулированных правил, которые позволяют решать те или иные классы задач. А общепринятым стало следующее определение: «Алгоритм — конечный набор правил, позволяющий чисто механически решать любую конкретную задачу из некоторого класса однотипных задач».

Ясно, что это определение не является математическим: оно не содержит строгого понятия, точной характеристики того, что понимается под классом задач и под правилами решения.

До 30-ых годов XX века это устраивало всех, общей теории алгоритмов не существовало. Если какой-либо исследователь утверждал, что для некоторого класса задач существует алгоритм, он просто приводил пример. Но со временем положение изменилось, стали появляться проблемы с поиском алгоритмов; впервые встал вопрос об отсутствии алгоритмов для некоторых классов задач. В этой ситуации интуитивного, неформального определения стало явно недостаточно. Поэтому задача строгого определения понятия «алгоритм» стала одной из центральных математических проблем.

1.2 Алфавитный оператор

Под алгоритмом в современной математике понимают конструктивно задаваемые соответствия между словами в абстрактных алфавитах.

Абстрактный алфавит — любая конечная совокупность объектов, называемых буквами. Это могут быть буквы какого-либо языка (русского, английского), цифры, любые знаки, рисунки, вещи и так далее.

Слово в алфавите — любая упорядоченная последовательность букв. Пусть $A = \{x, y\}$, тогда словами могут быть: x, y, xy, yx, xx, xxx и так далее. Множество слов в абстрактном алфавите бесконечно, при этом каждое слово имеет длину (число букв): 1, 1, 2, 2, 2, 3 соответственно. Существует также специальное слово нулевой длины — пустое слово, обозначаемое чаще всего как e или λ.

Примечание. *Русский алфавит, если он принят в качестве абстрактного, имеет всевозможные наборы букв как равные слова (дом, собака, яма, ъъъ и так далее).*

Любой алфавит может быть расширен путем добавления новых букв, и тогда понятие слова изменяется. Например, $A_1 = \{а, б, ..., я\}$, $A_2 = \{A_1, \#, \&, !, ...\}$. Сочетание букв: «лодка #вошла& в бухту», — теперь можно считать одним словом. Дополнив русский алфавит другими символами, можно в расширенном алфавите рассматривать как одно слово — абзац, параграф, главу и так далее.

Определение. *Алфавитным оператором (или отображением) называется всякое соответствие (или функция), сопоставляющее словам в том или ином алфавите слова в том же или другом алфавите. Первый алфавит называется при этом входным, а второй — выходным алфавитом данного оператора.*

На практике наиболее часто используются однозначные алфавитные операторы, когда каждому входному слову сопоставлено одно выходное слово. Если оператор не сопоставляет входному слову P ни одного выходного слова (в том числе пустого), то говорят, что он не определен на этом слове. Совокупность всех слов на которых определен алфавитный оператор называется областью определения. Следует отметить, что входной и выходной алфавиты могут быть объединены в один общий алфавит. Это

позволяет рассматривать любой оператор, как оператор объединенного алфавита, заданный лишь на тех словах, что входят в область определения до объединения.

С понятием оператора интуитивно связано понятие сложности. Наиболее простые операторы те, которые осуществляют побуквенное отображение: каждая буква слова P, составленного из букв входного алфавита $A = \{x_1, x_2, ..., x_n\}$ заменяется некоторой буквой выходного алфавита $\{y_1, y_2, ..., y_n\}$. Побуквенное отображение полностью определяется заданием соответствия между буквами входного и выходного алфавита.

Более сложными являются кодирующие операторы, когда слова в алфавите A кодируются словами в другом алфавите B так: каждой букве $a_i \in A$ сопоставляется конечная последовательность $b_{i,1}, b_{i,2}, ..., b_{i,k}$ из алфавита B. Для построения кодирующего отображения достаточно заменить все буквы в слове P соответствующими кодами. Полученное после этого слово называется кодом слова P.

Обязательным условием кодирования является его обратимость, то есть различные слова в алфавите A должны иметь разные коды для обеспечения взаимной однозначности. Но для обратимости мало одного условия различия кодов, так как если a_1 сопоставить код bb, а a_2 — b, то код bbb будет соответствовать словам a_1a_2, a_2a_1 и $a_2a_2a_2$.

Следовательно, необходимыми и достаточными условиями являются.

- Коды различных букв алфавита A различны.
- Код любой буквы не должен совпадать с начальными отрезками кодов других букв (ни один код не должен являться префиксом другого кода).

Если коды всех букв A имеют одинаковую длину, то кодирование называется нормальным. Оно позволяет сводить все отображения к одному из стандартных алфавитов, например, двоичному.

Понятие алфавитного оператора является чрезвычайно общим, так что к нему можно свести любые процессы обработки информации. Для некоторых специальных видов информации (лексической, цифровой) алфавитный способ задания — естественный, он применяется постоянно. Например, задача перевода текстов с одного языка на другой (правда с определенными оговорками, и возможно оператор может быть не однозначным, а многозначным).

Кроме того, к этому типу задач сводятся задачи редактирования текстов, реферирования статей, поиска по ключевым словам и многие другие. Используя шахматную нотацию, можно описывать шахматные партии в виде слов. Известны опыты по сочинению стихов и музыки с помощью ЭВМ.

А как быть в случае отображений, когда входные слова имеют непрерывную природу? Может показаться, что для характеристики преобразований непрерывной информация (зрительной, слуховой) понятие алфавитного оператора окажется недостаточным, однако это не так.

Из практики известно, что прием и преобразование непрерывной информации осуществляется приборами, которые имеют неидеальные параметры, то есть они не различают слишком малые величины. Всегда существует ряд ограничений, которые можно проиллюстрировать на примере восприятия зрительной информации.

Ограничение первое — разрешающая способность прибора, принимающего непрерывную информацию. Достаточно близкие между собой точки участка пространства (рисунка, фотографии и тому подобное) воспринимаются как одна. Следовательно, прибор воспринимает не непрерывную, а дискретную информацию — конечное множество точек.

Ограничение второе — чувствительность прибора. Данное ограничение приводит к тому, что прибор способен различать лишь конечное число уровней интенсивности, цвета или яркости.

Третье ограничение — пропускная способность прибора (в отличие от предыдущих — динамическая характеристика). Полоса пропускания прибора всегда ограничена, например, малое изменение скорости движения не может быть отмечено и зафиксировано.

Отсюда вывод: любой прибор (в том числе и человеческий глаз), в силу своих ограничений, в каждый конкретный момент времени способен воспринимать лишь одну картинку из конечного множества картин мгновенного распределения анализируемой информации в пространстве. Если обозначить через n число точек пространства, воспринимаемых прибором как отдельные элементы информации, а через m — число уровней физической величины, несущей информацию, то можно показать, что число букв в алфавите A будет равно m^n. Конечно, это число может оказать очень большим, но оно конечно, а значит принципиально воспроизводимо.

Отметим, что подобные рассуждения применимы не только к входной,

но и к выходной информации. Тогда, справедливо следующее заключение.

Лемма. *Любой реальный прибор, преобразующий информацию, может рассматриваться как устройство, реализующее некоторый алфавитный оператор.*

Алфавитный оператор, реализуемый прибором, полностью (с точностью до кодирования) определяет информационную сущность этого прибора. В этом смысле и была подчеркнута чрезвычайная общность понятия «алфавитный оператор», так как к этому понятию может быть сведена теория любых преобразований информации.

В основе теории алфавитных операторов лежит способ задания операторов. Если область определения оператора конечна, то вопрос его задания теоретически не вызывает затруднений и может быть решен составлением таблицы соответствий. В левой части данной таблицы слова, входящие в область определения, а в правой — выходные слова, полученные путем его применения ко входным. При большой мощности области определения таблица может оказаться очень громоздкой, нереализуемой практически.

В ситуации, когда область определения бесконечна, применение табличного способа принципиально невозможно. В таком случае требуется отыскать способ задания оператора на бесконечном множестве входных слов, не строя никаких таблиц соответствия. Пример такого способа:

$$x_1x_2...x_n \to y_1y_2...y_ny_{n+1},$$

где $n = 0, 1, 2, ...$

Вместо таблицы соответствия эта формула описывает правило, при помощи которого за конечное число шагов по входному слову можно установить выходное слово.

Во всех случаях описания алфавитного оператора с бесконечной областью определения вместо таблицы соответствия приходится задавать конечное число правил, позволяющих за конечное число шагов найти для любого входного слова соответствующее ему выходное.

Определение. *Алфавитные операторы, задаваемые с помощью конечных систем правил, называются алгоритмами.*

Из определения вытекает, что любой алфавитный оператор, который можно задать, непременно оказывается алгоритмом. Тем не менее, алфа-

витный оператор задает только соответствие между словами двух алфавитов A и B; в отличие от него, алгоритм дает возможность по входному слову найти выходное слово, то есть первый отвечает на вопрос «Что получено?», а второй — «Как получено?». Другими словами, алгоритм есть не что иное, как алфавитный оператор вместе с правилами, определяющими его действие.

Определение. *Два алфавитных оператора считаются равными, если у них одна область определения и любому входному слову из этой области они сопоставляют одинаковые выходные слова.*

Определение. *Два алгоритма считаются равными, если их алфавитные операторы равны, а системы правил совпадают. В случае, когда системы правил различны, алгоритмы называются эквивалентными.*

При решении некоторых практических задач требуется расширить понятие алгоритма, вводя в систему правил его реализации такие механизмы, которые позволяют выбирать входные слова случайным образом, так же как и сами правила. Вероятность выбора может фиксироваться заранее или определяться по ходу выполнения алгоритма. Такие алгоритмы называют стохастическими, или случайными; они приводят к многозначным алфавитным операторам. В этом случае для любого входного слова P из области определения алгоритма однозначно формируется набор вероятностей появления одного из выходных слов q_i. При этом сами вероятности появления выходных слова q_i не должны изменяться в процессе реализации алгоритма.

Наконец, нужно отметить как отдельный класс самоизменяющиеся алгоритмы, которые не только перерабатывают входные слова, но и сами претерпевают изменения в процессе переработки. Результат обработки входного слова P такого алгоритма зависит не только от этого слова, но и от предыстории работы алгоритма — конечной последовательности слов $\{P_i, P_{i+1}, ..., P_{i+n}\}$, которые поступили до появления слова P.

Понятие самоизменяемости применимо в общем случае как к детерминированным, так и к стохастическим алгоритмам. В последнем случае это может выразиться в изменении вероятностей различных выходных слов. Самоизменяемые алгоритмы удобно представлять в виде двух алгоритмов, когда первый (рабочий) осуществляет переработку входных слов, а второй (управляющий) вносит изменения.

Схематичное изображения алгоритма как отображения представлено на рис. 1.

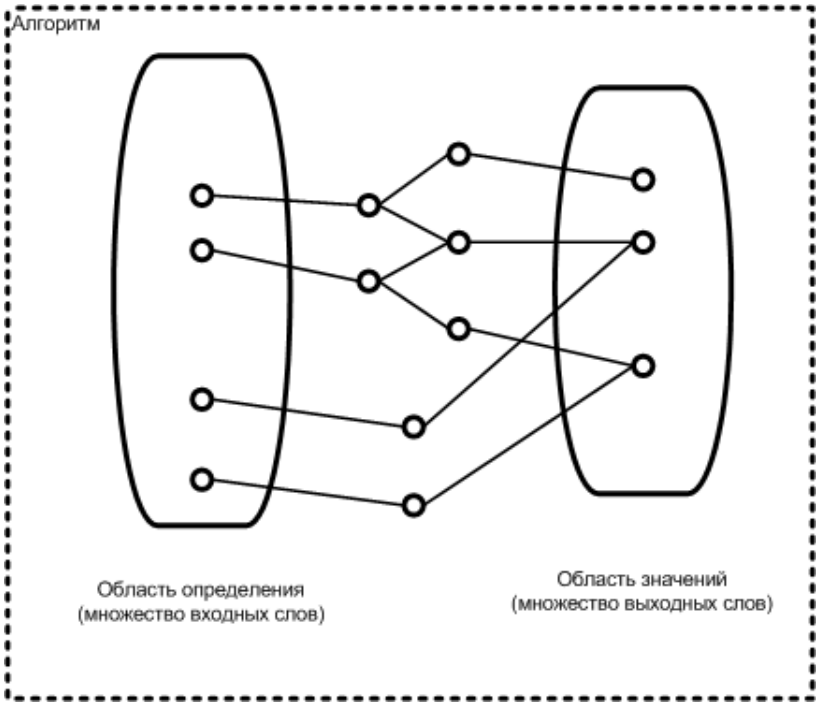

Рис. 1 — Схематичное изображение алгоритма

В общей теории алгоритмов в основном рассматриваются алгоритмы, операторы которых имеют обозначенное соответствие.

1.3 Примеры однозначных и многозначных алфавитных операторов

Как было отмечено выше, на практике чаще всего применяются однозначные алфавитные операторы. Суть подобных операторов заключается в однозначном установлении соответствия между элементами множества входных слов (область определения) и элементами множества выходных слов (область значения).

Примером однозначного алфавитного оператора может служить попарное отображение элементов множества $\overline{P} = \{1, 2, 3, ...\}$ в элементы множества $\overline{Q} = \{1, 2, 6, ...\}$. Очевидно, что для любого положительного p_i соответствующий q_i может быть вычислен как $p_i!$.

Тем не менее, нередки случаи использования многозначных алфавитных операторов, которые позволяют сопоставить некоторому входному слову любой элемент из подмножества множества выходных слова (или наобо-

рот — подмножеству множества входных слов сопоставляют одно выходное слово).

Примером многозначного алфавитного оператора может служить отображение элементов множества $\overline{P} = \{(1,2,3),(1,3,2),(2,1,3),...\}$ в элементы множества $\overline{Q} = \{(1,2,3)\}$. По сути, каждое слово p_i представляет собой перестановку чисел $1,2,3$. Описываемый алфавитный оператор ставит в соответствие каждому p_i перестановку $1,2,3$. Тем самым, осуществляя сортировку заданного массива по возрастанию.

1.4 Свойства алгоритмов

К основным следует отнести перечисленные ниже свойства алгоритмов.

- Дискретность — процесс сопоставления входному слову выходного должен быть представим как последовательное выполнение конечного числа некоторых шагов. Кроме того, для выполнения каждого шага должно требоваться конечное время.

- Детерминированность — в каждый момент времени следующий шаг процесса сопоставления должен быть однозначно определяем. Кроме того, каждому входному слову P должно соответствовать только одно выходное слово Q. Любой стохастический алгоритм может быть представлен как детерминированный путем включения вероятностных характеристик в состав алгоритма.

- Понятность — все шаги процесса сопоставления должны быть доступны для осуществления исполнителю.

- Завершаемость — для любого входного слова из области определения алгоритм должен выполнять процесс сопоставления ему выходного слова за конечное число шагов.

- Массовость — алгоритм должен быть применим для решения не только одной задачи, а целого класса однотипных задач (мощность области определения должна быть больше единицы).

- Результативность — в конечном итоге применение алгоритма должно приводить к получению выходного слова.

1.5 Алгоритмическая система

В теории алгоритмов большое внимание уделяется общим способам задания алгоритмов универсального типа, позволяющим сконструировать любой наперед заданный алгоритм.

Определение. *Всякий общий способ задания алгоритмов называется алгоритмической системой.*

При описании алгоритмической системы используют специальные формальные средства — формализмы. Все направления в прикладной теории алгоритмов условно делятся на два направления — алгебраические и геометрические.

Алгебраические формальные средства используют символику, когда алгоритмы рассматриваются в виде линейных текстов (рекурсивные функции, операторные системы Ван-Хао, логические схемы алгоритмов Ляпунова, Янова и другие).

В геометрической теории алгоритмы строятся в виде множеств, в которых реализуются отображения или бинарные отношения, а также используется математический аппарат теории графов (нормальные алгоритмы Маркова, граф-схемы алгоритмов, матричные схемы алгоритмов, блок-схемный способ и прочее).

2 Абстрактные алгоритмические системы

2.1 Машина Поста

Наиболее простой формальной алгоритмической системой является машина Поста, предложенная американским математиком польского происхождения Эмилем Леоном Постом.

2.1.1 Формальное определение машины Поста

Прежде всего необходимо отметить, что машина Поста является абстрактным вычислителем, и, следовательно, не может быть реализована в реальной жизни. Однако, исследование подобной машины можно проводить с помощью различного рода эмуляторов.

С точки зрения структуры машины Поста может быть представлена как совокупность двух элементов.

- Лента.
- Каретка (считывающая и записывающая головка).

Лента представляет собой бесконечную полоску, разделенную на секции (ячейки) равной длины. Для наглядности, можно считать, что лента расположена горизонтально. В каждой секции ленты присутствует или отсутствует отметка. Таким образом, секции ленты называются отмеченными и неотмеченными. Кроме того, каждой секции ленты сопоставляет уникальное целое число $..., -3, -2, -1, 0, 1, 2, 3, ...$ Схематично изображение машины представлено на рис. 2.

Каретка может перемещаться вдоль ленты, передвигаясь за один шаг на одну секцию влево или вправо. Каретка способна ставить отметку в секцию, напротив которой находится, или стирать отметку в находящейся напротив секции.

Определение. *Жестко сопоставленные каждой секции уникальные номера образуют целочисленную базовую систему координат машины Поста.*

Определение. *Временной системой координат машины Поста называется полученная в результате сдвига на фиксированное число относительно базовой система координат.*

Определение. *Информация о том, какие секции отмечены, а какие не отмечены называется состоянием ленты.*

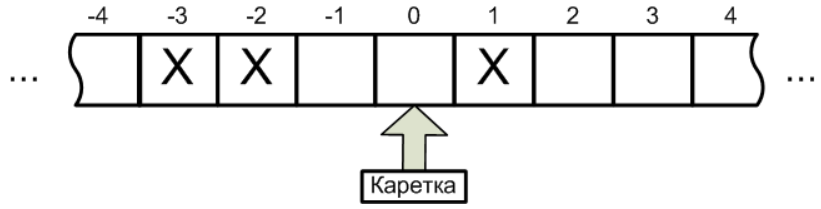

Рис. 2 — Схематичное изображение машины Поста

Работа машины Поста состоит в том, что каретка передвигается вдоль ленты и печатает или стирает метки. Этот процесс происходит согласно инструкциям определенного вида. Пронумерованная натуральными числами совокупность таких инструкций называется программой машины Поста.

Каждая инструкция, называемая иногда также командой, должна относиться к одному из шести типов.

- $l\ \alpha$ — команда сдвига каретки влево. После выполнения действия осуществляется переход к команде с номером α.
- $r\ \alpha$ — команда сдвига каретки вправо. После выполнения действия осуществляется переход к команде с номером α.
- $p\ \alpha$ — команда постановки отметки в секцию, напротив которой находится каретка. После выполнения действия осуществляется переход к команде с номером α.
- $e\ \alpha$ — команда удаления отметки из секции, напротив которой находится каретка. После выполнения действия осуществляется переход к команде с номером α.
- $j\ \alpha\ \beta$ — команда передачи управления. Если напротив каретки находится неотмеченная секция, то осуществляется переход к команде с номером α, иначе — к команде с номером β.
- s — команда остановки.

Определение. *Номер следующей команды, расположенный в конце каждой инструкции, называется отсылкой.*

Определение. *Первый номер следующей команды в инструкции передачи управления называется верхней отсылкой, второй — нижней.*

Следует заметить, что в начале выполнения программы каретка располагается напротив секции с нулевым номером в базовой системе координат.

Уточним определение термина «программа машины Поста».

Определение. *Программой машины Поста называется конечный непустой список команд, обладающий двумя свойствами. Во-первых, на k-ой позиции списка должна стоять команда с номером k. Во-вторых, отсылка любой из команд списка должна совпадать с номером некоторой (другой или той же самой) команды списка.*

Для некоторой программы и начального состояния ленты работа машины Поста может закончиться одним из двух вариантов.

- В ходе выполнения программы машина дойдет до выполнения невыполнимой команды (печать метки в отмеченной секции, удаление метки из неотмеченной секции, отсылка к команде с номером, превышающим число инструкций в программе). В таком случае выполнение программы прекращается. Происходит так называемая безрезультатная остановка.

- В ходе выполнения программы машина дойдет до выполнения команды остановки. В таком случае программа считается выполненной, машина останавливается, происходит результативная остановка.

Кроме того, возможен еще один вариант. В ходе выполнения программы машина не дойдет до выполнения ни одной команды остановки, также не произойдет безрезультатной остановки. Таким образом, выполнение программы никогда не прекратится — процесс работы будет происходить бесконечно.

Необходимо ввести еще два определения.

Определение. *Совокупность положения каретки в базовой системе координат и номер текущей команды в выполняемой программе называется состоянием каретки.*

Определение. *Совокупность состояния ленты и состояния каретки называется состоянием машины Поста.*

Состояние машины Поста позволяет однозначно определить следующее выполняемое действие. Фактически, процесс работы машины Поста заключается в последовательных переходах машины от одного состояния к другому до тех пор, пока не произойдет останов.

2.1.2 Пример использования машины Поста

Рассмотрим подробнее процесс работы машины Поста на следующем примере. Пусть на ленте записано некоторое натуральное число. Требуется прибавить к этому числу единицу.

Для записи числа на ленте будем использовать последовательность отмеченных секций — $n+1$ подряд идущая отмеченная секция соответствует числу n.

Для начала введем одно ограничение, которое позволит значительно упростить алгоритм решения задачи: пусть в начальном положении каретка находится напротив отмеченной секции или же напротив секции, расположенной слева от записи числа на ленте.

Одна из возможных программа для решения поставленной задачи будет иметь следующий вид.

1. j 6 2	4. p 5	7. j 6 8	10. s
2. l 3	5. s	8. l 9	
3. j 4 2	6. r 7	9. p 10	

Определение. *Поглощением называется операция замены одной команды другой с исключением последней.*

В общем случае при поглощении команды с номером α командой с номером β необходимо выполнить следующие действия. Во-первых, всюду отсылку на α заменить отсылкой на β. Во-вторых, вычеркнуть команду с номером α. В-третьих, во всех командах образовавшегося списка числа $\alpha+1, \alpha+2, \alpha+3, ...$ (которые могут быть как номерами, так и отсылками) заменить соответственно числами $\alpha, \alpha+1, \alpha+2, ...$

После поглощения некоторыми командами других может быть получена наименьшая по числу инструкций программа, решающая указанную задачу с учетом ограничения.

1. j 6 2
2. l 3
3. j 4 2
4. p 5
5. s
6. r 1

Представим теперь решение описанной задачи без учета введенного ограничения: в момент начала выполнения программы каретка может находиться в любом месте относительно числа на ленте. Программа будет выглядеть следующим образом.

1. j 2 3
2. l 4
3. r 4
4. j 5 3
5. p 6
6. l 7
7. j 8 15
8. r 9
9. j 10 8
10. p 11
11. r 12
12. j 13 19
13. l 14
14. j 5 13
15. r 16
16. r 17
17. j 23 18
18. e 16
19. l 20
20. l 21
21. j 23 22
22. e 20
23. s

2.1.3 Анализ и синтез программ для машины Поста

Эффективным способом анализа программ для машины Поста является построения специализированной диаграммы Поста. На такой диаграмме команды изображаются кружками (называемыми также узлами), а переходы от одной команды к другой — стрелками (называемыми также дугами). Очевидно, что из узла, соответствующего команде передачи управления, выходит две дуги. Рядом с одной из дуг может стоять символ метки. Это позволят отличать нижнюю отсылку от верхней. Кроме того, из узла, соответствующего команде останова, не выходит ни одна дуга. Узел делится горизонтальной чертой на две части. В верхней изображается номер команды, а в нижней — символ команды.

На рис. 3 изображена диаграмма Поста для последней представленной программы.

Для лучшего понимания сути анализируемой программы целесообразным, как правило, оказывается разделение узлов на группы в зависимости от их назначения. Каждый блок в таком случае изображается в виде прямоугольника, содержащего уникальный номер, список включенных в него команд из диаграммы и текстовую метку, описывающую функциональное

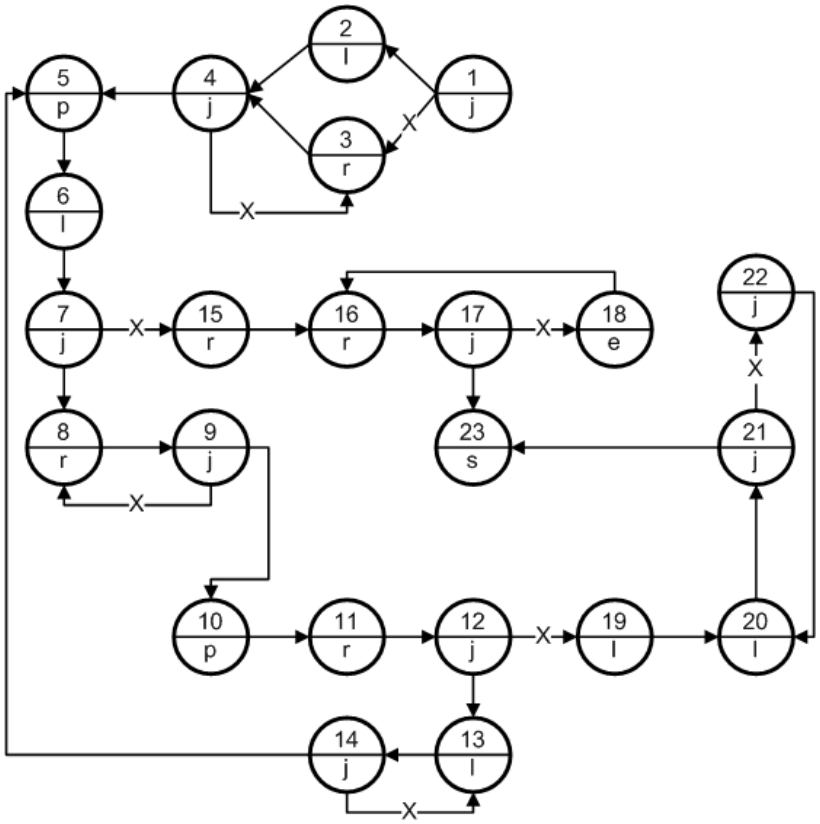

Рис. 3 — Пример диаграммы Поста

назначение блока. Стрелки используются аналогичным образом для соединения блоков. При этом переход из одного блока в другой существует тогда и только тогда, когда в состав первого блока входит узел из которого есть переход в узел, входящий во второй блок). Отметки на стрелках, как правило, не ставятся. Описанная структура получила название схемы Поста.

На рис. 4 изображена схема Поста, построенная для представленной выше диаграммы.

К сожалению, для синтеза программ машины Поста не существует универсального метода. Как и любой процесс построения алгоритма, данная деятельность является творческой и не может быть формализована.

2.2 Машина Тьюринга

2.2.1 Формальное определение машины Тьюринга

Одной из самых популярных алгоритмических систем является так называемая машина Тьюринга (МТ). Она была предложена в 1936 году одним из наиболее значимых исследователей в области computer science — англи-

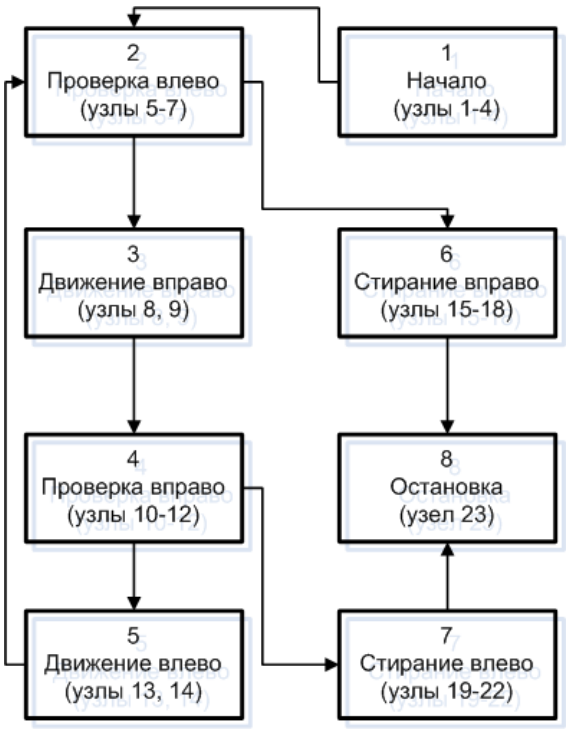

Рис. 4 — Пример схемы Поста

чанином Аланом Тьюрингом. Результаты, полученные им, соответствуют результатам Эмиля Поста (использовалась одинаковая концепция), однако носят более общий характер. Именно поэтому описываемая алгоритмическая система получила его имя (хотя нередко в литературе можно встретить упоминание о машина Тьюринга-Поста).

Алгоритмическая система МТ — это формальное понятие, введенное как уточнение интуитивного понятия алгоритма. С технической точки зрения (не забывая про абстракцию, свойственную МТ как и машине Поста) эта конструкция состоит из следующих составных элементов.

- Неограниченная в обе стороны лента, разделенная на ячейки.
- Управляющее устройство.
- Чувствительная головка для анализа содержимого ячеек и работы с ним.

Схематичное изображение машины приведено на рис. 5.

Определение. *МТ называется стандартной, если при сдвиге ленты может быть изменено содержимое обозреваемой ячейки.*

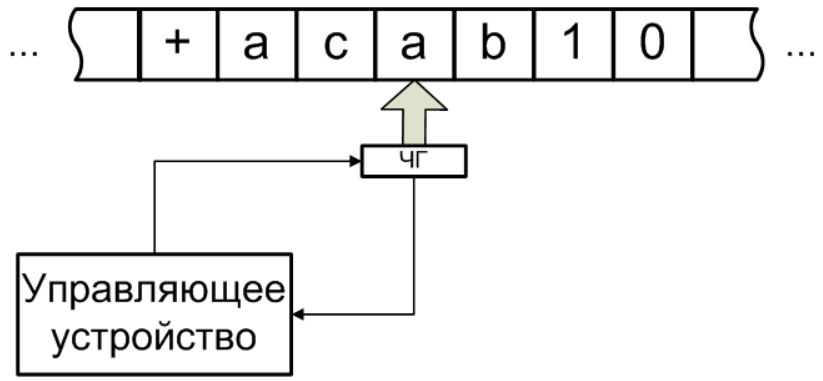

Рис. 5 — Схематичное изображение машины Тьюринга

Допустим, задан некоторый алфавит $A = \{s_0, s_1, s_2, ...\}$, где s_0 — пустой символ. Назовем A внешним алфавитом МТ. Допустим, кроме того, задан внутренний алфавит $Q = \{q_0, q_1, q_2, ..., \dot{q_m}\}$, описывающий внутренние состояния управляющего устройства. Здесь $\dot{q_i}$ — заключительное состояние. Назовем алфавит Q внутренним алфавитом машины Тьюринга. В общем случае алфавит Q может содержать более одного заключительного состояния.

Определение. *Любую совокупность из последовательности содержимого ячеек МТ и одного из внутренних состояний будем называть конфигурацией машины Тьюринга.*

$$... \; s_2 \; s_5 \; s_1 \; q_{10} \; s_0 \; s_3 \; s_3 \; s_8 \; ...$$

В указанной конфигурации машина находится в состоянии q_{10} напротив ячейки, содержащей символ s_0.

Если МТ в состоянии q_i воспринимает символ s_j, заменяет этот символ на s_k и переходит в следующее состояние q_l, возможно, осуществляя при этом сдвиг чувствительной головки на одну позицию влево или вправо, то будем говорить, что МТ выполняет команду, описываемую пятеркой следующего вида «$q_i \; s_j \rightarrow q_l \; s_k \; d$», где d — описание движения ленты (может принимать следующие значения: L — сдвиг влево, R — сдвиг вправо, S — отсутствие сдвига). Необходимо отметить, что МТ останавливается после выполнения команды, если q_l принадлежит множеству заключительных состояний.

Определение. *Конечная совокупность команд, заданная на одних и тех*

же внутренних и внешних алфавитах, называется программой машины Тьюринга. Программа МТ называется также «системой команд».

Таким образом, «задать МТ» — значит решить следующие задачи.

1. Задать внешний алфавит (с пустым символом).
2. Задать внутренний алфавит (с заключительными состояниями).
3. Задать программу работы МТ.
4. Задать начальную конфигурацию МТ.

Рассмотрим процесс работы МТ на следующем примере.

1. $A = \{a, b, c, d, s_0\}$
2. $Q = \{q_0, q_1, q_2, q_3, q_4, \dot{q_5}\}$
3. Программа МТ состоит из следующих команд.

   (a) $q_0\ a\ q_1\ a\ L$ (c) $q_2\ a\ \dot{q_5}\ d\ R$ (e) $q_1\ d\ q_2\ c\ R$ (g) $q_4\ b\ q_2\ c\ R$
   (b) $q_0\ b\ q_0\ b\ L$ (d) $q_0\ c\ q_0\ c\ L$ (f) $q_3\ a\ q_4\ d\ R$

4. Начальная конфигурация МТ: «$q_0\ b\ c\ a\ d\ c$»

Выполнение программы по шагам будет иметь следующий вид.

1. Выполнение команды (b) (новая конфигурация «$b\ q_0\ c\ a\ d\ c$»).
2. Выполнение команды (d) (новая конфигурация «$b\ c\ q_0\ a\ d\ c$»).
3. Выполнение команды (a) (новая конфигурация «$b\ c\ a\ q_1\ d\ c$»).
4. Выполнение команды (e) (новая конфигурация «$b\ c\ q_2\ a\ c\ c$»).
5. Выполнение команды (c) (заключительная конфигурация машины Тьюринга «$b\ \dot{q_5}\ c\ d\ c\ c$»).

Таким образом, заданная МТ перерабатывает входное слово *bcadc* в выходное слово *bcdcc*. Работа Тьюринга декларирует, что таким же образом могут быть реализованы любые, даже самые сложные, алгоритмы.

Тезис. *Любой процесс, который было бы естественно назвать эффективной процедурой (алгоритмом), может быть реализован алгоритмической системой машины Тьюринга.*

Данный тезис является главным результатом, полученным А. Тьюрингом. Он имеет чрезвычайно важное значение и используется для доказательства существования алгоритмов при решении самых различных классов задач. С математической точки зрения тезис недоказуем, так как понятие «любой процесс» не является строгой категорией. Но вся практическая деятельность подтверждает его высокую достоверность.

Теорема. *Любая функция, которая может быть вычислена с помощью машины Тьюринга, может быть также вычислена с помощью машины Поста.*

2.2.2 Пример использования машины Тьюринга

Рассмотрим следующую задачу. Пусть задано конечное слово, состоящее из нулей и единиц. Требуется построить МТ, которая позволяла бы определять, четно или нечетно число единиц в заданном слове. Допускается, чтобы в начальной конфигурации чувствительная головка располагалась напротив самого левого символа слова.

Поставленная задача может быть решена с помощью следующей МТ.

1. $A = \{0, 1, B\}$, где B — вспомогательный символ границы слова.
2. $Q = \{q_0, q_1, \dot{q_2}\}$
3. Программа МТ состоит из следующих команд.

   (a) $q_0\ 0\ q_0\ 0\ L$ (c) $q_0\ B\ \dot{q_2}\ 0\ S$ (e) $q_1\ 1\ q_0\ 0\ R$

   (b) $q_0\ 1\ q_1\ 0\ L$ (d) $q_1\ 0\ q_1\ 0\ L$ (f) $q_1\ B\ \dot{q_2}\ 1\ S$

4. Начальная конфигурация МТ: «$q_0\ \{0|1\}\ B$»

Анализ приведенных простых примеров показывает, что для решения поставленной задачи нужно вначале выбрать способ решения, который, в

общем случае, может быть не единственным. Внешний алфавит, кроме символов входного слова, может содержать несколько вспомогательных символов. Внутренний алфавит должен содержать столько символов, сколько различных простых операций необходимо выполнить над входным словом.

Рассмотрим более сложный пример. Пусть задано скобочное выражение, имеющее конечное множество правых и левых скобок. Требуется построить МТ, позволяющую проверить корректность расставленных скобок. Аналогично предыдущей задаче, допускается, чтобы в начальной конфигурации чувствительная головка располагалась напротив самого левого символа слова.

Поставленная задача может быть решена с помощью следующей МТ.

1. $A = \{0, 1, (,), B, X\}$, где B — вспомогательный символ границы слова (граничный символ), а X — символ замены.

2. $Q = \{q_0, q_1, q_2, \dot{q_3}\}$, где q_0 — состояния для поиска первой правой скобки, q_1 — состояние для поиска парной левой скобки, а q_2 — состояние проверки.

3. Программа МТ состоит из следующих команд.

(a) q_0 (q_0 (L	(e) q_1 (q_0 X L	(i) q_2 ($\dot{q_3}$ 0 S
(b) q_0) q_1 X R	(f) q_1) $\dot{q_3}$ 0 S	(j) q_2) $\dot{q_3}$ 0 S
(c) q_0 X q_0 X L	(g) q_1 X q_1 X R	(k) q_2 X q_2 X R
(d) q_0 B q_2 B R	(h) q_1 B $\dot{q_3}$ 0 S	(l) q_2 B $\dot{q_3}$ 1 S

4. Начальная конфигурация МТ: «B q_0 $\{(|)\}$ B»

В конце завершения выполнения программы следует проанализировать последний записанный на ленту символ. Если им окажется 0 — входное слово не является корректной скобочной последовательностью, если 1 — является.

Примечание. *Предлагаемая система команд избыточна (содержит элементы, которые могут быть исключены) и может быть упрощена.*

2.2.3 Универсальная машина Тьюринга

Можно заметить, что различные алгоритмы реализуются на различных машинах Тьюринга. По аналогии с ЭВМ можно назвать их специализированными машинами Тьюринга (спецЭВМ — спецМТ). Возникает вопрос: существует ли универсальная МТ, способная решать любые задачи, и, следовательно, выполнять работу любой наперед заданной МТ? Оказывается, существует. Эта проблема решается путем кодирования конфигураций и программы некоторой заданной МТ в символах внешнего алфавита универсальной МТ по следующим правилам.

1. Различные символы заменяются кодовыми группами.

2. Строки кодовых записей должны быть разбиты на кодовые группы.

3. Отдельные кодовые группы соответствуют операциям движения ленты МТ.

Например, пусть задана МТ с $A = \{s_1, s_2, ..., s_k\}$ и с $Q = \{q_1, q_2, ..., q_m\}$, а внешний алфавит универсальной МТ имеет вид $A' = \{0, 1\}$. Выберем следующий способ кодирования: в качестве кодовой группы возьмем выражения $(3 + k + m)$ слов вида 100...01 (нули в обрамлении единиц), где k — число внешних символов, а m — число состояний устройства управления.

Тогда кодирование может быть сведено к таблице с четным числом нулей (начиная с четырех) для внешнего алфавита и нечетных (начиная с пяти) — для внешнего. При этом символы движения ленты могут быть закодированы как 101 (L), 1001 (R) и 10001 (S).

В примере переработки слова *bcadc* символы внешнего алфавита были бы закодированы для универсальной машины как: a — 100001, b — 10000001, c — 1000000001, d — 100000000001. Далее работа осуществлялась бы аналогично за тем исключением, что потребовалось бы реализовать цикл поиска подходящей команды на ленте универсальной машины.

Современные ЭВМ в основном строятся как универсальные, но в отличие от теоретической МТ физически реализованные машины не имеют неограниченной памяти.

Универсальные МТ могут иметь различные структуры. Для сопоставления этих структур между собой К. Шеннон предложил специальную меру — произведение числа символов на число внутренних состояний. Эта

мера определяет «сложность» универсальной МТ. Известны следующие результаты — теоремы о существовании универсальной МТ.

- Ватанабе — пять символов на шесть состояний.
- Минский — четыре символа на семь состояний.
- Триттер — четыре символа на шесть состояний.

2.2.4 Многоленточная машина Тьюринга

Существуют и активно используются на практике различные специализированные модификации МТ, наиболее популярной из которых является многоленточная МТ. Она состоит из конечного числа лент, в каждой ячейке которых может быть записан один из символов внешнего алфавита A. Устройство управления в каждый момент времени может находится в одном из состояний Q. Схематичное изображение данной конструкции приведено на рис. 6.

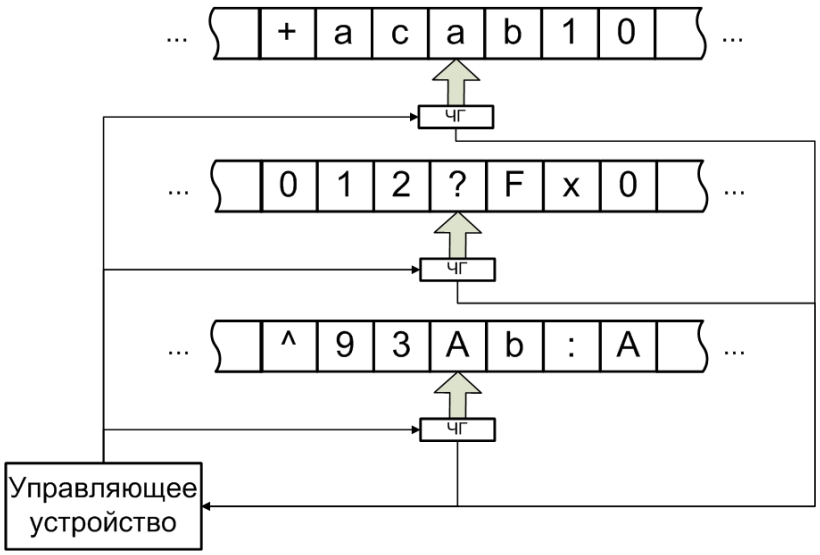

Рис. 6 — Схематичное изображение многоленточной машины Тьюринга

Предположим, что МТ обозревает одну ячейку каждой ленты. Конфигурация МТ считается заданной, если известно состояние управляющего устройства q_i, состояния всех ячеек всех лент и отмечены обозреваемые ячейки. Для L-ленточной МТ, где L — число лент, в j-ый момент времени можно записать.

$$s_{j,1,1}\ s_{j,1,2}\ \dots\ s_{j,1,p_1-1}\ q_i\ s_{j,1,p_1}\ \dots\ s_{i,1,r_1}$$

...

$$s_{j,L,1}\ s_{j,L,2}\ \dots\ s_{j,L,p_L-1}\ q_i\ s_{j,L,p_L}\ \dots\ s_{i,L,r_L}$$

В указанном описании конфигурации первый нижний индекс символа внутреннего алфавита — момент времени, второй — номер ленты, третий — номер символа на ленте.

Если МТ, взаимодействующая с L лентами, в состоянии q_i воспринимает символ s_{j_1} на первой лента, s_{j_2} на второй ленте и так далее, заменяет эти символы на s_{k_1}, s_{k_2} и так далее соответственно и переходит в следующее состояние q_l, возможно, осуществляя при этом сдвиг чувствительных головок на одну позицию влево или вправо, то будем говорить, что МТ выполняет команду, описываемую следующим образом «$q_i\ s_{j_1}\ s_{j_2}\ \dots\ s_{j_L} \to q_l\ s_{k_1}\ s_{k_2}\ \dots\ s_{k_L}\ d_{j_1}\ d_{j_2}\ \dots\ d_{j_L}$», где d — описание движения ленты (может принимать следующие значения: L — сдвиг влево, R — сдвиг вправо, S — отсутствие сдвига). Необходимо отметить, что МТ останавливается после выполнения команды, если q_l принадлежит множеству заключительных состояний.

2.2.5 Функции, вычислимые по Тьюрингу

Как отмечает американский математик М. Минский: «Всякий раз при анализе результата на ленте МТ мы сознаем, что он зависит от входного слова, то есть является «функцией» от входного слова. Следовательно, задавая МТ, мы задаем некоторую функцию от аргумента. Если входные и выходные слова не являются числами, то в принципе мы всегда можем пронумеровать слова в определенном порядке.» Рассмотрим для примера метод, предложенный К. Геделем.

Дано некоторое число $n = 2^{a_1} \cdot 3^{a_2} \cdot 5^{a_3} \cdot \dots \cdot p_m^{a_m}$, где $2, 3, 5, \dots, p_m$ — простые целые числа. Согласно теореме о единственности разложения любого числа n на простые множители каждому значению n соответствует набор $a_1, a_2, \dots, a_m$ и наоборот. Так $n = 60 = 2^2 \cdot 3^1 \cdot 5^1 (a_1 = 2, a_2 = 1, a_3 = 1)$.

Используя метод Геделя, логично нумеровать любые упорядоченные последовательности из m чисел. Данный принцип получил название принципа геделизации.

Существует несколько равнозначных определений функции. Функция — это правило, устанавливающее как по данному числу (аргументу) вычислить другое число (значение функции). Функция есть однозначное соответствие между словами одного множества (аргументами) и словами другого множества (значениями).

Одним из возможных вариантов определения функции простого вида является отображение, чьи аргументы и значения — целые неотрицательные числа $(0, 1, 2, ...)$. Тогда справедливо следующее определение.

Определение. *Функция $f(x)$ вычислима по Тьюрингу, если ее значения могут быть вычислены некоторой МТ, на ленте которой первоначально было задано стандартное представление аргумента x. Значением $f(x)$ будет запись на ленте после останова МТ.*

Выяснилось, что существует МТ для любой функции, вычислимой по Тьюрингу, если надлежащим образом подобрать представление аргументов (это и есть стандартное представление x).

2.2.6 Композиция машин Тьюринга

Совершенно очевидно, что с математической точки зрения МТ — определенный алгоритм. Известно, что существуют операции над алгоритмами, позволяющие получать более сложные алгоритмы. Рассмотрим некоторые из этих операций.

Определение. *Произведением $MT1$ на $MT2$ будем называть MT с общим внешним алфавитом $A = a_0, a_1, ..., a_m$ и суммарным внутренним алфавитом $Q = \dot{q}_0, q_1, ..., q_n, ..., q_{n+k}$ и программой задаваемой следующим образом. Во всех командах $MT1$, содержащих переход в заключительное состояние $\dot{q}_0$, производится замена его на символ q_{n+1}. Во всех команда $MT2$ наоборот: переход в $\dot{q}_0$ остается неизменным, но все остальные символы $q_j (j = 1, 2, ..., k)$ заменяются символом q_{n+j}. Совокупность команд $MT1$ и $MT2$ и будет представлять программу произведения $MT = MT1 \cdot MT2$. Другими словами работа* МТ *эквивалента работе последовательно включенных $MT1$ и $MT2$.*

Рассмотрим произведение машин Тьюринга на примере. Пусть имеется МТ1, позволяющая отыскивать ближайшую слева единицу после группы нулей. Внешний алфавит машины $A = \{0, 1\}$, внутренний — $Q =$

$\{\dot{q_0}, q_1, q_2\}$. Начальное состояние при работе машины — q_1. Программа МТ1 будет состоять из следующих команд.

- $q_1\ 0\ q_2\ 0\ R$
- $q_1\ 1\ q_1\ 1\ R$
- $q_2\ 0\ q_2\ 0\ R$
- $q_2\ 1\ \dot{q_0}\ 1\ S$

Пусть имеется МТ2, позволяющая стирать все единицы слева, если они есть, до ближайшего нуля. Внешний алфавит машины $A = \{0, 1\}$, внутренний — $Q = \{\dot{q_0}, q_1\}$. Начальное состояние при работе машины — q_1. Программа МТ2 будет состоять из следующих команд.

- $q_1\ 0\ \dot{q_0}\ 0\ S$
- $q_1\ 1\ q_1\ 0\ R$

Тогда произведение МТ1 и МТ2 позволит отыскивать ближайшую группу единиц и стирать их. Внешний алфавит машины $A = \{0, 1\}$, внутренний — $Q = \{\dot{q_0}, q_1, q_2, q_3\}$. Начальное состояние при работе машины — q_1. Программа будет состоять из следующих команд.

- $q_1\ 0\ q_2\ 0\ R$
- $q_3\ 0\ \dot{q_0}\ 0\ S$
- $q_2\ 1\ q_3\ 1\ S$
- $q_2\ 0\ q_2\ 0\ R$
- $q_1\ 1\ q_1\ 1\ R$
- $q_3\ 1\ q_3\ 0\ R$

Определение. *Возведением в степень машины МТ назовем последовательное произведение МТ на саму себя n раз.*

Если при этом МТ имеет несколько заключительных состояния, то умножение определяется аналогично, с обязательным указанием того, какое из заключительных состояний предыдущего сомножителя отождествляется с начальным состоянием последующего.

Например, если МТ1 имеет два заключительных состояния, то при произведении МТ1 на МТ2 необходимо указать из какого конкретно (или из всех) заключительного состояния необходимо переходить к выполнению программы МТ2.

Определение. *Операцией итерации называется отождествление одного из конечных состояния МТ с начальным.*

Следует заметить, что при этом МТ должна иметь более одного заключительного состояния. В противном случае будет получена МТ без возможности останова.

Если объектом итерации является МТ, которая сама является результатом операций умножения или итерации другой МТ, тогда соответствующее

число точек ставят над теми машинами, чьи заключительные и начальные состояния отождествляются.

$$MT = M\dot{T}1 \cdot M\ddot{T}2 \begin{cases} (1) & M\dot{T}3 \\ (2) & \mathrm{MT4} \cdot M\ddot{T}5 \\ (3) & MT6 \end{cases}$$

Данная запись означает, что отождествляются заключительное состояние MT3 и начальное состояния MT1, а также заключительное состояния $\mathrm{MT4} \cdot \mathrm{M\dot{T}5}$ и начальное состояние MT2.

Рассмотрим данную операцию на примере. Пусть задана машин MT, описываемая как

$$MT = M\dot{T}4 \cdot MT5 \begin{cases} (1) & MT3 \\ (2) & M\dot{T}2 \end{cases}$$

MT3 после группы нулей отыскивает справа группу единиц и останавливается в ее конце. MT4 идет влево и останавливается через две ячейки от группы единиц. MT5, посредством двух заключительных состояний, распознает напротив ячейки с каким содержимым остановилась головка (ноль или единица) и затем переходит к одному из двух продолжений.

Итоговая MT в состоянии q_1 стирает все единицы слева от начального положения до того места, где встретятся две пустые ячейки подряд. Затем лента возвращается назад и останавливается у крайней правой отмеченной ячейки группы единиц, от которой MT начинала работать. MT повторяет стирание до тех пор, пока левее некоторой группы единиц не появляются две пустые ячейки.

Системы команд машин приведены в табличной форме на рис. 7.

MT2	**0**	**1**
q1	q0,0,S	q1,0,R

MT3	**0**	**1**
q1	q2,0,L	q1,1,L
q2	q2,0,L	q3,1,L
q3	q0,0,L	q3,1,L

MT4	**0**	**1**
q1	q0,0,R	q1,1,R

MT5	**0**	**1**
q1	q0',0,S	q0",1,S

MT	**0**	**1**
q1	q2,0,R	q1,1,R
q2	q3,0,S	q4,1,S
q3	q5,0,L	q3,1,L
q4	q1,0,S	q4,0,R
q5	q5,0,L	q6,1,L
q6	q0,0,L	q6,1,L

Рис. 7 — Список команд машин MT2 — MT5 и MT

2.3 Машина с бесконечными регистрами

2.3.1 Формальное определение машины с бесконечными регистрами

Около 80 лет назад появилось понятие вычислимой функции. Это послужило основанием для создания новой ветви математических исследований в области теории машинных вычислений, в философии, во многих областях обработки информации.

Начальной целью теории вычислимости было придать научную строгость интуитивной идее вычислимой функции, чтобы затем приступить к математическому исследованию данного понятия.

В рамках среднего образования мы знакомы с операциями сложения и умножения чисел. Никто в явном виде не говорит о том, что для каждой пары чисел есть конкретные сумма и произведение, но показывают путь их нахождения — алгоритм(вычислительно эффективную процедуру).

Если алгоритм используется для вычисления значений числовой функции, то говорят, что эта функция «эффективно вычислима». Ясно, что любой компьютер может реализовывать вычислительные алгоритмы, но реальный компьютер имеет ограничение как по разрядности представления чисел, так и по объему памяти. Поэтому с математической точки зрения удобнее использовать идеализированный компьютер.

Рассмотрим один из вариантов такого компьютера, который получил название «машина с бесконечными регистрами» (МБР), лишенный указанных ограничений. Все программы будут конечными, а входы и выходы ограничим натуральными числами (в совокупности с принципом геделизации это будет означать, что информация может быть любой).

МБР включает в себя бесконечное число регистров $R_1, R_2, ...$, каждый из которых может содержать любое натуральное число. $R_n := r_n$, где n — номер регистра. Машина может изменять содержимое регистров по некоторой команде, которая является чрезвычайно простой операцией над числами. В МБР существует четыре вида команд.

- Команда обнуления $Z(n)$. Эта команда заменяет содержимое регистра $R_n := 0$ и не затрагивает остальные регистры.

- Команда прибавления единицы $S(n)$. Содержимое некоторого регистра увеличивается на единицу, т.е. $R_n := r_n + 1$.

- Команда переадресации $T(m, n)$. При выполнении этой команды содержимое регистра R_n заменяется содержимым регистра R_m, т.е. $R_n := r_m$.

- Команда условного перехода $J(m, n, q)$. Реакция машины на команду заключается в следующем: сравнивается содержимое регистров R_n и R_m; если $r_n = r_m$, то машина переходит к выполнению q-ой команды, иначе к следующей по очереди.

Чтобы производить вычисления, нужно задать программу P и так называемую начальную конфигурацию (заполнение регистров $R_1, R_2, ...$). Допустим $P = I_1, I_2, ..., I_s$. МБР выполняет инструкции по порядку $I_1, I_2, ...$ и так далее, пока не встретит команду $I_j = J(m, n, q)$. После этого, в зависимости от содержимого регистров R_n и R_m, машина выполняет либо очередную, либо q-ю команду. Вычисления идут до тех пор, пока не кончится список команд.

2.3.2 Пример использования машины с бесконечными регистрами

Рассмотрим подробнее процесс работы МБР на примерах. Пусть дана программа P.

- $I_1 - J(1, 2, 6)$
- $I_2 - S(2)$
- $I_3 - S(3)$
- $I_4 - J(1, 2, 6)$
- $I_5 - J(1, 1, 2)$
- $I_6 - T(3, 1)$

Начальная конфигурация $\{9, 7, 0\}$. В ходе выполнения программы регистры будут изменяться следующим образом $\{9, 7, 0\}(1) \to \{9, 7, 0\}(2) \to \{9, 8, 0\}(3) \to \{9, 8, 1\}(4) \to \{9, 8, 1\}(5) \to \{9, 8, 1\}(2) \to \{9, 9, 1\}(3) \to \{9, 9, 2\}(4) \to \{9, 9, 2\}(6) \to \{2, 9, 2\}$.

Ход вычислений на машине указанного типа удобно описывать с помощью схемы алгоритма (диаграммы переходов), пример которой изображен на рис. 8.

В общем случае будем использовать обозначения:

1. $P(a_1, a_2, ..., a_s)$ — вычисление по программе P с начальной конфигурацией $(a_1, a_2, ..., a_s)$;

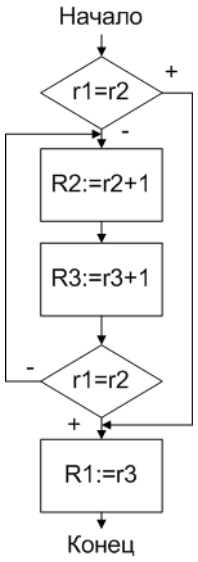

Рис. 8 — Схема алгоритма МБР

2. $P(a_1, a_2, ..., a_s) \downarrow$ — вычисление в определенный момент останавливается (сходится);

3. $P(a_1, a_2, ..., a_s) \uparrow$ — вычисление никогда не останавливается (расходится);

4. $P(a_1, a_2, ..., a_s) = P(a_1, a_2, ..., a_s, 0, 0, ...)$.

Если вычисление расходящееся, то есть никогда не кончается, это означает, что значение функции вычислимо не на всех наборах аргументов $(a_1, a_2, ..., a_s)$; функция в этом случае является частичной.

Определение. *Пусть F — частичная функция на множестве целых неотрицательных $N = \{0, 1, 2, 3, ...\}$. Если задана программа P, начальная конфигурация $(a_1, a_2, ..., a_i, ..., a_s)$, и получается некоторый результат b, где $a_i, b \in N$, то*

1. *Вычисление $P(a_1, a_2, ..., a_s)$ сходится к b, когда $P(a_1, a_2, ..., a_s) \downarrow$ и в заключительной конфигурации в регистре R_1 находится b ($R_1 := b$);*

2. *Программа P вычисляет F, если для всех $a_1, a_2, ..., a_s, b$ имеет место $P(a_1, a_2, ..., a_s) \downarrow b$, тогда и только тогда, когда $(a_1, a_2, ..., a_s) \in DomF$ и $F(a_1, a_2, ..., a_s) = b$;*

3. *Функция F является МБР-вычислимой, если есть программа, которая МБР-вычисляет F.*

Класс всех МБР-вычислимых функций обозначим как ϕ, класс n-местных МБР-вычислимых функций через ϕ_n.

Рассмотрим еще один пример. Пусть необходимо вычислить значение функции $f(x, y) = x + y$, то есть прибавить к x единицу y раз. Для начальной конфигурации $(x, y, 0)$ это может быть сделано с помощью следующей программы.

- $I_1 - J(3, 2, 5)$
- $I_2 - S(1)$
- $I_3 - S(3)$
- $I_4 - J(1, 1, 1)$

Рассмотрим последний пример. Пусть необходимо вычислить значение функции

$$f(x) = \begin{cases} x - 1 & , x > 0; \\ 0 & , x = 0. \end{cases}$$

Пусть начальная конфигурация имеет вид $(x, 0, ...)$. Проверим вначале условие $x = 0$? Если да, то нужно остановиться, иначе — изменить содержимое двух счетчиков, содержащих k и $k + 1$, начиная с $k = 0$ (конфигурация $(x, k, k + 1, 0, ...)$). Затем проверим условие $x = k + 1$? Если да, то результат вычисления k, иначе — увеличим содержимое счетчиков на единицу. Выполним проверку заново. Схема предлагаемого алгоритма представлена на рис. 9.

- $I_1 - J(1, 4, 9)$
- $I_2 - S(3)$
- $I_3 - J(1, 3, 7)$
- $I_4 - S(2)$
- $I_5 - S(3)$
- $I_6 - J(1, 1, 3)$
- $I_7 - T(2, 1)$

Таким образом, функция $f(x) = x - 1$ вычислима.

2.3.3 Порождение вычислимых функций

Отметим, что некоторые очень простые функции вычислимы; это очевидно и не требует доказательства. Используя их, можно строить более сложные вычислимые функции.

Лемма. *Следующие функции вычислимы:*

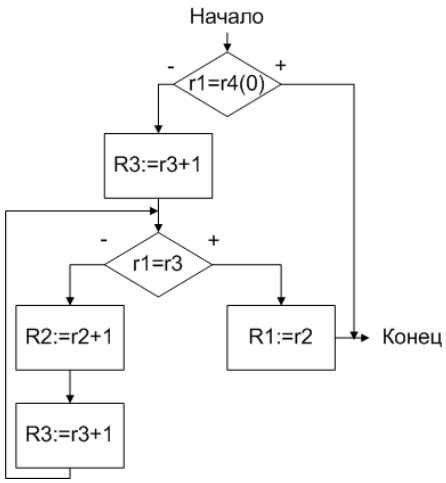

Рис. 9 — Схема алгоритма МБР

1. *Нуль-функция* $0(x) = 0$ *для всех* $x \in N$;

2. *Функция следования* $x + 1$;

3. *Для всех* $n \geq 1$ *и* $1 \leq i \leq n$ *функция проекции определена как* $V_i^n(x_1, ..., x_n) = x_i$.

Вычислимость данных функций на МБР не требует доказательства, т.к. они соответствуют арифметическим командам обнуления ($Z(1)$), прибавления единицы ($S(1)$) и переадресации ($T(i, 1)$).

Для построения более сложных вычислительных функций в теории МБР используются некоторые специальные операции.

Соединение программ. Предположим, имеется две программы P и Q. Требуется построить более сложную программу $P \cdot Q$ так, что $P = I_1, I_2, ..., I_s$. Вычисление P завершается, когда следующая команда I_v, причем $v > s$, т.е. $v = s + 1$. Но это происходит лишь тогда, когда I_v — арифметическая. Исключение возможно только если I_v — условный переход. В связи с этим, введем следующее определение.

Определение. *Программа* $P = I_1, I_2, ..., I_s$ *имеет «стандартный вид», если для всякой команды условного перехода* $J(m, n, q)$ *в* P *имеем* $q \leq s+1$.

Определение. *Даны* P *и* Q *— программы стандартного вида длины* s *и* t *соответственно. Конкатенацией (соединением)* P *и* Q *называется программа* $P \cdot Q = I_1, ..., I_s, I_{s+1}, ..., I_{s+t}$, *где каждый условный переход из программы* Q $J(m, n, q)$ *заменен на* $J(m, n, s + q)$.

Подстановка программ. Распространенным способом построения новых сложных функций является подстановка значений одних функций вместо аргументов других. При этом вновь полученные функции также являются вычислимыми.

Теорема. *Пусть* $f(y_1, ..., y_k), g_1(x_1, ..., x_n), g_2(x_1, ..., x_n), ..., g_k(x_1, ..., x_n)$ — *вычислимые функции. Тогда функция* $h(x_1, ..., x_n) = f(g_1, ..., g_k)$, *также вычислима.*

Так, например, функция $f(x_1, x_2, x_3) = x_1 + x_2 + x_3$ вычислима. Это следует из того факта, что функция $x + y$ вычислима, а f представима в аналогичном виде, если $x = x_1 + x_2$, а $y = x_3$.

Рекурсия. Рекурсия — это способ задания функции путем определения каждого из ее значений в терминах ранее определенных значений для младших аргументов (и, возможно, использования других уже определенных функций).

Предположим, что $f(x_1, ..., x_n)$ и $g(x_1, ..., x_n, y, z)$ есть функции (не обязательно вычислимые).

Тогда определим некоторую новую функцию $h(x_1, ..., x_n, y)$.

$$h(x_1, ..., x_n, y) = \begin{cases} f(x_1, ..., x_n), & y = 0; \\ g_1(x_1, ..., x_n, y-1, h(x_1, ..., x_n, y-1)), & y > 0. \end{cases}$$

Чтобы найти значение $h(x_1, ..., x_n, 3)$, вначале найдем $h(x_1, ..., x_n, 0)$, затем по найденному значению найдем $h(x_1, ..., x_n, 1)$, $h(x_1, ..., x_n, 2)$ и так далее.

Теорема. *Пусть* $f(x_1, ..., x_n)$ *и* $g(x_1, ..., x_n, y, z)$ — *вычислимые функции, тогда функция* $h(x_1, ..., x_n, y)$, *полученная из* f *и* g *с помощью рекурсии, вычислима.*

Минимизация. Существует также специализированная операция, порождающая новые вычислимые функции — «неограниченная минимизация» или просто минимизация.

Пусть $f(x_1, ..., x_n, y)$ — функция, определим функцию $g(x_1, ..., x_n)$, положив, что $g(x_1, ..., x_n) = y_{min}$ из $f(x_1, ..., x_n, y) = 0$.

Отметим два обстоятельства.

- Во-первых, для некоторых наборов $x_1, ..., x_n$ может не найтись y, для которого $f(x_1, ..., x_n, y) = 0$.

- Во-вторых, предположим f вычислима и рассмотрим алгоритм вычисления $g(x_1, ..., x_n)$: «Вычисляем $f(x_1, ..., x_n, 0)$, $f(x_1, ..., x_n, 1)$, ... до тех пор, пока не найдется y для $f(x_1, ..., x_n, y) = 0$». Очевидно, что эта процедура может не завершиться.

Теорема. *Пусть $f(x_1, ..., x_n, y)$ — вычислимая функция, тогда вычислима функция $g(x_1, ..., x_n) = \mu y(f(x_1, ..., x_n, y) = 0)$.*

Используя операцию минимизации совместно с операциями подстановки и рекурсии, можно построить большее число функций, чем только при помощи последних двух.

Как показали теоретические исследования все новые функции, какими бы они не были, относятся к МБР-вычислимым. Другими словами, класс ϕ замкнут относительно выше указанных операций.

2.3.4 Пример доказательства вычислимости функции

Пусть имеется функция $\psi(x, y)$, определяемая системой уравнений, содержащих двойную рекурсию, которая значительно сильнее, чем одинарная. Требуется показать ее вычислимость.

$$\begin{cases} \psi(0, y) = y + 1; \\ \psi(x + 1, 0) = \psi(x, 1); \\ \psi(x + 1, y + 1) = \psi(x, \psi(x + 1, y)). \end{cases}$$

Заметим, что каждое значение $\psi(x, y)$ определено в терминах «более ранних» значений $\psi(x_1, y_1)$, где $x_1 < x$ и $y_1 < y$. Оказывается $\psi(x, y)$ можно получить, использовав всего несколько таких «более ранних значений».

Строго математически доказать вычислимость $\psi(x, y)$ трудно, поэтому коснемся идеи возможного направления — метода подходящего множества S.

Допустим $(x, y, z) \in S$ и соблюдаются требования:

- $z = \psi(x, y)$,

- S содержит все ранее вычислимые тройки $(x_1, y_1, \psi(x_1, y_1))$, необходимые для вычисления $\psi(x, y)$.

Определение. *Конечное множество троек S назовем подходящим, если выполняется:*

- *если* $(0, y, z) \in S$, *то* $z = y + 1$;
- *если* $(x + 1, 0, z) \in S$, *то* $(x, 1, z) \in S$;
- *если* $(x+1, y+1, z) \in S$, *то существует* u, *такое, что* $(x+1, y, u) \in S$ *и* $(x, u, z) \in S$.

Часть функций является не всюду определенными. Если $\alpha(x)$ и $\beta(x)$ — выражения, где $x = (x_1, ..., x_n)$, тогда запись $\alpha(x) \simeq \beta(x)$ означает, что для каждого x выражения $\alpha(x)$ и $\beta(x)$ либо определены и равны, либо оба неопределенны.

Доказательство вычислимости рассматриваемой функции может базироваться на нахождении алгоритма построения множества S.

2.3.5 Формальное определение параллельной машины с бесконечными регистрами

Под параллельной машиной с бесконечными регистрами (пМБР) понимается расширение классической машины, способное одновременно выполнять произвольное число программ $P_1, P_2, ...$ Доступ к регистрам при этом возможен из каждой программы, то есть регистры являются общей (разделяемой) памятью.

Множество команд пМБР дополнено пятью элементами.

- Команда захвата ресурса $G(n)$. Эта команда предписывает заблокировать регистр R_n и разрешить доступ к нему только из программы, выполнившей блокировку. Другие обратившиеся к такому регистру программы входят в состояние ожидания и пребывают в нем до того момента, пока регистр не будет освобожден.
- Команда освобождения ресурса $P(n)$. Эта команда предписывает разблокировать регистр R_n и разрешить доступ к нему из всех программ. Данная команда дополняет предыдущую. Очевидно, что совместное их использование позволяет гибко управлять доступом к регистрам.

- Команда запуска программы $s(k, n)$. Предписывает начать выполнение программы с номером k, сопоставив ему регистр с номером R_n. Сопоставленный регистр используется как индикатор состояния выполнения программы. В начале выполнения потока команд регистр обнуляется ($r_n := 0$), а в момент завершения потока в регистр заносится единица ($r_n := 1$), свидетельствующая об окончании выполнения программы.

- Команда ввода $I(n)$ предписывает прочитать из проассоциированного с машиной устройства ввода значение и занести его в регистр R_n. Данная команда не является обязательной с точки зрения структуры машины и даже в определенной мере противоречит абстрактным основам МБР, однако ее использование значительно облегчает процесс взаимодействия оператора с машиной.

- Команда вывода $O(n)$ предписывает перенаправить на проассоциированное с машиной устройство вывода значение регистра R_n. Данная команда дополняет предыдущую.

Другим важным усовершенствованием является поддержка косвенной регистровой адресации. В командах $Z(n)$, $S(n)$, $T(m, n)$, $J(m, n, q)$, $I(n)$, $O(n)$, $G(n)$, $P(n)$ и $s(k, n)$ вместо непосредственного указания номера регистра n или m разрешается использовать передачу значения через вспомогательный регистр. Например, команда $S(n)$ с использованием косвенной регистровой адресации примет вид $S(R(n))$, что будет означать «увеличить значение регистра с номером, соответствующим значению регистра с номером n, на единицу».

2.3.6 Пример использования параллельной машины с бесконечными регистрами

Для пояснения вышеизложенного рассмотрим пример. Пусть требуется вычислить функцию $f(a, b, c, d) = a \cdot b + c \cdot d$.

Очевидно, что выражения $a \cdot b$ и $c \cdot d$ не зависят друг от друга, поэтому могут быть вычислены параллельно. Тогда значение функции может быть найдено с помощью следующих программ P_1, P_2, P_3. Начальная конфигурация машины $(0, 0, 0, 0, 0, 0, a, b, 0, 0, c, d, 0, ...)$.

Основная программа P_1.

- $I_1 - I(7)$
- $I_2 - I(8)$
- $I_3 - I(11)$
- $I_4 - I(12)$
- $I_5 - s(2,1)$
- $I_6 - s(3,2)$
- $I_7 - S(3)$
- $I_8 - J(1,3,10)$
- $I_9 - J(3,3,8)$
- $I_{10} - J(2,3,12)$
- $I_{11} - J(3,3,10)$
- $I_{12} - J(5,6,16)$
- $I_{13} - S(4)$
- $I_{14} - S(5)$
- $I_{15} - J(1,1,12)$
- $I_{16} - O(4)$

Вспомогательная программа P_2 для вычисления $a \cdot b$.

- $I_1 - J(7,9,9)$
- $I_2 - Z(10)$
- $I_3 - J(8,10,7)$
- $I_4 - S(4)$
- $I_5 - S(10)$
- $I_6 - J(3,3,3)$
- $I_7 - S(9)$
- $I_8 - J(3,3,1)$

Вспомогательная программа P_3 для вычисления $c \cdot d$.

- $I_1 - J(11,13,9)$
- $I_2 - Z(14)$
- $I_3 - J(12,14,7)$
- $I_4 - S(5)$
- $I_5 - S(14)$
- $I_6 - J(3,3,3)$
- $I_7 - S(13)$
- $I_8 - J(3,3,1)$

2.4 Нормальные алгоритмы

2.4.1 Определение алгоритмической системы

Всякий общий способ задания алгоритма принято называть алгоритмической системой. Обычно алгоритмические системы включают в себя объекты двоякой природы — элементарные операторы и элементарные распознаватели.

Определение. *Элементарные операторы — это достаточно простые (то есть просто задаваемые) алфавитные операторы, с помощью последовательного выполнения которых могут быть реализованы любые алгоритмы в данной алгоритмической системе.*

Определение. *Элементарные распознаватели — это объекты для распознавания наличия тех или иных свойств обрабатываемой информации*

и изменения последовательности, в зависимости от результата распознавания, элементарных операторов.

Практика показала, что для указания набора элементарных операторов и порядка их следования при реализации алгоритма удобно использовать направленные графы особого рода — граф-схемы алгоритмов (ГСА), предложенные Л.А. Калужниным, и представляющие собой конечное множество соединенных между собой стрелками вершин — узлов ГСА. Каждому узлу, кроме входного и выходного, сопоставляется какой-либо элементарный оператор или распознаватель. Из каждого узла выходит либо одна стрелка (операторный узел), либо две (распознавательный узел), а число входных стрелок неограниченно.

Алгоритм, представленный ГСА, функционирует следующим образом. Входное слово поступает на первый узел, и продвигается по направлению, указанному стрелкой, преобразуясь при прохождении операторных узлов. Если слово попадает в распознавательный узел, то выполняется проверка сопоставленного этому узлу условия. При выполнении условия слово продолжает движение по стрелке со знаком «+», иначе — по стрелке со знаком «-» (само слово при этом не изменяется).

Если слово p подается на входной узел, проходит через узлы ГСА, попадает через конечное число шагов в выходной узел, то считается, что алгоритм применим к слову p (то есть p входит в область определения данного алгоритма).

В 1953 г. академик А.А. Марков предложил и исследовал функционирующую схожим образом алгоритмическую систему, которая получила название «нормальные алгоритмы».

В нормальных алгоритмах используют только один тип элементарного оператора — подстановка — и один тип элементарного распознавателя — вхождение.

Введем ряд определений.

Определение. *Если p и q — два слова алфавита A, то говорят, что слово q входит в слово p, если p может быть представлено $p = p_1 \cdot q \cdot p_2$, где p_1 и p_2 — некоторые слова (возможно, пустые). При этом вхождение слова q в слово p называется первым, если слово p_1 имеет наименее возможную длину среди всех допустимых представлений слова p.*

Определение. *Соединением слов p и q называется слово $R = p \cdot q$, получаемое путем приписывания (конкатенации) слова q справа к слову p.*

Определение. *λ — пустое слово в алфавите A: $\lambda \cdot p = p \cdot \lambda = p$.*

Определение. *Говорят, что слово p начинается словом q, если есть слово R, такое, что $p = q \cdot R$.*

Определение. *Слово p равно слову q, если они содержат одинаковые буквы алфавита A в одинаковом порядке.*

Определение. *Если $R = p \cdot q \cdot s$ и V — некоторые слова в A, то образование нового слова $p \cdot V \cdot s$ называется подстановкой слова V вместо слова q везде, где оно встречается, а новое слово $X = p \cdot V \cdot s$ — результат подстановки.*

Оператор подстановки обычно задаётся в виде двух слов, соединённых стрелкой $(p_1 \to p_2)$. Работа оператора состоит в том, что он осуществляет подстановку слова p_2 вместо первого вхождения слова p_1 в любое слово p.

Определение. *Алгоритмы, заданные ГСА, составленные исключительно из распознавателей вхождения слов и операторов подстановки, называются обобщенными нормальными алгоритмами. При этом предполагают, что каждому оператору подстановки вида $(p_1 \to p_2)$ подсоединяется только одна стрелка: стрелка «+» от распознавателя вхождения p_1.*

Допустим, даны $A = \{x, y\}$ и $P = xyxxyyxxx$, а также подстановки $\{xy \to e, yx \to xy\}$ (e — пустое слово). Указанный алгоритм расставляет все x левее всех y, а затем вычеркивает пары xy. В конце работы остаются либо e, либо xxx, либо yyy. ГСА данного алгоритма изображена на рис. 10.

Определение. *Нормальными алгоритмами (НА) называют такие обобщенные нормальные алгоритмы, ГСА которых имеют некоторый специальный вид, когда всякий оператор $(p_1 \to p_2)$ входит в паре с распознавателем.*

Если в предыдущем примере на ГСА объединить в пары распознаватель и преобразователь, тогда из вновь полученного узла всегда будут выходить

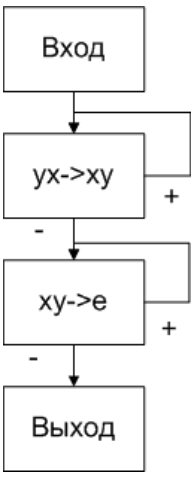

Рис. 10 — ГСА алгоритма

по две стрелки: с отметкой «+», если оператор применим к слову, и с отметкой «-», если неприменим (то есть левая часть подстановки отсутствует в слове).

Объединенная ГСА, в случае нормальных алгоритмов, должна удовлетворять ряду следующих условий.

- Все объединенные узлы ГСА упорядочиваются путем нумерации от 1 до N, стрелка со знаком «-» i-го узла присоединяется к $(i+1)$-ому узлу, а N-ого узла — к выходному;
- положительные выходы подсоединяются либо к первому, либо к выходному узлу ГСА;
- входной узел подсоединяется к первому объединенному (распознавательному и, одновременно, преобразовательному) узлу.

Очевидно, что ГСА с объединенными узлами на рис. 10 не отвечает требованиям НА, так как второе условие не выполняется.

Традиционно, НА задаются не ГСА, а упорядоченными системами подстановок всех операторов (схема алгоритма). Порядок выполнения подстановок полностью определяется указанными выше пунктами. Произвольная подстановка i должна выполняться тогда и только тогда, если она является первой из применимых подстановок. Процесс оканчивается, когда ни одна из подстановок не применима или выполнена заключительная подстановка.

Пусть даны три подстановки $\{yyx \to y, xx \to y, yyy \to \cdot x\}$. Тогда строка $P = xyxxyy$ путем применения второй, первой и третьей подстановок будет преобразована в $xyyxyy$, $xyyy$ и $x \cdot x$.

Следует отметить, что если бы третья подстановка не была заключительной, то процесс реализации алгоритма мог быть продолжен и результат был бы y.

Применение в схемах НА, наряду с обычными, заключительных подстановок, является обязательным для возможности реализации произвольных конструктивных алфавитных операторов. Дело в том, что любой алфавитный оператор и алгоритм, не имеющий в схеме заключительной подстановки, окончит работу лишь тогда, когда ни одна из подстановок не применима. Отсюда следует, что повторное воздействие алгоритма A на слово p уже не может изменить этого слова. Другими словами, для алгоритма A выполняется следующее тождественное соотношение: $A(A(p)) = A(p)$.

Таким образом, необходимым условием для универсальности НА является использования как обычных, так и заключительных подстановок.

Тезис. *Для любого алгоритма (конструктивно задаваемого алфавитного отображения) в произвольном конечном алфавите A можно построить эквивалентный ему нормальный алгоритм над алфавитом A.*

Данный тезис получил название принципа нормализации.

Выражение «над алфавитом A» означает, что иногда не удается построить НА, если использовать только буквы алфавита A. Однако можно построить требуемый НА, добавив к алфавиту A некоторое их количество (расширив A). Несмотря на расширение, алгоритм применяется лишь к словам исходного алфавита A.

Как показал А.А. Марков, если можно построить НА, эквивалентный заданному, добавив к алфавиту A некоторое количество букв (возможно, очень большое), то оказывается можно провести это построение НА, добавив к алфавиту A всего лишь одну букву.

Принцип нормализации нельзя доказать, поскольку понятие «произвольного» алгоритма не является строгим математическим понятием. Его необходимо воспринимать как один из естественнонаучных законов. Однако его считают в высшей степени достоверным. В сокращенном варианте он звучит так: «Все алгоритмы нормализуемы».

Справедливость этого принципа основана на том факте, что все известные алгоритмы нормализуемы. Можно показать, что все способы композиции алгоритмов не выводят вновь полученные алгоритмы за пределы класса нормализуемых. Для получения ненормализуемых алгоритмов нуж-

но принять такие меры, которые сегодня неизвестны. Были предприняты усилия для построения алгоритмов самого общего вида, но это не привело к факту вывода их за пределы указанного класса. Неудачу этих попыток следует рассматривать как убедительное доказательство в пользу принципа нормализации.

2.4.2 Композиции алгоритмов

Одним из распространенных видов композиции алгоритмов является «суперпозиция», когда выходное слово некоторого алгоритма A является входным для алгоритма B. Результатом этой операции следует считать $D(p) = B(A(p))$; суперпозиция может быть применена к любому количеству алгоритмов.

«Объединением» двух алгоритмов A и B в одном и том же алфавите X называют алгоритм C в том же алфавите, который преобразует любое входное слово p из области определения, являющейся пересечением областей определения A и B, в записанные рядом слова $A(p)B(p)$. На других входных словах алгоритм C считается неопределенным.

«Разветвлением» алгоритмов называют композицию трех алгоритмов A, B, C, результатом которой является алгоритм D. Считается, что область определения D совпадает с пересечением областей определения всех трех алгоритмов, а для любого слова p из этого пересечения $D(p) = A(p)$, если $C(p) = e$ или $D(p) = B(p)$, если $C(p) \neq e$.

«Повторение» (итерация) — это композиция двух алгоритмов A и B с результатом P. Для любого входного слова q выходное $P(q)$ определяют так: существует ряд слов $q = q_0, q_1, q_2, ..., q_n = P(q)$ такой, что для всех $i = 1, 2, 3, ..., n$ $q_i = A(q_i - 1)$, для всех $i = 1, 2, 3, ..., n - 1$ $B(q_i) \neq e$, а $B(q_n) = e$, т.е. алгоритм A применяется последовательно несколько раз до тех пор, пока не получится слово, перерабатываемое алгоритмом B в пустое e.

Все эти и другие методы композиции приводят к нормализованным алгоритмам. Поясним вышеизложенное на примерах.

Пусть дан алфавит $A = \{1, *\}$ и система подстановок $\{*111 \to 1*, * \to \cdot*, \to *\}$. Входное слово $P = 1111111$ путем последовательного применения третьей, первой и первой подстановок может быть преобразовано в слова $*1111111$, $1 * 1111$ и $11 * 1$. Этот алгоритм может быть рассмотрен как

операция деления числа 7 на число 3 (2 и 1 в остатке).

Пусть дан алфавит $A = \{x, y\}$ и система подстановок $\{yyx \to y, xx \to y, yyy \to \cdot y\}$. Входное слово $P = xyxxxyy$ путем последовательного применения второй, первой и третьей подстановок может быть преобразовано в слова $xyyxyy$, $xyyy$ и $x \cdot y$.

2.4.3 Универсальный нормальный алгоритм

Для всякой универсальной алгоритмической системы важным обстоятельством является задача построения так называемого «универсального» алгоритма. Поэтому встает задача построить нормальный алгоритм, способный выполнять функции любого алгоритма, если задана схема (то есть система подстановок) последнего.

Постановка этой задачи имеет много вариантов. Рассмотрим один из них. Зафиксируем некий стандартный алфавит Υ (например, двоичный). Определим способ кодирования для всех других возможных алфавитов в Υ, например, буквы любого алфавита нумеруются как $1, 2, 3, ...$ Затем i-ой букве присваивается двоичный код, где 01...10 ровно i единиц. Если общее число букв n, то можно ввести $(n+1)$-ую, $(n+2)$-ую, и т.д. буквы для обозначения знаков (стрелок, точек и т.д.). Записывая схему алгоритма одним словом и используя указанный код, получаем «изображение» данного алгоритма.

Поясним на примере. Пусть дан алгоритма A в алфавите $B = \{x, y\}$ и система подстановок $\{xy \to x, y \to \cdot\}$. Построим изображение алгоритма $A - A^U$.

Для этого обозначим $010 - x$, $0110 - y$, $01110 - \to$, $011110 - \cdot$, 0111110 — знак раздела, 01111110 — концевой знак.

$$A^U = 060010020030010050020030040060$$

Для краткости число единиц изображено в виде десятичного числа.

Аналогичным образом может быть получено в Υ изображение любого входного слова P^U из области определения этого алгоритма.

Справедлива следующая теореме об универсальном нормальном алгоритме.

Теорема. *Существует такой НА V, называемый универсальным, который для любого НА A и любого входного слова p переводит слово $A^U p^U$, по-*

лученное приписыванием изображения слова p к изображению алгоритма A, в слово, являющееся изображением соответствующего входного слова $A(p)$.

Эта теорема является очень важным математическим результатом. Дело в том, что из этой теоремы как следствие вытекает возможность построения машины, способной реализовать любой алгоритм нормализованного вида, то есть в принципе любого алгоритма. А специальное кодирование — это программа.

Итак, доказана принципиальная возможность нормализации всех известных алгоритмов, однако процесс нормализации крайне громоздок даже для простых алгоритмов, в следствие чего ЭВМ используют другие алгоритмические схемы.

2.5 Рекурсивные функции

Данная алгоритмическая система сложилась исторически первой (1931-1934 гг.), ввиду чего получила достаточно полное и всестороннее развитие. В ее основании лежит использование конструктивно определяемых арифметических целочисленных функций, которые получили специальное название — рекурсивные функции.

Определение. *Рекурсией называют способ задания функции, при котором значение определяемой функции для произвольных значений аргументов выражается через значения определяемой функции для меньших значений аргументов.*

Уточняя интуитивное определение алгоритма, А.А. Марков ввел понятие нормального алгоритма, что оказалось весьма удобным для специалистов в области математической логики. Однако для специалистов, имеющих дело преимущественно с ЭВМ, естественно иметь стандартный аппарат, приближенный к форме численных алгоритмов. Исследования показали, что любой алгоритм может быть всегда сведен к численному алгоритму.

Применение рекурсивных функций в теории алгоритмов основано на идее нумерации слов в произвольном алфавите последовательными числами. Например, логично нумеровать целыми числами слова, располагая их в порядке возрастания длины, а слова одинаковой длины — в произвольном порядке или, например, по алфавиту. Тогда после завершения ну-

мерации входных и выходных слов алфавитный оператор превращается в некоторую функцию вида $y = f(x)$, где как аргумент x, так и сама функция y, принимают целые неотрицательные значения. Эта функция может быть определена лишь для некоторых значений аргумента x, составляющих область определения функции y. Подобные, частично определенные, целочисленные функции получили название «арифметических».

Определение. *Функция $y = f(x_1, x_2, ..., x_n)$ называется алгоритмически вычислимой, если существует алгоритм, позволяющий определить значение функции при любых значениях переменных $x_1, x_2, ..., x_n$, входящих в область определения f.*

Это определение носит интуитивный характер, поскольку не ясно что понимается под вычисляющим алгоритмом. Для уточнения попытаемся построить класс всех вычислимых функций, начиная с простейших.

Назовем «элементарными» арифметические функции, получаемые из целых неотрицательных чисел и переменных с помощью операций сложения, арифметического вычитания, умножения и арифметического деления, а также построений сумм и произведений.

Примечание. *Под арифметическим вычитанием будем понимать получение абсолютной величины $\mid x - y \mid$.*

Примечание. *Под арифметическим делением будем понимать целое от частного a/b при $b \neq 0$.*

Вычислимость элементарных функций не вызывает никаких сомнений, поскольку операции четырех арифметических действий хорошо известны. В качестве исходного числа для построения элементарных функций можно взять число 1, т.к. $0 = \mid 1 - 1 \mid$, 1, $2 = 1 + 1$, $3 = (1 + 1) + 1$, и т.д. Очевидно, число элементарных функций бесконечно. Некоторые примеры:

- $f(x) = x + 1$;
- $f(y) = 50 \cdot x$;
- $f(a, b, c) = a \cdot b + c$;
- $f(x, y) = \mid x + y \mid /2$.

Попробуем ответить на вопрос: все ли вычислительные функции являются элементарными? Другими словами — шире ли класс вычислимых функций класса элементарных? Чтобы получить ответ на этот вопрос, будем усложнять элементарные функции, взяв за критерий сложности скорость роста значений функций.

Из всех элементарных функций быстрее всего растет произведение, так как $a \cdot n$ — это n раз повторенное сложение $a+a+a+a+...+a$. Очевидно, что умножение есть итерация сложения.

Операция возведения в степень, в свою очередь, есть итерация умножения: $a^n = a \cdot a \cdot a \cdot ... \cdot a$. И та и другая функции являются элементарными, так как выражаются через сложение и умножение.

Рассмотрим еще более быстро растущую функцию, которая является итерацией возведения в степень:

$$\psi(0,a) = a, \psi(1,a) = a^a, ..., \psi(n+1,a) = a^{\psi(n,a)}$$

Эта функция растет чрезвычайно быстро. Оказывается, что она не может быть реализована с помощью элементарных функций; начиная с некоторого числа $a = a*$ эта функция мажорирует над всеми элементарными функциями. Это означает, что для любой элементарной функции $\phi(a)$ найдется такое число $m*$, что будет выполняться неравенство $\phi(a) < \psi(m*, a)$ для всех $a \geq a*$, следовательно функция не является элементарной. Действительно, если бы $\psi(x,n)$ была элементарной функцией, то нашлось бы число $m*$ такое, что $\psi(x,n) < \psi(m*,n)$, при $n \geq 2$. Это справедливо при $n = m*$, $m* \geq 2$, получим $\psi(m*,m*) < \psi(m*,m*)$, что невозможно.

Таким образом, итерация операции возведения в степень позволяет получить неэлементарную функцию. В то же время $\psi(x,a)$ заведомо вычислима. Пусть нужно вычислить значение $\psi(x,a)$ при любых $x = n*$, $a = a*$. Используя обобщенную формулу, имеем при $a = a*$: $\psi(x+1,a*) = (a*)^{\psi(n,a*)}$. Обозначим $(a*)^m = \chi(m)$, где $\chi(m)$ — элементарная всюду вычислимая функция. Алгоритм ее вычисления сводится к повторенному m раз умножению на $a*$. Запись $\psi(n+1,a*) = \chi(\psi(n,a*))$ определяет связь значений функции в текущей и предыдущей точках. Задав начальное значение $\psi(0,a*) = a*$, получим последовательность вычисляемых значений:

$$\psi(1,a*) = \chi(\psi(0,a*)) = \chi(a*),$$

$$\psi(2,a*) = \chi(\psi(1,a*)) = \chi(\chi(a*)),$$

$$\psi(3, a*) = \chi(\psi(2, a*)) = \chi(\chi(\chi(a*))),$$

...

Продолжая вычисления до значения $\psi(n*, a*)$, можно показать, что функция ψ всюду однозначно определена. Это позволяет сделать вывод о том, что класс вычислительных функций шире класса всех элементарных функций.

Обратим внимание на тот факт, что $\psi(x, a)$ была задана по индукции, то есть вначале было определено начальное значение $\psi(0, a)$ — базис, а также указан способ вычисления всех ее значений по предыдущим значениям с помощью вполне доступных операций. Отметим, что определение по индукции может быть введено на любом упорядоченном множестве, где введены понятия «предыдущий» и «следующий».

Обозначим через x' функцию «следование за». Она определяет переход от одного элемента к следующему за ним. Для натурального ряда чисел $N = 0, 1, 2, 3, ...$ имеем: $0' = 1, 1' = 2, 2' = 3$ и т.д. Очевидно, в этом случае функция следования совпадает с функцией $x + 1$. Однако, в зависимости от вида множества приращения может и отличаться от 1.

Уточним общую схему задания функции $\phi(x)$ по индукции.

1. Задаем базис $\phi(0)$;

2. Для любого x укажем, каким образом значение $f(x')$ формально описано
$$\begin{cases} \phi(0) = q_0; \\ \phi(x') = \psi(x, \phi(x)). \end{cases}$$

В общем случае функция может быть зависимой от n переменных. Тогда схема будет иметь вид

$$\begin{cases} \phi(0, x_1, x_2, ..., x_n) = g(x_1, x_2, ..., x_n); \\ \phi(y', x_1, x_2, ..., x_n) = h(y, \phi(y, x_1, x_2, ..., x_n), x_2, x_3, ..., x_n). \end{cases}$$

Если g и h известны и вычислимы, тогда с помощью указанной схемы можно последовательно вычислять $\phi(1, x_1, x_2, ..., x_n)$, $\phi(2, x_1, x_2, ..., x_n)$ и так далее. Таким образом эта схема определяет вычислимую функцию.

Какие же арифметические функции могут быть определены по индукции? Чтобы дать конкретный ответ необходимо указать какие функции считаются известными первоначально и какие операции кроме, кроме схемы индукции, допустимы при определении функций.

Среди арифметических функций выделим следующие, особо простые, которые получили название «элементарные арифметические».

1. $\phi(x) = x'$ — функция следования, применимая для натурального ряда (сокращенное обозначение $S(x)$);

2. $\phi(x_1, x_2, ..., x_n) = q, q = const$ — функция константы (сокращенное обозначение C_q^n);

3. $\phi(x_1, x_2, ..., x_n) = x_i$ — функция тождества (сокращенное обозначение X_i^n);

4. $\phi(x_1, x_2, ..., x_n) = \psi(\chi_1(x_1, x_2, ..., x_n), ..., \chi_m(x_1, x_2, ..., x_n))$ — функция подстановки.

Используя в качестве исходных функций перечисленные элементарные функции, можно с помощью небольшого числа конструктивных приемов строить все более и более сложные арифметические функции.

Полная система формального описания схем для получения произвольных функций может быть представлена следующим образом.

1. $\phi(x) = S(x) = x'$;

2. $\phi(x_1, x_2, ..., x_n) = C_q^n = q$;

3. $\phi(x_1, x_2, ..., x_n) = X_i^n = x_i$;

4. $\phi(x_1, x_2, ..., x_n) = \psi(\chi_1(x_1, x_2, ..., x_n), ..., \chi_m(x_1, x_2, ..., x_n))$;

5. $$\begin{cases} \phi(0) = q; \\ \phi(x') = \psi(x, \phi(x)). \end{cases}$$

6. $$\begin{cases} \phi(0, x_1, x_2, ..., x_n) = g(x_1, x_2, ..., x_n); \\ \phi(y', x_1, x_2, ..., x_n) = h(y, \phi(y, x_1, x_2, ..., x_n), x_2, x_3, ..., x_n). \end{cases}$$

Схемы под номерами 1-4 задают первоначальные функции, играя роль аксиом, а схемы 5 и 6 можно рассматривать как правила вывода.

Определение. *Функция $\phi(x_1, x_2, ..., x_n)$ называется примитивно рекурсивной, если она может быть определена с помощью конечного числа применений схем 1-6.*

Рассмотрим пример. Определим функцию $\phi(x, y)$ следующим образом.

$$\begin{cases} \phi(0, x) = x; \\ \phi(y', x) = [\phi(y, x)]'. \end{cases}$$

В соответствии с вышеприведенным определением можно представить следующую последовательность:

$$\phi(1, x) = x' = x + 1,$$

$$\phi(2, x) = [\phi(1, x)'] = (x + 1)' = x + 2,$$
$$\phi(3, x) = [\phi(2, x)'] = (x + 2)' = x + 3,$$

$$...$$

$$\phi(y, x) = x + y.$$

Таким образом, заданная функция соответствует операции сложения двух аргументов x и y.

Примитивно-рекурсивное описание этой функции составляется с использованием схемы 5.

$$\begin{cases} \phi(0, x) = g(x); \\ \phi(y', x) = h(y, \phi(y, x), x). \end{cases}$$

Здесь функция $g(x)$ имеет вид $g(x) \equiv x$, т.е. это функция тождества $X_1^1(x)$.

В свою очередь функция $h(y, z, x) \equiv z'$ может быть получена из первоначальной функции $X_2^3(y, z, x) \equiv z$ с помощью использования схемы 4, в качестве ϕ берется функция следования $S(z) \equiv z'$. Тогда имеем $h(y, z, x) = S[X_2^3(y, z, x)]$. Примитивно-рекурсивным описанием функции h будет последовательность S, X_2^3, x. Добавляя к ней функцию $g(x) \equiv X_1^1(x)$, от которой зависит функция $\phi(y, x)$ получаем: S, X_2^3, x, X_1^1, ϕ.

Рассмотрим еще один пример. Зададим следующую функцию.

$$\begin{cases} \phi(0,x) = 0; \\ \phi(y',x) = \phi(y,x) + x. \end{cases}$$

Тогда получим последовательность:

$$\phi(1,x) = \phi(0,x) + x = x,$$

$$\phi(2,x) = \phi(1,x) + x = x + x = 2 \cdot x,$$

$$\phi(3,x) = \phi(2,x) + x = 2 \cdot x + x = 3 \cdot x,$$

$$...$$

$$\phi(y,x) = y \cdot x.$$

Очевидно, что произведение аргументов x и y — также есть примитивно-рекурсивная функция. К этому же классу относятся возведение в степень и многие другие операции.

Итак, мы установили, что класс примитивно-рекурсивных функций содержит вычислимые функции. Можно поставить вопрос — все ли вычислимые функции входят в класс примитивно-рекурсивных?

Оказывается, класс вычислимых функций шире, чем класс примитивно-рекурсивных функций. Это показали независимо друг от друга Р. Петер и В. Аккерман, что означает, что средства построения вычислимых функций нуждаются в расширении. Операторы рекурсии не дают желаемого замыкания класса всех вычислимых функций. Подходящим для этой цели оказался μ-оператор.

Определение. *Функция называется частично-рекурсивной, если она построена из простейших функций константы C_q^n, тождества X_i^n, следования $S(x)$ и с помощью применения конечного числа операций суперпозиции, примитивной рекурсии и μ-оператора.*

Определим операцию с применением μ-оператора как «операцию наименьшего корня». Эта операция позволяет определить новую арифметическую функцию $f(x_1, x_2, ..., x_n)$ от n переменных с помощью ранее построенной арифметической функции $g(x_1, x_2, ..., x_n, y)$ от $n+1$ переменной. Для $x_1 = a_1, x_2 = a_2, ..., x_n = a_n$ в качестве соответствующего значения $f(a_1, a_2, ..., a_n)$ определяемой функции принимается наименьший целый неотрицательный корень $y = \beta$ из уравнения $g(a_1, a_2, ..., a_n, y) = 0$.

Если такой целый корень не существует, функция считается неопределенной при соответствующем наборе значений переменных.

Итак, среди рекурсивных функций появляются полностью определенные (частичные) функции, а операторы над частичными функциями порождают новые частичные функции. Характер неопределенности может быть довольно сложным. Действительно, для набора значений $x_1, x_2, ..., x_n$ не всегда найдется способ установить, определена ли функция f на этом наборе, что вызывает продолжение процесса вычислений в течении неопределенного времени, не зная заранее, остановится он или нет.

В случае, когда функция g является частичной, естественным образом функция $f(x_1, x_2, ..., x_n) = \mu y(g(x_1, x_2, ..., x_n, y) = 0)$ вычисляется с учетом следующего условия: если для $y = 0, 1, ..., i-1$ функция $g(x_1, x_2, ..., x_n, y) \neq 0$,а $g(x_1, x_2, ..., x_n, i)$ не определена, то и $f(x_1, x_2, ..., x_n)$ также не определена.

Определение. *Частично-рекурсивная функция называется общерекурсивной, если она всюду определена.*

Понятие частично-рекурсивной функции оказалось достаточной формализацией понятия вычислимой функции, что содержится в тезисе А. Черча

Тезис. *Всякая функция, вычислимая некоторым алгоритмом, частично-рекурсивна.*

2.6 Заключительные замечания по алгоритмическим системам

Начиная с 1930-ых гг. было предложено значительное количество алгоритмических систем для уточнения интуитивного понятия эффективная процедура вычисления или алгоритм. При этом последняя из них, машина с бесконечными регистрами, была предложена сравнительно недавно (публикации 1980-90 гг.).

Возникают следующие вопросы.

1. Каким образом можно сопоставить между собой все эти системы?

2. Насколько хорошо эти системы характеризуют неформальное понятие вычислимости?

Перечислим основные подходы к уточнению алгоритмов различных авторов:

- Общерекурсивные функции (Гедель, Эрбран, Клини, Мендельсон);
- λ-определимые функции (Черч);
- μ-рекурсивные функции и частично-рекурсивные функции (Гедель, Клини);
- Машина Тьюринга-Поста (Тьюринг, Пост);
- Функции, определяемые классическими дедуктивно-описываемыми системами (Минский, Пост);
- Нормальные алгоритмы Маркова (Марков, Мендельсон);
- МБР-вычислительные функции(Шепердсон, Стерджис, Катленд).

Несмотря на значительные отличия, все эти системы имеют фундаментальную общую черту: каждая из указанных алгоритмических систем приводит в конечном счете к одному и тому же классу вычислимых функций ϕ.

Этот вывод дает достаточно ясный ответ на первый вопрос, то есть с точки зрения сопоставимости все эти алгоритмические системы эквивалентны.

Марков, Тьюринг, Черч и некоторые другие в своих исследованиях предложили некие утверждения (гипотезы, тезисы, принципы) о том, что класс определенных ими функций совпадает с неформально определенным классом эффективно вычислимых функций. В современной научной литературе все эти утверждения стали называть тезисами Черча (наиболее общая из всех формулировок), которые приняли и к МБР: «интуитивно и неформально определенный класс эффективно вычислительных функций совпадает с классом МБР-вычислимых функций ϕ».

Данный результат — не теорема, он недоказуем, а имеет статус утверждения, которое следует принимать как аксиому с высокой степенью достоверности.

Во-первых, все независимые исследования по уточнении эффективности вычислимости привели к одному и тому же классу ϕ.

Во-вторых, огромное семейство вычислимых функций входит в этот класс ϕ и его можно расширить до бесконечности известными методами.

В-третьих, реализация программ P для МБР наглядно свидетельствует о наличии алгоритма, который может быть реализован реальным компьютером.

В-четвертых, никому еще не удалось найти такую функцию, которую можно было бы признать вычислимой и одновременно не принадлежащей к классу ϕ.

3 Алгоритмически неразрешимые проблемы

Одним из важнейших свойств алгоритма является массовость, то есть способность решать не одну конкретную задачу, а целый класс однотипных задач.

Как показали исследования, существуют такие классы задач, для которых не существует единого универсального приема решения. Такие задачи в теории алгоритмов получили название алгоритмически неразрешимых проблем. Заметим, что алгоритмическая неразрешимость не означает, что такие задачи нельзя решить. Речь идет о невозможности решения одним и тем же приемом.

Примером алгоритмически разрешимой проблемой проблемы является доказательство тождеств в алгебре. Известна последовательность действий: раскрыть все скобки, привести подобные члены и перенести их все на одну сторону, если $0 = 0$, то тождество истинно, иначе ложно.

Вместе с тем, уже ранние исследования в области построения алгоритмов показали неразрешимость некоторых задач. Одним из первых результатов подобного рода является доказательство неразрешимости проблемы распознавания выводимости в математической логике, известное в литературе под названием «теорема Черча» (А. Черч, 1936 г.). Основная идея состоит в том, что Черч смог показать нерекурсивность функции, решающей данную задачу, в результате чего стало ясно, что искать алгоритм бессмысленно.

Одним из примеров алгоритмически неразрешимой проблемы является проблема распознавания «самоприменимости». Существуют самоприменимые и несамоприменимые алгоритмы. Например, тождественный алгоритм в любом алфавите А, содержащем две и более букв, является самоприменимым, способным перерабатывать любое входное слово $p \in$ А в само себя.

С другой стороны, нулевой алгоритм в алфавите А является несамоприменимым. Этот алгоритм задается схемой, содержащей единственную подстановку «$\rightarrow y$»($y \in A$). По определению алгоритм неприменим ни к какому входному слову, а значит и к своему изображению.

Проблема самоприменимости алгоритмов состоит в том, чтобы установить за конечное количество шагов по схеме любого алгоритма является ли он самоприменимым или нет. Рассмотрим проблему детальнее посредством алгоритмической системы машины Тьюринга.

Пусть на ленте МТ зафиксирована некоторая конфигурация. Тогда возможны два случая:

1. МТ применима к этой конфигурации, тогда она после N шагов остановится в заключительном состоянии.

2. МТ неприменима к этой конфигурации, то есть она никогда не достигнет заключительного состояния и будет продолжать свою работу бесконечно.

Теорема. *Проблема распознавания самоприменимости алгоритма является неразрешимой.*

Рассмотрим вариант универсальной МТ, когда на ленте изображен ее собственный шифр (таблицы соответствия и начальная конфигурация) в алфавите машины. Естественно считать, что если МТ обработает этот шифр — она самоприменима, иначе — несамоприменима.

Предположим, что такая МТ, анализирующая собственный шифр и выносящая решение о самоприменимости, существует. Тогда в этой МТ всякий самоприменимый шифр перерабатывается в некоторый символ ν (положительный ответ), а всякий несамоприменимый шифр — в символ τ (отрицательный ответ).

При таком допущении можно построить такую МТ', которая перерабатывает несамоприменимые шифры в τ (как и МТ), но неприменима к самоприменимым шифрам. Это достигается введением такого изменения таблицы соответствия, когда после появления символа ν вместо остановки МТ' бесконечно повторяла бы $\nu, \nu, \nu, ...$

Итак, МТ' применима ко всякому несамоприменимому шифру и неприменима к самоприменимым шифрам. Это приводит к противоречию.

1. Пусть МТ' самоприменима, то есть она она применима к собственному шифру МТ' и останавливается, но тогда она несамоприменима, так как вырабатывает τ.

2. Пусть МТ' несамоприменима, что означает ее применимость к ее собственному шифру МТ', то есть она самоприменима.

Указанное противоречие и есть доказательство теоремы, так как оно опровергает утверждение о существовании МТ.

Из этой теоремы вытекает алгоритмическая неразрешимость более широкой проблемы — проблемы распознавания применимости, заключающейся в необходимости установить применимость любой МТ к любой заданной конфигурации.

В терминах нормальных алгоритмов проблема самоприменимости может быть сформулирована следующим образом: «Требуется найти единственный прием, позволяющий за конечное число шагов по схеме любого алгоритма A узнать, является ли A самоприменимым или нет».

С точки зрения НА единый конструктивный прием — это некий нормальный алгоритм B, определенный на любом слове p, которое является изображением произвольного НА A, переводящий это слово в два различных фиксированных слова q_1 и q_2 в зависимости от того, будет ли алгоритм A самоприменимым или нет.

На любом входном слове L, не являющемся изображением какого-либо НА, алгоритм B также должен быть определен. Иначе, после некоторого числа шагов работы, было бы неизвестно, изображением какого алгоритма (самоприменимого или нет) является слово L. Результат обработки алгоритмом B слова L, не являющегося изображением алгоритма, должен отличаться от q_1 и q_2.

Предположим, что B существует. В таком случае существует и НА C в том же алфавите X, что и алгоритм B, определенный на всех тех и только тех словах в X, которые являются изображениями несамоприменимых алгоритмов.

Для этого построим НА D в алфавите X, область определения которого состоит из одного слова q_2. Его можно задать, например, в виде суперпозиции двух НА $D1$ и $D2$. $D1$ задается схемой, состоящей из одной подстановки «$q_2 \to *$», а $D2$ — схемой из подстановок «$x_i \to x_i$», где x_i принимает последовательно значение всех букв алфавита X. $D1$ переводит в пустое слово только слово q_2, а область определения $D2$ — пустое слово. Область определения суперпозиции D алгоритмов $D1$ и $D2$ содержит лишь q_2. Построив алгоритм D, образуя суперпозицию с ним алгоритма B и нормализуя эту суперпозицию, получим НА C в алфавите X, область определения которого состоит из тех и только тех слов X, которые являются записями несамоприменимых алгоритмов. Однако подобное свойство алгоритма C внутренне противоречиво, поскольку к своему собственному изображению алгоритм C не может быть одновременно применимым и неприменимым.

Действительно, если алгоритм C применим к своему изображению, то он самоприменим. Это противоречит тому, что алгоритм C, в силу своего построения, должен быть применим только к несамоприменимым алгоритмам. С другой стороны, если алгоритм C неприменим к своему изображению, то он является несамоприменимым. Но тогда он должен быть применим к своему изображению, так как он применим к изображениям всех несамоприменимых алгоритмов. Следовательно, алгоритм C — самоприменим. Таким образом, найдено логическое противоречие, демонстрирующее алгоритмическую неразрешимость всей проблемы (при условии, что принцип нормализации справедлив).

Природа противоречия, использованного в ходе рассуждений, имеет более глубокие корни и связана с парадоксом Рассела («множество всех множеств, не содержащее себя в качестве своего элемента»).

Этот же метод «от противного» может быть использован при доказательстве неразрешимости проблемы эквивалентности слов для ассоциативного исчисления.

Определение. *Ассоциативным исчислением называется совокупность всех слов в некотором алфавите A с конечной системой подстановок.*

Чтобы задать ассоциативное исчисление достаточно указать алфавит A и систему подстановок. Подстановки бывают вида «$p \leftrightarrow q$» или «$p \to q$», где p, q — слова в алфавите A. Неориентированная подстановка позволяет замену в прямом и обратном направлениях, ориентированная — лишь в прямом.

Допустим, задан алфавит $A = \{a, b, c\}$ и одна подстановка «$ab \leftrightarrow bcb$», а также $p = abcbcbab$. Заменим вхождения ab: $bcbcbcbbcb$, а затем используем подстановку в обратном направлении: $abcbcbab \to aabcbab \to aaabab$.

Если слово r может быть получено из слова s применением лишь одной подстановки, то говорят, что r и s смежные слова. Если слово r может быть получено из слова s применением конечного числа подстановок, то говорят, что смежные слова образуют «дедуктивную цепочку от s к r». Наконец, слова s и r называются «эквивалентными» ($s \sim r$), если существует дедуктивная цепочка от s к r.

Для отношения эквивалентности слов справедливо следующее.

1. Рефлексивность: $r \sim r$.

2. Симметричность: $r \sim s \leftrightarrow s \sim r$.

3. Транзитивность: $r \sim t, t \sim s \rightarrow r \sim s$.

Проблема эквивалентности в ассоциативном исчислении состоит в необходимости определения для для любых двух слов в данном ассоциативном исчислении эквивалентны они или нет.

Впервые, в 1914 г., данная проблема была сформулирована А. Туэ. Он предложил некоторые алгоритмы распознавания эквивалентности слов для частных специальных исчислений с повторениями сочетаний букв в словах заданного алфавита Σ. Туэ назвал слово «бесквадратным», если оно не содержит сочетаний вида $x^2 = xx$ (или «бескубным», если оно не содержит сочетаний вида x^3), где $x \neq \lambda$.

Суть решаемой Туэ проблемы состояла в построении бесквадратных (бескубных) слов максимальной длины в Σ. После этого многие математики предпринимали усилия для построения такого общего алгоритма, который бы для любого ассоциативного исчисления и для любой пары слов в нем позволял бы установить, являются ли они эквивалентными. В 1947 г. Э. Пост и в 1964 г. А.А. Марков независимо друг от друга построили примеры ассоциативных исчислений, где проблема эквивалентности слов оказалась алгоритмически неразрешимой. Это позволило сделать вывод о том, что искомого алгоритма не существует в принципе. В 1955 г. П.С. Новиков доказал алгоритмическую неразрешимость проблемы распознавания эквивалентности слов для ассоциативного исчисления специального вида — теории групп.

Следует отметить, что примеры, построенные для опровержения алгоритмической разрешимости проблемы эквивалентности слов в ассоциативных исчислениях, оказались очень сложными и громоздкими, включающими в себя сотни допустимых подстановок. Однако в 1959 г. Г. Цейкинсну удалось построить пример ассоциативного исчисления, имеющего всего семь допустимых подстановок, для которого проблема распознавания эквивалентности слов оказалась также алгоритмически неразрешимой.

Пусть задан алфавит $A = \{a, b, c, d, e\}$ и следующая система подстановок.

- $ac \rightarrow ca$;
- $ad \rightarrow da$;

- $bc \to cb$;
- $eca \to ae$;
- $bd \to db$;
- $abac \to abace$;
- $be \to edb$.

Рассмотрим некоторое слово $p_1 = abcde$. Используя подстановки, построим дедуктивную цепочку $acbde \to cabde \to cadbe \to cadedb$. Соответственно, $abcde \sim cadedb$. Рассмотрим другое слово $p_2 = aaabb$. Очевидно, к нему неприменима ни одна из данных подстановок, а это значит, что p_2 не имеет ни смежных, ни, следовательно, эквивалентных слов. Цейкинс успешно доказал, что для любой пары слов в этом ассоциативном исчислении нет и не может быть единого алгоритма распознавания эквивалентности.

Существует не один десяток примеров алгоритмически неразрешимых проблем. Одним из важнейших является теорема Геделя о неполноте арифметической логики (совокупности аксиом и правил вывода в элементарной теории чисел).

Определение. *Логика называется полной, если в рамках ее можно доказать истинность или ложность каждого утверждения.*

Определение. *Логика называется непротиворечивой, если она свободна от противоречий (например, в ней нельзя получить одновременно истинное утверждение* A *и ложное* $\neg$A*).*

Теорема. *Каждая адекватная ω-непротиворечивая арифметическая логика неполна.*

Математический смысл теоремы в том, что в арифметической логике существуют истинные утверждения о целых положительных числах, которые нельзя вывести и доказать средствами этой логики. Кроме того, Гедель показал, что невозможно доказать непротиворечивость арифметической логики теми методами, которые выразимы в данной логике.

Результаты работы Геделя являются фундаментальными и одними из наиболее значимых для современной математики

Первой известной аксиоматической системой принято считать геометрию Евклида(3 век до нашей эры). В ее основе лежит набор определений и аксиом, которые отражают простейшие геометрические свойства, не вызывающие сомнений и подтвержденные всем опытом человечества. В их

число входит и аксиома о параллельных прямых: «Если две прямые, лежащие в одной плоскости, при пересечении их какой-нибудь третьей образуют внутренние углы, сумма которых меньше двух прямых, то эти прямые пересекаются по ту сторону от третьей, на которой сумма указанных углов меньше двух прямых».

Данная аксиома является наиболее сложной, ее пытались вывести из какой-либо другой, но безуспешно. В истории математики она получила название «темное пятно Евклида». Не удалось также доказать полноту и противоречивость системы аксиом Евклида Декарту, Лейбницу, Гауссу и многим другим выдающимся математикам.

В 1826 г. бессмысленность этих попыток впервые строго доказал русский математик Н.И. Лобачевский. В частности он показал, что данная аксиома Евклида невыводима из остальных аксиом. Совокупность его научных трудов получила название «геометрия Лобачевского» («неевклидова геометрия»). Одним из результатов Лобачевского является отрицание истинности одной из аксиом Евклида: «Если даны прямая и точка, не лежащая на ней, то существует только одна прямая, которая проходит через данную точку параллельно первой прямой». Таким образом, в геометрии Лобачевского нет параллельных прямых, а, в то же время, геометрия Евклида есть частный случай геометрии Лобачевского. Немецкий математик Б. Риман доказал, что неевклидова геометрия может быть непротиворечивой только в том случае, если удастся доказать непротиворечивость геометрии Евклида. Другой выдающийся немецкий математик Д. Гильберт в свою очередь показал, что геометрия Евклида могла бы быть непротиворечивой, если бы удалось доказать непротиворечивость арифметики. Для этого нужно было найти вариант такой арифметической логики, которая оказалась бы полной, то есть в которой можно вывести в качестве теорем все истинные утверждения о целых положительных числах.

Теперь вернемся к теореме Геделя. Оказалось, что такой арифметической логики нет и быть не может, а это значит, что существует бесконечное число проблем элементарной теории чисел, решение которых невозможно никаким аксиоматическим методом.

Одним из практических следствий теоремы Геделя является то, что алгебра Буля в алфавите $A = \{0, 1\}$, который есть подмножество $B = \{0, 1, 2, ..., 9\}$ или $A \subseteq B$, может иметь также истинные выражения и формулы, которые невозможно вывести из системы тождественных соотноше-

ний. Этот результат напрямую касается вопросов проектирования систем обработки информации.

4 Схемы алгоритмов

4.1 Логические схемы алгоритмов

4.1.1 Формальное определение логической схемы

Последовательности элементарных действий, выполняемых в течение одного микротакта синхросерии генератора тактовых импульсов, получили в вычислительной техники название микроопераций (или микроинструкций). Совокупность же микроопераций, выполняемых одновременно в i-том такте, называется микрокомандой. Иногда последовательность микроопераций является неизменной, однако более общим и важным случаем является случай, когда порядок их выполнения существенно зависит от результата выполнения предыдущих действий, то есть от некоторых условий.

Используя аппарат логики, это можно представить так. Предположим, нужно сравнить два числа x и y по абсолютной величине. Тогда запись вида $P(|x| = |y|)$ символизирует условие: если $|x| = |y|$, то $P = 1$, иначе $P = 0$. Это условие — основа для разветвления микропрограммы. Нередко условие в целом зависит от нескольких простых условий $p_1, p_2, ..., p_k$, т.е.: $\alpha_i = \alpha(p_1, p_2, ..., p_k)$.

Вполне понятно, что в этом случае выполнение микропрограммы является однозначно заданной функцией от k заданных условий. В работах А.А. Ляпунова подобная функция записывается в виде конечной строки, включающей символы $A_1, A_2, ..., A_n$ (микрооперации), логические функции $\alpha_1, \alpha_2, ..., \alpha_m$ и символов $\uparrow^i$ (начало) и $\downarrow^i$ (конец) стрелок, где $i = 1, 2, 3, ...$

В дальнейшем символы $\mathrm{A}_i (i = 1, 2, ..., n)$ будем называть «операторами». Строка $A_1\alpha \uparrow^1 A_2 ... \downarrow^1 A_i ... A_n$ в этом случае означает, что после выполнения оператора A_1 может быть два случая.

- $\alpha = 1$, тогда выполняется оператор A_2;
- $\alpha = 0$, тогда выполняется оператор A_i, непосредственно справа от конца стрелки $\downarrow^1$.

При этом всегда будем считать, что функция α может принимать лишь два значения — 0 или 1.

Условимся логическую функцию $\alpha \uparrow^i$ называть «логическим условием».

Определение. *Логической схемой алгоритма (ЛСА) называют конечную строку из символов* $A_1, A_2, ..., A_n$, *логических условий* $\alpha_1, \alpha_2, ..., \alpha_m$ *и концов стрелок* $\downarrow^1, ..., \downarrow^m$ *такую, что для каждого начала стрелки* $\uparrow^i$ *найдется один и только один конец стрелки с тем же индексом.*

Будем считать, что в ЛСА каждый оператор A_i встречается только один раз (в случае, если это не так, можно ввести переобозначение).

Назовем символы операторов и логических условий (ЛУ) «элементарным выражением», и всякую строку, состоящую из таких выражений и концов стрелок, так что для каждого индекса i найдется не более одного начала и одного конца стрелки, «выражением».

Итак, ЛСА по вышеизложенному определению задает порядок выполнения операторов в зависимости от значений ЛУ.

Рассмотрим ЛСА $U(p_1, ..., p_m, A_1, ..., A_n)$, где p_i — ЛУ, являющиеся независимыми логическими переменными. По условию каждая переменная может принимать значение 0 или 1.

Обозначим всевозможные наборы значений переменных $p_1, ..., p_m$ через $\Delta = \Delta_1, \Delta_2, ..., \Delta_{2^m}$.

Тогда процесс реализации ЛСА для произвольной последовательности Δ может быть определен следующим образом:

- На первом шаге задаем значения переменных в виде Δ_{s1}.
- На втором — отмечаем самое левое элементарное выражение схемы алгоритма U.
- ...
- После нескольких последовательно выполняемых шагов получаем: $A_{i1}, A_{i2}, ..., A_{i(L-1)}$.
- Проанализируем элементарное выражение, следующего шага.
  - Если это оператор A_{iL}, то приписываем его к полученной строке справа и задаем значения набора переменных в виде $\Delta_{s(L+1)}$, а затем переходим к следующему элементарному выражению.
  - Если же это ЛУ $\alpha_j(p_1, ..., p_m)$ $\uparrow^i$, то действуем уже описанным способом, то есть при $\alpha_j = 1$ переходим к следующему элемен-

тарному выражению справа, иначе — к ближайшему справа от конца $\downarrow^i$.

- Процесс реализации ЛСА заканчивается, если выполнен заключительный оператор A_n.

Полученную строку операторов назовем значением ЛСА U для последовательности наборов Δ. При этом заметим, что значения ЛУ могут изменяться только во время выполнения операторов.

Введем некоторые понятия и определения, предложенные в работах Ю.И. Янова.

Пусть задана ЛСА U и два множества $\Phi = \{A_1, A_2, ..., A_n\}$ и $\Psi = \{p_1, ..., p_m\}$, принадлежащие U. Каждому оператору $A_i (i = 1, 2, ..., n) \in A$ поставим в соответствие некоторое множество ЛУ $\Psi_i \subseteq \Psi$, которые могут изменять свои значения на обратные во время выполнения оператора A_i. Ю.И. Янов назвал это соответствие «распределением сдвигов» : $A_i - \Psi_i$.

Среди всевозможных вариантов распределений сдвигов различают следующие.

1. Пустое, когда оператору поставлено в соответствие пустое множество $\Psi_i = \lambda$.

2. Универсальное, когда оператору поставлено в соответствие множество всех логических переменных $\Psi_i = \Psi$.

3. Нормальное, когда $\Psi_i \subset \Psi$.

Положим, ЛСА для фиксированной последовательности наборов логических условий $\Delta_{s1}, \Delta_{s2}, ..., \Delta_{st}, \Delta_{s(t+1)}, ...$ реализуется в виде цепочки операторов $A_{i1}, A_{i2}, ..., A_{is_t}, A_{is_{(t+1)}}, ...$

Последовательность наборов рассмотренного вида называют допустимой для схемы алгоритма при заданном распределении сдвигов, если всякий набор логических переменных $\Delta_{s(t+1)}$ либо совпадает с набором Δ_{st}, либо отличается от него значениями переменных из множества Ψ_{it}.

Говорят, что две ЛСА $U1$ и $U2$ «равносильны» при заданном распределении сдвигов $(U1 = U2)$, если для всякой допустимой последовательности наборов значения этих ЛСА совпадают.

Само по себе это определение не дает эффективного способа для определения равносильности из-за огромного перебора в случае сложных ЛСА, но зато помогает получить ряд формул преобразований ЛСА в равносильную ей.

Равносильное преобразование состоит в замене одних выражений другими и в изменении взаимного расположения выражений. Это оказывается чрезвычайно полезно в процессе создания систем обработки информации, когда алгоритмы функционирования представлены в терминах ЛСА, кроме того при оптимизации ПО.

Будем говорить, что в ЛСА $U(p_1, ..., p_m, A_1, ..., A_n)$ элементарное выражение D_i при заданном распределении сдвигов «подчинено» логической функции $\alpha(p_1, ..., p_m)$, и записывать этот факт как $D_i \prec \alpha(p_1, ..., p_m)$, если всякий набор Δ_r, при котором должно выполняться D_i, обращает функцию α в единицу.

Например, $U = A_1\alpha(p_1, ..., p_m) \uparrow^1 A_2A_3A_4 \downarrow^1 A_5$. Здесь оператор $A_2 \prec \alpha(p_1, ..., p_m)$, так как после выполнения A_1 оператор A_2 выполняется лишь в случае, если $\alpha = 1$. Оператор A_3 не подчинен α, так как оператор A_2 при распределении сдвигов A_2–$\Psi_2 \subset \Psi$ может изменить значения некоторых переменных таким образом, что функция α при выполнении A_3 уже не будет равна 1. Тем не менее, в работе В.Ф. Дъяченко показано, что оператор A_3 подчинен некоторой логической функции $\beta(p_1, ..., p_m)$, которая является $\beta = max\alpha(p_1, ..., p_m)$ на всех наборах значений переменных множества Ψ_2.

Свойства функции $\beta = max(\Psi_i)\alpha$ таковы, что если

- $\alpha(\Delta) = 1$, то $\beta(\Delta) = \beta(\Delta') = 1$,
- $\alpha(\Delta) = 0$ и $\alpha(\Delta') = 0$, то $\beta(\Delta) = \beta(\Delta') = 0$,
- $\alpha(\Delta) = 0$ и $\alpha(\Delta') = 1$, то $\beta(\Delta) = \beta(\Delta') = 0$,

где Δ' — набор значений переменных $p_1, ..., p_m$, который отличается от набора Δ значениями тех переменных, которые принадлежат Ψ_i. Отсюда следует, что A_4, например, подчинен исходной функции $A_4 \prec max(\Psi_3)\beta = max(\Psi_3)(max(\Psi_2)\alpha)$.

Назовем конъюнкцию логических переменных «элементарной», если в ней не содержится повторяющихся ЛУ. Чтобы по заданной логической функции найти ее max по всем наборам переменных из множества Ψ_i, следует.

1. Представить α в ДНФ.

2. В каждой элементарной конъюнкции все переменные, входящие в Ψ (в том числе и с отрицанием), заменить единицей.

Рассмотрим пример. Дана функция $\alpha = p_1\neg p_2 \vee p_3(\neg p_1 p_2 \vee p_4)$, тогда $\beta = max(\{p_1, p_4\})[p_1\neg p_2 \vee p_3(\neg p_1 p_2 \vee p_4)] = max(\{p_1, p_4\})[p_1\neg p_2 \vee \neg p_1 p_2 p_3 \vee p_3 p_4] = 1\neg p_2 \vee 1 p_2 p_3 \vee p_3 1 = \neg p_2 \vee p_3$

4.1.2 Полная система преобразований Янова

Ю.И.Янов в своих работах предложил специальное ассоциативное исчисление, где формулы представлены как $B = G$, где B и G — выражения. Во всякой ЛСА $U(B)$ можно провести операцию подстановки $B \to G$. Это дает возможность получить ЛСА с минимумом ЛУ, но путем перебора всевозможных ЛСА, равносильных заданной. Сложность подобных преобразований чрезвычайно высока, ввиду чего были предприняты усилия, направленные на обхождение указанной проблемы.

4.1.3 Представление ЛСА системой формул перехода

В работах В.Г. Лазарева и В.Ф. Дъяченко предлагается применять специальные формализмы для особой записи ЛСА, которая позволяет применять уже известные апробированные методы, развитые в теории релейно-контактных схем.

Если $\alpha, \beta, \gamma, ...$ — логические функции от переменных $p_1, ..., p_n, \mathrm{A}_i \in A$, где A — множество операторов $\mathrm{A} = \{A_1, ..., A_m\}$, то запись вида $\alpha\mathrm{A}_i$ (или $p_j\mathrm{A}_i$) назовем «отмеченной функцией» (отмеченной переменной).

$$\alpha A_i = \begin{cases} A_i & , \alpha = 1, \\ 0 & , \alpha = 0. \end{cases}$$

В работах П.С. Новикова показано, что над логическими функциями, входящими в состав отмеченных, можно производить все преобразования алгебры логики.

Для них справедливы следующие правила, сокращенно называемые системой $C1$.

1. $(\alpha \vee \beta)A_i = \alpha A_i \vee \beta A_i$;

2. $\alpha\beta A_i \vee \alpha\gamma A_j = \alpha(\beta A_i \vee \gamma A_j)$;

3. $\alpha = \beta \to \alpha A_i = \beta A_i$.

Указанную систему правил совместно с полной системой аксиом алгебры логики следует рассматривать как систему правил тождественных преобразований отмеченных функций.

Запись вида «$\mathrm{A}_i \rightharpoonup \mathrm{A}_j$» следует читать как «за A_i следует A_j», то есть $U = ...\mathrm{A}_i\mathrm{A}_j....$

Между всеми операторами $\mathrm{A}_i \in \mathrm{A}$ в ЛСА существует различная возможность взаимопереходов, что может быть формально описано следующим образом: $\mathrm{A}_i \rightharpoonup \alpha_{i1}A_i \vee \alpha_{i2}A_2 \vee ... \vee \alpha_{in}A_n$, где $\mathrm{A}_i \in \mathrm{A}$ — операторы, $\alpha_{ij}(p_1, ..., p_m)$ — логические функции от независимых переменных, $\alpha_{ij}\mathrm{A}_j$ — отмеченная функция.

Очевидно, что после выполнения оператора A_i может следовать лишь один из операторов A_1 или A_2, или ..., поэтому множество функций α_{ij} должно обладать двумя свойствами.

1. Ортогональность $\alpha_{ij}\alpha_{ik} \equiv 0, (j, k = 1, ..., n)$;

2. Полнота $\vee(j = 1, ..., n)[\alpha_{ij}] = 1$.

Полное ортогональное множество функций называется «нормальным». Ортогональным будет также и множество отличных функций $\{\alpha_{ij}\mathrm{A}_j\}$. Выражение рассмотренного вида называется формулой перехода для оператора A_i.

Конструктивный способ построения формул перехода заключается в следующем. Необходимо выписать из заданной ЛСА всевозможные последовательности логических переменных (конъюнкций), которые следуют за оператором A_i, после чего построить дизъюнкцию отмеченных логических функций.

В общем виде сказанное может быть представлено как система переходов $S1$.

$$\begin{cases} A_1 \rightharpoonup \vee(j=1,...,n)[\alpha_{1j}A_j] \\ ... \\ A_i \rightharpoonup \vee(j=1,...,n)[\alpha_{ij}A_j] \\ ... \\ A_n \rightharpoonup \vee(j=1,...,n)[\alpha_{nj}A_j] \end{cases}$$

Определим процесс выполнения системы формул перехода $S1$ для последовательности наборов Δ.

- Задать значения переменных Δ_{i1} и определить значение функции $\alpha_{1j}(j=1,...,n)$ в формуле перехода для оператора A_1; если тождественно $\alpha_{1j}(\Delta_{i1})=1$, то выполняется переход к оператору A_j.
- Задать значения переменных Δ_{ij} и определить значение $\alpha_{ij}=1$, при этом найдя оператор, к которому выполняется очередной переход.
- Процесс продолжается до тех пор, пока не выполнится оператор, символизирующий окончание алгоритма.

В результате будет получена строка $\mathrm{A}_{i1}\mathrm{A}_{i2}\mathrm{A}_{i3}...\mathrm{A}_{ik}\mathrm{A}_{i(k+1)}...$, являющаяся значением системы формул переходов для последовательности наборов Δ. Аналогичным образом назовем последовательность наборов вида Δ допустимой, если значение $\Delta_{i(k+1)}$ совпадает со значением Δ_{ik} или отличается от него только теми переменными, которые входят в подмножество, изменяемое оператором A_{ik} во время его выполнения.

Две формулы перехода называются «равносильными», если их значения на данном наборе из любой допустимой последовательности совпадают. Назовем функцию α_{ij} приведенной по переменной p_s, если она может быть представлена как $\alpha_{ij}=p_s(\alpha'_{ij})\vee\neg p_s(\alpha''_{ij})$.

Это свойство распространяется также и на отмеченные функции.

Поясним вышесказанное на примере.

$$A_1 \rightharpoonup p_2p_3A_1 \vee p_1\neg p_2A_2 \vee \neg p_1\neg p_3A_3 \vee (p_1p_2\neg p_3 \vee \neg p_1\neg p_2p_3)A_4$$

Пусть требуется реализовать операцию приведения по переменной p_1. Первый член дизъюнкции не содержит переменной p_1, но если умножить

его на $1 \equiv p_1 \vee \neg p_1$, то получим $(p_1 \vee \neg p_1)p_2p_3A_1 = p_1p_2p_3A_1 \vee \neg p_1p_2p_3A_1$. Затем вынесем за скобки p_1 и $\neg p_1$, что дает

$$p_1(p_2p_3A_1 \vee \neg p_2A_2 \vee p_2\neg p_3A_4) \vee \neg p_1(p_2p_3A_1 \vee \neg p_3A_3 \vee \neg p_2p_3A_4).$$

Таким же образом можно выполнить операции приведения по переменной p_2, p_3. Например, для рассматриваемого примера получим

$$A_1 \rightharpoonup p_1[p_2(p_3A_1 \vee \neg p_3A_4) \vee \neg p_2A_2] \vee \neg p_1[p_3(p_2A_1 \vee \neg p_2A_4)\neg p_3A_3].$$

В соответствии с введенной В.Ф. Дьяченко терминологией назовем данное выражение «скобочной» формулой перехода, а все формулы перехода U — «системой скобочных формул перехода» $S2$. При получении системы скобочных формул полезно использовать некоторые очевидные равносильные соотношения.

$$\begin{cases} p_iA_j \vee p_iA_j = p_iA_j \\ p_iA_j \vee \neg p_iA_j = A_j \\ p_iA_j \vee p_ip_sA_j = p_iA_j \\ A_j \vee p_iA_j = A_j \end{cases}$$

При этом порядок вынесения переменных за скобки не задается, а выбирается из соображений целесообразности.

Таким образом, любые ЛСА могут быть представлены системой формул перехода $S1$, а также системой скобочных формул перехода $S2$. Однако, существует и третья форма — система схемных формул перехода $S3$, получаемая из $S2$ введением в алфавит ЛСА символа разделительного знака «$*$». Для рассматриваемого примера формулы перехода будут иметь следующий вид

$$A_1 \rightharpoonup p_1 \uparrow^1 p_2 \uparrow^2 p_3 \uparrow^3 A_1* \downarrow^3 A_4* \downarrow^2 A_2* \downarrow^1 A_1 \\ \neg p_3 \uparrow^4 A_3* \downarrow^4 p_2 \uparrow^5 A_1* \downarrow^5 A_4.$$

Условимся называть символы между «$*$» «выражениями», а выражение, стоящее сразу после $\rightharpoonup$ — «начальным выражением».

Заметим, что стрелки $\uparrow^i$ и их концы $\downarrow^i$ в схемной формуле перехода (как и в ЛСА) следует считать знаками операторной связи, а знаки дизъюнкции $\vee$, конъюнкции $\wedge$, отрицания $\neg$ — логической связи. Переход от скобочной к равносильной ей схемной формуле перехода заключается в замене логических связей между элементарными выражениями операторными.

Предположим, дана формула перехода $A_i \rightharpoonup p_{s1}(Q_1) \vee \neg p_{s1}(Q_2)$.

Нетрудно показать, что для нее будут равносильны следующие две схемные формулы.

$$\begin{cases} A_i \rightharpoonup p_{s1} \uparrow^1 Q_1 * \downarrow^1 Q_2 \\ A_i \rightharpoonup \neg p_{s1} \uparrow^1 Q_2 * \downarrow^1 Q_1 \end{cases}$$

Составные выражения Q_1 и Q_2 в свою очередь могут быть представлены как

$$\begin{cases} Q_1 = p_{s2} Q_1' \vee \neg p_{s2} Q_1'' \\ Q_2 = p_{s3} Q_2' \vee \neg p_{s3} Q_2'' \end{cases}$$

Тогда формула перехода может быть записана следующим образом

$$A_i \rightharpoonup p_{s1}(p_{s2} Q_1' \vee \neg p_{s2} Q_1'') \vee \neg p_{s1}(p_{s3} Q_2' \vee \neg p_{s3} Q_2'').$$

Пары переменных p_{s1} и $\neg p_{s1}$, p_{s2} и $\neg p_{s2}$, p_{s3} и $\neg p_{s3}$ называются «связными».

Следовательно, для перехода от системы $S2$ к системе $S3$ следует воспользоваться следующим набором правил.

1. Опустить все скобки формул перехода системы $S2$.
2. Заменить знаки «$\vee$» равносильными знаками «*».
3. Заменить все пары связанных переменных p_s и $\neg p_s$ парами $p_s \uparrow^i$ и $\downarrow^i$.

Порядок выполнения правил не играет роли.

Полученная система $S3$ будет равносильна $S2$, а число формул не изменится.

Определение. *Две системы $S3'$ и $S3''$ называются равносильными, если для любой последовательности наборов логических операторов их значения совпадают.*

4.1.4 Тождественные преобразования схемных формул перехода

Определение. *Оператор или часть схемной формулы перехода, которая соответствует заключенной в скобки части скобочной формулы перехода, назовем «подформулой».*

Другими словами, подформулой является такой набор выражений, где для каждого начала стрелки с индексом $\uparrow^i$ найдется конец $\downarrow^i$ с тем же индексом. Например.

$$A_i \rightharpoonup p_1(p_2A_1 \vee \neg p_2A_2) \vee \neg p_1[p_3A_3 \vee \neg p_3(p_2A_1 \vee \neg p_2A_2)]$$
$$A_i \rightharpoonup p_1 \uparrow^1 p_2 \uparrow^2 A_1* \downarrow^2 A_2* \downarrow^1 p_3 \uparrow^3 A_3* \downarrow^3 p_2 \uparrow^4 A_1* \downarrow^4 A_2$$

Тогда подформулами будут являться следующие последовательности символов.

- $N_1 = p_2 \uparrow^2 A_1* \downarrow^2 A_2$
- $N_2 = p_3 \uparrow^3 A_3* \downarrow^3 p_2 \uparrow^4 A_1* \downarrow^4 A_2$
- $N_3 = A_1$
- $N_4 = A_2$
- $N_5 = A_3$

Можно заметить, что N_1 входит в N_2, а N_3, N_4, N_5 входят в N_1 и N_2. Теоретически, всю формулу перехода можно считать подформулой. Ясно, что выделение подформул из формулы перехода является неоднозначным процессом. Например, схемная формула перехода может быть записана как $A_i \rightharpoonup p1 \uparrow^1 N_1* \downarrow^1 N_2$ или $A_i \rightharpoonup p_1 \uparrow^1 N_1* \downarrow^1 p_3 \uparrow^2 N_5* \downarrow^2 N_1$, или же и вовсе в общем виде $A_i \rightharpoonup R_1 \uparrow^1 N_1 * ... * R_i \downarrow^i N_i * W$, где N_i — подформулы, R_i — остаток выражения, не вошедший в подформулу, W — выражение, не вошедшее в подформулы и остатки.

Определение. *Две подформулы N_1 и N_2 называются «равносильными», если на заданном наборе Δ_i логических переменных, входящих в полформулы, выполняется один и тот же оператор A_i (то есть их значения совпадают).*

Рассмотрим систему тождественных преобразований схемных формул перехода $S3$, получившую название системы Лазарева-Дьяченко.

1. (a) $(A_i \rightharpoonup N) = (A_i \rightharpoonup 1 \uparrow^1 N \downarrow^1)$

   (b) $(A_i \rightharpoonup N) = (A_i \rightharpoonup \downarrow^1 N\ 1 \uparrow^1)$

2. (a) $(A_i \rightharpoonup R_{i1} \downarrow^{s1} N * R_{i2} \downarrow^{s2} N * W) = (A_i \rightharpoonup R_{i1} \downarrow^i \downarrow^{s1} \downarrow^{s2} N * R_{i2}\omega \uparrow^i N * W) = (A_i \rightharpoonup R_{i1}\omega \uparrow^i * R_{i2} \downarrow^i \downarrow^{s1} \downarrow^{s2} N * W)$

   (b) $(A_i \rightharpoonup P_1 p_i \uparrow^i P_2 A_m * \downarrow^i P_3 A_l) = (A_i \rightharpoonup P_1 \neg p_i \uparrow^i P_3 A_l * \downarrow^i P_2 A_m)$

   При этом $\omega \uparrow^i$ — безусловный переход, P_1, P_2, P_3 — некоторые (возможно, пустые) последовательности логических условий $p_{s1} \uparrow^1 p_{s2} \uparrow^2 p_{s3} \uparrow^3 ... p_{sn} \uparrow^n$

3. $$\begin{cases} A_{i1} \rightharpoonup W_{i1} * \downarrow^{s1} N \\ ... \\ A_{ik} \rightharpoonup W_{ik} * R \downarrow^{s2} N \end{cases} = \begin{cases} A_{i1} \rightharpoonup W_{i1} \\ ... \\ A_{ik} \rightharpoonup W_{ik} * R \downarrow^{s1} \downarrow^{s2} N \end{cases}$$

4. $(A_i \rightharpoonup P_1 p_2 \uparrow^i P_2 N_k * \downarrow^i p_2 \uparrow^j N_q * \downarrow^j P_3 N_2 * W) = (A_i \rightharpoonup P_1 p_2 \uparrow^i P_2 N_k * \downarrow^i P_3 N_2 * W)$

   N_q — любая подформула

5. $(A_i \rightharpoonup R_p \uparrow^i N * \downarrow^i \downarrow^s \lambda * W) = (A_i \rightharpoonup R \downarrow^i \downarrow^s p \uparrow^i N * W)$

   λ — пустая подстановка

6. $\downarrow^i \downarrow^s = \downarrow^s \downarrow^i$

7. $p \uparrow^i ... \downarrow^i = p \uparrow^s ... \downarrow^s$

8. $(\alpha \uparrow^i \prec \beta(p_1, ..., p_n)) \to (\alpha \uparrow^i = (\alpha \vee \neg\beta) \uparrow^i)$

9. $(N_1 = N_2 * N_3) \to (RN_1 = RN_2 * N_3)$

10. $(N_n = N_m; \Phi'(N_n) = \Phi'') \to (\Phi'(N_m) = \Phi'')$

Все указанные соотношения принято называть системой $C2$ и вместе с системой $C1$ они составляют полную систему тождественных преобразований схемных формул. Таким образом, для любой ЛСА U_1 через конечное число применений $\{C1, C2\}$ можно получить равносильную $U_2 = U_1$, оптимальную по некоторым параметрам (например, по числу ЛУ).

4.1.5 Алгоритм перехода от системы $S3$ к ЛСА

После цепочки преобразований ЛСА $U1 \to S1 \to S2 \to S3$ предоставляется возможность произвести оптимизацию схемных формул перехода при помощи полной системы тождественных преобразований $\{C1, C2\}$. Затем возникает проблема обратного перехода от $S3$ к ЛСА $U2 = U1$.

Алгоритм перехода изображен на рис. 11. Необходимо оговориться, что представленный алгоритм основывается не предположении, что каждый из операторов A_i встречается в системе $S3$ лишь однажды.

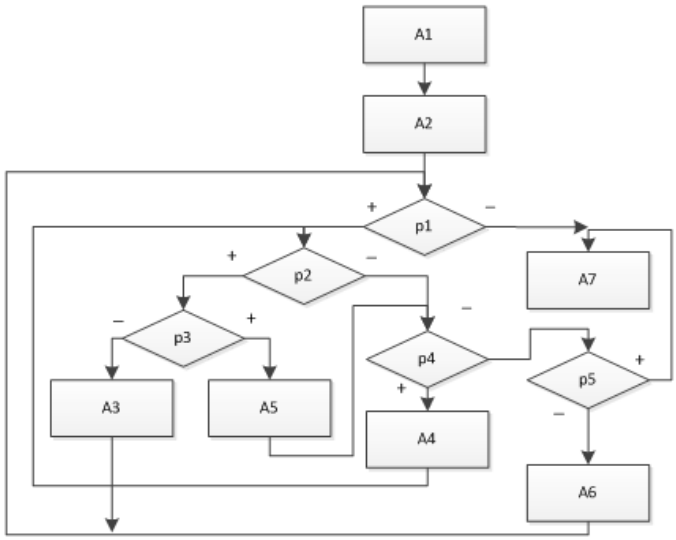

Рис. 11 — ГСА

Значение операторов.

- $A1$ — выписать оператор $A0$ вначале строки.
- $A2$ — выписать справа от $A0$ начальное выражение из схемной формулы $A0$, вычеркнув его.
- $A3$ — выписать начальное выражение из схемной формулы $A_i (i = 1, 2, ...)$,вычеркнув его.
- $A4$ — выписать любое не вычеркнутое не начальное выражение, если оно есть.
- $A5$ — заменить в выписанном выражении оператор A_k со всеми концами стрелок слева от него на $\omega \uparrow^s$ (s — целое положительное число); конец $\downarrow^s$ и оператор A_k переместить в конец строки ЛСА.

- $A6$ — выписать любой еще не встречавшийся в строке ЛСА оператор, в схемной формуле которого не вычеркнуто начальное выражение.
- A7 — останов.

Значение логических переменных.

- $p1$ — 1: в системе $S3$ есть невычеркнутые выражения; 0: все выражения вычеркнуты.
- $p2$ — 1: строка оканчивается оператором A_i; 0: строка оканчивается выражением $\omega \uparrow^i$.
- $p3$ — 1: строка оканчивается оператором A_k; 0: строка оканчивается выражением A_i.
- $p4$ — 1: в системе $S3$ есть невычеркнутые неначальные выражения; 0: все неначальные выражения вычеркнуты.
- $p5$ — 1: в строке имеются все операторы ЛСА; 0: есть неиспользуемые операторы.

4.1.6 Пример оптимизации ЛСА

$$U_1 = A_0 \downarrow^4 A_1 p_1 \uparrow^1 \downarrow^3 \downarrow^9 A_2 \downarrow^2 p_3 \uparrow^5 \omega \uparrow^4 \downarrow^1 p_2 \uparrow^2 \omega \uparrow^{10} \downarrow^5 \\ \neg p_4 \uparrow^6 \omega \uparrow^3 \downarrow^7 \neg p_4 \uparrow^8 \omega \uparrow^9 \downarrow^6 \downarrow^8 A_3 \neg p_1 \uparrow^7 \downarrow^{10} A_4 A_k$$

Построим систему $S1$.

$$\begin{cases} A_0 \rightharpoonup A_1 \\ A_1 \rightharpoonup p_1 A_2 \vee \neg p_1 p_2 A_4 \vee \neg p_1 \neg p_2 p_3 A_1 \vee \neg p_1 \neg p_2 \neg p_3 \neg p_4 A_2 \vee \\ \vee \neg p_1 \neg p_2 \neg p_3 p_4 A_3 \\ A_2 \rightharpoonup p_3 A_1 \vee \neg p_3 \neg p_4 A_2 \vee \neg p_3 p_4 A_3 \\ A_3 \rightharpoonup \neg p_1 A_4 \vee p_1 \neg p_4 A_2 \vee p_1 p_4 A_3 \\ A_4 \rightharpoonup A_k \end{cases}$$

Построим систему $S2$.

$$\begin{cases} A_0 \rightharpoonup A_1 \\ A_1 \rightharpoonup p_1 A_2 \vee \neg p_1 \{p_2 A_4 \vee \neg p_2 [p_3 A_1 \vee \neg p_3 (\neg p_4 A_2 \vee p_4 A_3)]\} \\ A_2 \rightharpoonup p_3 A_1 \vee \neg p_3 (\neg p_4 A_2 \vee p_4 A_3) \\ A_3 \rightharpoonup \neg p_1 A_4 \vee p_1 (\neg p_4 A_2 \vee p_4 A_3) \\ A_4 \rightharpoonup A_k \end{cases}$$

Построим систему $S3$.

$$\begin{cases} A_0 \rightharpoonup A_1 \\ A_1 \rightharpoonup p_1 \uparrow^1 A_2* \downarrow^1 p_2 \uparrow^2 A_4* \downarrow^2 p_3 \uparrow^3 A_1* \downarrow^3 \neg p_4 \uparrow^4 A_2* \downarrow^4 A_3 \\ A_2 \rightharpoonup p_3 \uparrow^5 A_1* \downarrow^5 \neg p_4 \uparrow^6 A_2* \downarrow^6 A_3 \\ A_3 \rightharpoonup \neg p_1 \uparrow^7 A_4* \downarrow^7 \neg p_4 \uparrow^8 A_2* \downarrow^8 A_3 \\ A_4 \rightharpoonup A_k \end{cases}$$

Анализирую формулы перехода A_1, A_2, A_3, заметим, что они содержат совпадающие (равносильные) подформулы (за исключением индексов стрелок).

Однако, правило 7 позволяет заменять индексы произвольным образом. Произведем замену индексов.

$$A_2 \rightharpoonup p_3 \uparrow^3 A_1* \downarrow^3 \neg p_4 \uparrow^4 A_2* \downarrow^4 A_3$$

$$A_3 \rightharpoonup \neg p_1 \uparrow^7 A_4* \downarrow^7 \neg p_4 \uparrow^4 A_2* \downarrow^4 A_3$$

Затем перенесем совпадающие подформулы в формулу перехода A_1.

$$\begin{cases} A_0 \rightharpoonup A_1 \\ A_1 \rightharpoonup p_1 \uparrow^1 A_2* \downarrow^1 p_2 \uparrow^2 A_4* \downarrow^2 \downarrow^9 p_3 \uparrow^3 A_1* \downarrow^7 \downarrow^3 \neg p_4 \uparrow^4 A_2* \downarrow^4 A_3 \\ A_2 \rightharpoonup \omega \uparrow^9 \\ A_3 \rightharpoonup \neg p_1 \uparrow^7 A_4 \\ A_4 \rightharpoonup A_k \end{cases}$$

После этого применим к формуле перехода A_1 правило 2, то есть проинвертируем пары связных переменных. Перенесем операторы A_4 в формулу A_3 и A_1 — в A_0.

$$\begin{cases} A_0 \rightharpoonup\downarrow^3 A_1 \\ A_1 \rightharpoonup \neg p_1 \uparrow^1 \neg p_2 \uparrow^2\downarrow^9 \neg p_3 \uparrow^3\downarrow^7 \neg p_4 \uparrow^4\downarrow^1 A_2* \downarrow^4 A_3 \\ A_2 \rightharpoonup \omega \uparrow^9 \\ A_3 \rightharpoonup \neg p_1 \uparrow^7\downarrow^2 A_4 \\ A_4 \rightharpoonup A_k \end{cases}$$

Воспользуемся вышеизложенным алгоритмом возврата от системы $S3$ к ЛСА.

$$U_2 = A_0 \downarrow^3 A_1\neg p_1 \uparrow^1 \neg p_2 \uparrow^2\downarrow^9 \neg p_3 \uparrow^3\downarrow^7 \neg p_4 \uparrow^4\downarrow^1 A_2\omega \uparrow^9\downarrow^4 A_3\neg p_1 \uparrow^7\downarrow^2 A_4A_k$$

Полученная ЛСА U_2 равносильна U_1, но имеет меньшее число элементарных выражений $(U1/U2)$.

- Число операторов — 6/6;
- Число логических условий — 6/5;
- Число тождественно ложных переходов ω — 4/1.

Очевидно, что если число операторов в исходной ЛСА было минимально, то оно останется таким же. Сокращение числа ЛУ и безусловных переходов ω зависит как от наличия равносильных подформул, так и от способа вынесения за скобки логических переменных. Следовательно, порядок вынесения за скобки переменных должен быть таким, чтобы увеличить число равносильных подформул, которые можно объединить.

В качестве заключения следует выделить некоторые практические рекомендации вынесения за скобки логических переменных в формулах перехода.

1. Начинать вынесение за скобки следует с той переменной, по которой приведено множество отмеченных элементарных конъюнкций.

2. Если множество приведено по нескольким логическим переменным, то можно начинать с любой переменной.

3. Если указанное множество отмечено по всем логическим переменным, то порядок безразличен при условии, что в формуле нет пары или более отмеченных элементарных конъюнкций с одинаковыми операторами (в противном случае будет получено увеличение числа ЛУ).

4.2 Граф-схемы алгоритмов

Как уже упоминалось ранее, граф-схемы алгоритмов были предложены Л.А. Калужниным. Наиболее полное описание данного аппарата содержится в работе «Об алгоритмизации математических задач». Введем формальное определение ГСА и связанных с ними понятий.

ГСА — конечный связный граф G, удовлетворяющий следующим условиям.

- В каждом графе имеется два отмеченных узла — входной, из которого выходит лишь одна стрелка, и выходной, из которого не выходит ни одной стрелки.

- Из любого другого узла выходит либо одна (γ-узел), либо две (β-узел) стрелки.

- Стрелки, исходящие из β-узла имеют отметки «+» и «-».

- Имеются конечные множества функциональных элементов — множество преобразователей $Q = \{Q_i\}$ и множество распознавателей $R = \{R_j\}$, $i = 1...n, j = 1...m$.

- Каждому γ-узлу однозначно сопоставлен преобразователь Q_i, а β-узлу — распознаватель R_j, при этом некоторые преобразователи и распознаватели могут быть сопоставлены нескольким различным узлам.

Пример ГСА изображен на рис. 12.

Назовем подграф-схемой H_i ГСА Г некоторый связный подграф G' графа G, удовлетворяющий следующим условиям.

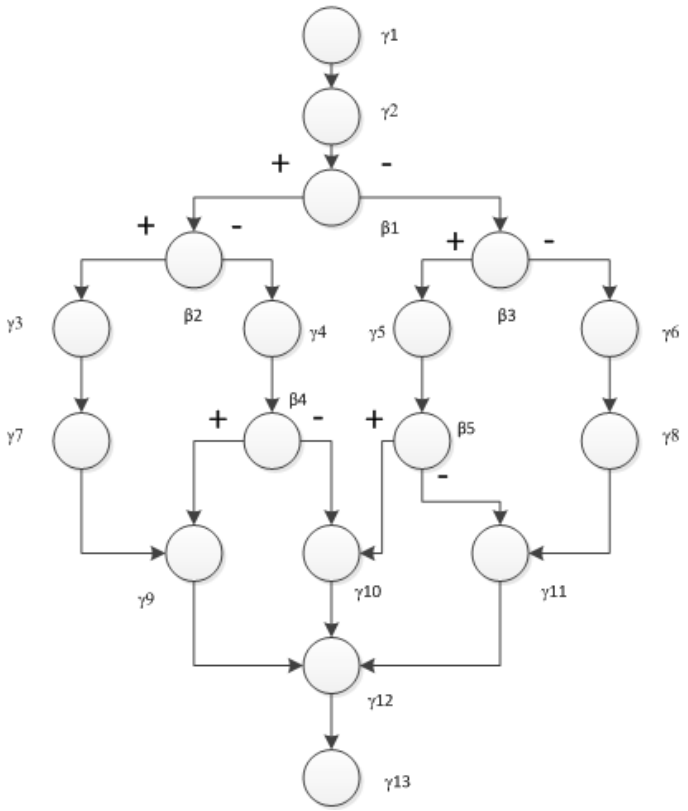

Рис. 12 — Граф-схема алгоритма

- Каждый подграф G' содержит один входной γ-узел, один или более выходных γ-узлов и β-узлы.
- Будем предполагать, что к подграфу относятся и стрелки выходных γ-узлов.
- Узлам подграфа G' сопоставлены те же функциональные элементы, что и узлам графа G.

Пример подграф-схемы представлен на рис. 13.

Если связный подграф G' не имеет ни одного контура, то он содержит «дерево», вершины которого — γ-узлы. «Дерево» имеет одну отмеченную вершину (вход), а также выходные узлы и их стрелки. В каждом дереве можно выделить «поддеревья» (если вершина не одна).

Определение. *Последовательность вида* $\updownarrow \beta_{i1}\beta_{i2}...\gamma_j$ *называется «ветвью».*

Здесь знак $\updownarrow$ означает («+») или («-»).

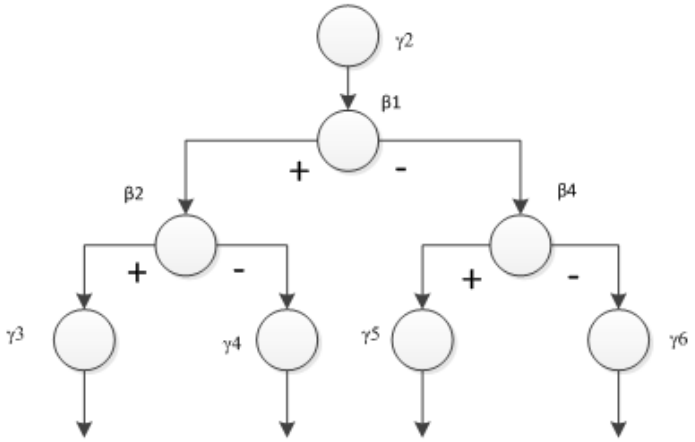

Рис. 13 — Подграф-схема ГСА

Определение. *Две ветви называются равносильными, если последовательности указного вида совпадают до порядка следования логических операторов.*

Определение. *Два дерева равносильны, если их ветви равносильны.*

Определение. *Две ГСА равносильны, если между путями от* A_0 *до* A_k *существует взаимно однозначное соответствие.*

4.3 Матричные схемы алгоритмов

Еще одним наглядным способом представления алгоритмов являются матричные схемы (МСА), зарекомендовавшие себя как удобный инструмент для анализа систем переходов.

Фактически, МСА является собой табличной формой отображения системы $S1$. Более формально.

Определение. *Матричной схемой алгоритма называется квадратная матрица, строки которой соответствуют операторам* $A_0, A_1, ..., A_n$, *а столбцы — операторам* $A_1, A_2, ..., A_n, A_k$. *При этом содержимое ячейки* $[i, j]$ *соответствует составному логическому условию, определяющему возможность перехода от выполнения оператора* A_i *к выполнению оператора* A_j.

Естественным образом, каждая строка матрицы, по аналогии с каждой формулой перехода, должна отвечать двум требованиям.

1. Ортогональность $A_{ij}A_{ik} \equiv 0, (j,k = 1,...,n)$, то есть конъюнкция любых двух элементов строки должна быть тождественно ложна;

2. Полнота $\vee(j = 1,...,n)[A_{ij}] = 1$, то есть дизъюнкция всех элементов строки должна быть тождественно истинна.

Определим процесс выполнения МСА U для последовательности наборов Δ.

- Задать значения переменных Δ_{i1} и определить значение функций $A_{1j}(j = 1,...,n,k)$ в строке 1 матрицы (оператор A_0); в связи с требованием ортогональности, найдется ровно одна обращающаяся в истину логическая функция; выполнить переход к j-ой строке матрицы (соответствующей определенному оператору A_{j-1}).

- Задать значения переменных Δ_{ij} и определить значение истинной функции $A_{ij} = 1$, при этом найдя строку (и оператор), к которому выполняется очередной переход.

- Процесс продолжается до тех пор, пока не выполнится оператор, символизирующий окончание алгоритма.

В результате будет получена строка $\mathrm{A}_{i1}\mathrm{A}_{i2}\mathrm{A}_{i3}...\mathrm{A}_{ik}\mathrm{A}_{i(k+1)}...$, являющаяся значением матричной схемы для последовательности наборов Δ.

Равносильная заданной ЛСА U_2 матричная схема U_1 может быть получена элементарным построением системы формул перехода $S1$ с последующим занесением логических функций в ячейки матрицы (с перечислением через символ дизъюнкции, если потребуется).

Для примера представим МСА рассмотренного в разделе 4.1.6 алгоритма.

$$\begin{array}{c} \\ A_0 \\ A_1 \\ A_2 \\ A_3 \\ A_4 \end{array} \begin{array}{c} \begin{array}{ccccc} A_1 & A_2 & A_3 & A_4 & A_k \end{array} \\ \begin{pmatrix} 1 & - & - & - & - \\ \neg p_1\neg p_2 p_3 & p_1 \vee \neg p_1\neg p_2\neg p_3\neg p_4 & \neg p_1\neg p_2\neg p_3 p_4 & \neg p_1 p_2 & - \\ p_3 & \neg p_3\neg p_4 & \neg p_3 p_4 & - & - \\ - & p_1\neg p_4 & p_1 p_4 & \neg p_1 & - \\ - & - & - & - & 1 \end{pmatrix} \end{array}$$

4.4 Объединение схем алгоритмов

4.4.1 Метод объединения схем алгоритмов

Нередко возникает необходимость в реализации устройства, поддерживающего выполнение сразу нескольких алгоритмов. Классическим примером подобной задачи является разработка арифметико-логического устройства (АЛУ) процессора ЭВМ. АЛУ должно иметь возможность осуществлять как стандартные арифметические (сложение, вычитание, умножение, деление, возведение в степень и так далее), так и элементарные логические операции (например, сравнение чисел). Каждой из перечисленных операций соответствует своя последовательность действий, а все устройство в целом должно обеспечивать выполнение каждого из алгоритмов.

В частном случае, объединение может быть произведено путем элементарного выбора конкретного алгоритма перед началом работы: «Если требуется осуществить операцию O_1, то следует использовать алгоритм U_1, если же необходимо осуществить операцию O_2, то следует использовать алгоритм U_2, и так далее». Однако в более общем случае, такой тривиальный подход приведет к возрастанию накладных расходов(в примере с АЛУ — к увеличению аппаратных затрат) в следствие многократной реализации общих для нескольких алгоритмов фрагментов. Следовательно, для практического применения больше подойдет техника объединения, способная учитывать повторяющиеся последовательности действий.

Более формально, пусть имеются алгоритмы $U_1, U_2, ..., U_l$, заданные в форме ЛСА, и требуется получить некоторый минимальный по некоторому критерию алгоритм U, который, при определенных дополнительных условиях, мог бы быть равносилен любому из $U_1, U_2, ..., U_l$.

В роли дополнительных могут использоваться специальные логические условия $r_1, r_2, ..., r_k$, применяемые в схеме U наряду с $p_1, p_2, ..., p_m$ таким образом, чтобы каждая из возможных их конъюнкций $R_1, R_2, ..., R_{2^k}$ соответствовала не более чем одному алгоритму из $U_1, U_2, ..., U_l$.

Определение. *Конъюнкции $R_1, R_2, ..., R_{2^k}$, сформированные из дополнительных условий $r_1, r_2, ..., r_k$, сопоставленные алгоритмам $U_1, U_2, ..., U_l$, называются определяющими конъюнкциями.*

Очевидным образом, оптимальным значением k будет такое минимальное, что $2^k > l$.

Объединенной ЛСА U называется такая ЛСА, которая соответствует двум условиям.

1. Любой оператор A_i, входяший хотя бы в одну из объединяемых ЛСА $U_1, U_2, ..., U_l$, входит ровно один раз в ЛСА U.

2. Если в ЛСА $U(r_1, r_2, ..., r_k, p_1, p_2, ..., p_m)$ подставить значения определяющей конъюнкции R_i, то U трансформируется в равносильный, соответствующий R_i, алгоритм.

Кроме того, введем понятие определяющей функции β_j^i, представляющей собой условное выражение

$$\beta_j^i = R_i \vee \frac{R'_1}{0} \vee \frac{R'_2}{0} \vee ... \vee \frac{R'_q}{0}.$$

При этом будем считать, что определяющая функция указанного вида соответствует оператору A_j из U_i, R_i задает выполнение U_i, а $R'_1, R'_2, ..., R'_q$ задают выполнение алгоритмов, в которых оператор A_j или отсутствует, или имеет в точности такую же строку переходов в эквивалентной МСА.

Принимая во внимание все вышеизложенное, задача объединения алгоритмов может быть решена следующим образом.

1. Построить равносильные ЛСА $U_1, U_2, ..., U_l$ МСА.

2. Построить объединенную МСА, в которой каждый элемент α_{ij} представим как $\alpha_{ij} = \vee(k = 1, ..., l)[\alpha_{ij}^k \beta_i^k]$, где α_{ij}^k — элемент α_{ij} соответствующей U_k МСА, а β_i^k — определяющая функция для оператора A_i из U_k.

3. Построить недоопределенную систему формул перехода $S1$.

4. Построить систему скобочных формул перехода $S2$, доопределив ее.

5. Построить систему схемных формул перехода $S3$.

6. Минимизировать $S3$ по заданному критерию посредством полной системы тождественных преобразований схемных формул.

7. Преобразовать $S3$ в эквивалентную ЛСА U.

Итогом выполнения перечисленных действий будет получение логической схемы объединенного алгоритма U.

4.4.2 Пример объединения схем алгоритмов

Рассмотрим подробнее предложенный метод на примере. Пусть требуется произвести объединение следующих алгоритмов.

$$U_1 = A_0 \downarrow^3 A_1 p_1 \uparrow^1 p_2 \uparrow^1 \downarrow^4 A_2 \omega \uparrow^2 \downarrow^1 A_3 \neg p_3 \uparrow^3 \omega \uparrow^4 \downarrow^2 A_k$$
$$U_2 = A_0 \neg p_1 \uparrow^1 \downarrow^5 A_4 \neg p_4 \uparrow^1 A_5 \neg p_2 \uparrow^2 p_4 \uparrow^1 \omega \uparrow^5 \downarrow^2 \neg p_4 \uparrow^4 A_6 \downarrow^1 A_2 \downarrow^4 A_k$$
$$U_3 = A_0 A_1 \neg p_1 \uparrow^1 A_4 \neg p_4 \uparrow^2 \downarrow^4 A_6 \omega \uparrow^3 \downarrow^1 p_2 \uparrow^4 \downarrow^2 A_2 \downarrow^3 A_k$$

Построим эквивалентные каждому из представленных алгоритмов матричные схемы.

$$\begin{array}{c} \\ A_0 \\ A_1 \\ A_2 \\ A_3 \end{array} \begin{array}{c} \begin{array}{cccc} A_1 & A_2 & A_3 & A_k \end{array} \\ \begin{pmatrix} 1 & - & - & - \\ - & p_1 p_2 & \neg p_1 \vee p_1 \neg p_2 & - \\ - & - & - & 1 \\ p_3 & \neg p_3 & - & - \end{pmatrix} \end{array}$$

$$\begin{array}{c} \\ A_0 \\ A_2 \\ A_4 \\ A_5 \\ A_6 \end{array} \begin{array}{c} \begin{array}{ccccc} A_2 & A_4 & A_5 & A_6 & A_k \end{array} \\ \begin{pmatrix} p_1 & \neg p_1 & - & - & - \\ - & - & - & - & 1 \\ p_4 & - & \neg p_4 & - & - \\ \neg p_2 \neg p_4 & \neg p_2 p_4 & - & p_2 \neg p_4 & p_2 p_4 \\ 1 & - & - & - & - \end{pmatrix} \end{array}$$

$$\begin{array}{c} \\ A_0 \\ A_1 \\ A_2 \\ A_4 \\ A_6 \end{array} \begin{array}{c} \begin{array}{ccccc} A_1 & A_2 & A_4 & A_6 & A_k \end{array} \\ \begin{pmatrix} 1 & - & - & - & - \\ - & p_1 p_2 & \neg p_1 & p_1 \neg p_2 & - \\ - & - & - & - & 1 \\ - & p_4 & - & \neg p_4 & - \\ - & - & - & - & 1 \end{pmatrix} \end{array}$$

Очевидно, что для выполнения объединения понадобится два дополнительных логических условия — r_1 и r_2. Сформируем определяющие конъюнкции как $R_1 = r_1 r_2$, $R_2 = r_1 \neg r_2$ и $R_3 = \neg r_1 r_2$. Кроме того, одна из возможных комбинаций, $R = \neg r_1 \neg r_2$, остается неиспользованной. Будем считать, что она соответствует «пустой схеме», не содержащей ни одного оператора.

Тогда семейство определяющих функций будет иметь следующий вид.

$$\beta_0^1 = r_1 r_2 \vee \frac{\neg r_1 r_2}{0} = \frac{r_1}{1} r_2$$

$$\beta_1^1 = r_1 r_2 \vee \frac{r_1 \neg r_2}{0} = r_1 \frac{r_2}{1}$$

$$\beta_2^1 = r_1 r_2 \vee \frac{r_1 \neg r_2}{0} \vee \frac{\neg r_1 r_2}{0} \vee \frac{\neg r_1 \neg r_2}{0} = 1$$

$$\beta_3^1 = r_1 r_2 \vee \frac{r_1 \neg r_2}{0} \vee \frac{\neg r_1 r_2}{0} \vee \frac{\neg r_1 \neg r_2}{0} = 1$$

$$\beta_0^2 = r_1 \neg r_2 \vee \frac{\neg r_1 \neg r_2}{0} = \frac{r_1}{1} \neg r_2$$

$$\beta_2^2 = r_1 \neg r_2 \vee \frac{r_1 r_2}{0} \vee \frac{\neg r_1 r_2}{0} \vee \frac{\neg r_1 \neg r_2}{0} = 1$$

$$\beta_4^2 = r_1 \neg r_2 \vee \frac{r_1 r_2}{0} \vee \frac{\neg r_1 \neg r_2}{0} = \frac{r_1 \dfrac{\neg r_2}{1}}{\dfrac{r_1}{1} \neg r_2}$$

$$\beta_5^2 = r_1 \neg r_2 \vee \frac{r_1 r_2}{0} \vee \frac{\neg r_1 r_2}{0} \vee \frac{\neg r_1 \neg r_2}{0} = 1$$

$$\beta_6^2 = r_1 \neg r_2 \vee \frac{r_1 r_2}{0} \vee \frac{\neg r_1 \neg r_2}{0} = \frac{r_1 \dfrac{\neg r_2}{1}}{\dfrac{r_1}{1} \neg r_2}$$

$$\beta_0^3 = \neg r_1 r_2 \vee \frac{r_1 r_2}{0} \vee \frac{\neg r_1 \neg r_2}{0} = \frac{\dfrac{\neg r_1}{1} r_2}{\neg r_1 \dfrac{r_2}{1}}$$

$$\beta_1^3 = \neg r_1 r_2 \vee \frac{\neg r_1 \neg r_2}{0} = \neg r_1 \frac{r_2}{1}$$

$$\beta_2^3 = \neg r_1 r_2 \vee \frac{r_1 r_2}{0} \vee \frac{r_1 \neg r_2}{0} \vee \frac{\neg r_1 \neg r_2}{0} = 1$$

$$\beta_4^3 = \neg r_1 r_2 \vee \frac{r_1 r_2}{0} \vee \frac{\neg r_1 \neg r_2}{0} = \frac{\dfrac{\neg r_1}{1} r_2}{\neg r_1 \dfrac{r_2}{1}}$$

$$\beta_6^3 = \neg r_1 r_2 \vee \frac{r_1 r_2}{0} \vee \frac{\neg r_1 \neg r_2}{0} = \frac{\dfrac{\neg r_1}{1} r_2}{\neg r_1 \dfrac{r_2}{1}}$$

Объединенная матричная схема примет вид.

$$
\begin{array}{c}
\\
A_0 \\
A_1 \\
A_2 \\
A_3 \\
A_4 \\
A_5 \\
A_6
\end{array}
\begin{array}{c}
\begin{array}{ccccccc}
A_1 & A_2 & A_3 & A_4 & A_5 & A_6 & A_k
\end{array} \\
\left(
\begin{array}{ccccccc}
\frac{r_1}{1}r_2 \vee \frac{\frac{\neg r_1}{1}r_2}{\neg r_1\frac{r_2}{1}} & \frac{r_1}{1}\neg r_2 p_1 & - & \frac{r_1}{1}\neg r_2\neg p_1 & - & - & - \\
- & (r_1\frac{r_2}{1} \vee \neg r_1\frac{r_2}{1})p_1p_2 & r_1\frac{r_2}{1}(\neg p_1 \vee p_1\neg p_2) & \neg r_1\frac{r_2}{1}\neg p_1 & - & \neg r_1\frac{r_2}{1}p_1\neg p_2 & - \\
- & - & - & - & - & - & 1 \\
p_3 & \neg p_3 & - & - & - & - & - \\
- & (\frac{r_1\frac{\neg r_2}{1}}{\frac{r_1}{1}\neg r_2} \vee \frac{\frac{\neg r_1}{1}r_2}{\neg r_1\frac{r_2}{1}})p_4 & - & - & \frac{r_1\frac{\neg r_2}{1}}{\frac{r_1}{1}\neg r_2}\neg p_4 & \frac{\frac{\neg r_1}{1}r_2}{\neg r_1\frac{r_2}{1}}\neg p_4 & - \\
- & \neg p_2\neg p_4 & - & \neg p_2 p_4 & - & p_2\neg p_4 & \frac{p_2p_4}{\frac{\neg r_1}{1}r_2} \\
- & \frac{r_1\frac{\neg r_2}{1}}{\frac{r_1}{1}\neg r_2} & - & - & - & - & \frac{\frac{\neg r_1}{1}r_2}{\neg r_1\frac{r_2}{1}}
\end{array}
\right)
\end{array}
$$

Эквивалентная недоопределенная система формул перехода $S1$.

$$
\begin{cases}
A_0 \rightharpoonup (\frac{r_1}{1}r_2 \vee \frac{\frac{\neg r_1}{1}r_2}{\neg r_1\frac{r_2}{1}})A_1 \vee \frac{r_1}{1}\neg r_2 p_1 A_2 \vee \frac{r_1}{1}\neg r_2\neg p_1 A_4 \\
A_1 \rightharpoonup (r_1\frac{r_2}{1} \vee \neg r_1\frac{r_2}{1})p_1p_2A_2 \vee r_1\frac{r_2}{1}(\neg p_1 \vee p_1\neg p_2)A_3 \vee \neg r_1\frac{r_2}{1}\neg p_1 A_4 \vee \\
\vee \neg r_1\frac{r_2}{1}p_1\neg p_2 A_6 \\
A_2 \rightharpoonup A_k \\
A_3 \rightharpoonup p_3A_1 \vee \neg p_3 A_2 \\
A_4 \rightharpoonup (\frac{r_1\frac{\neg r_2}{1}}{\frac{r_1}{1}\neg r_2} \vee \frac{\frac{\neg r_1}{1}r_2}{\neg r_1\frac{r_2}{1}})p_4A_2 \vee \frac{r_1\frac{\neg r_2}{1}}{\frac{r_1}{1}\neg r_2}\neg p_4 A_5 \vee \frac{\frac{\neg r_1}{1}r_2}{\neg r_1\frac{r_2}{1}}\neg p_4 A_6 \\
A_5 \rightharpoonup \neg p_2\neg p_4 A_2 \vee \neg p_2 p_4 A_4 \vee p_2\neg p_4 A_6 \vee p_2p_4A_k \\
A_6 \rightharpoonup \frac{r_1\frac{\neg r_2}{1}}{\frac{r_1}{1}\neg r_2}A_2 \vee \frac{\frac{\neg r_1}{1}r_2}{\neg r_1\frac{r_2}{1}}A_k
\end{cases}
$$

Доопределение и приведение формул по логическим переменных позволит получить систему $S2$.

$$\begin{cases} A_0 \rightharpoonup r_2 A_1 \vee \neg r_2(p_1 A_2 \vee \neg p_1 A_4) \\ A_1 \rightharpoonup p_1(p_2 A_2 \vee \neg p_2(r_1 A_3 \vee \neg r_1 A_6)) \vee \neg p_1(r_1 A_3 \vee \neg r_1 A_4) \\ A_2 \rightharpoonup A_k \\ A_3 \rightharpoonup p_3 A_1 \vee \neg p_3 A_2 \\ A_4 \rightharpoonup p_4 A_2 \vee \neg p_4(r_1 A_5 \vee \neg r_1 A_6) \\ A_5 \rightharpoonup p_4(p_2 A_k \vee \neg p_2 A_4) \vee \neg p_4(p_2 A_6 \vee \neg p_2 A_2) \\ A_6 \rightharpoonup r_1 A_2 \vee \neg r_1 A_k \end{cases}$$

Построим систему $S3$.

$$\begin{cases} A_0 \rightharpoonup r_2 \uparrow^1 A_1* \downarrow^1 p_1 \uparrow^2 A_2* \downarrow^2 A_4 \\ A_1 \rightharpoonup p_1 \uparrow^3 p_2 \uparrow^4 A_2* \downarrow^4 r_1 \uparrow^5 A_3* \downarrow^5 A_6* \downarrow^3 r_1 \uparrow^6 A_3* \downarrow^6 A_4 \\ A_2 \rightharpoonup A_k \\ A_3 \rightharpoonup p_3 \uparrow^7 A_1* \downarrow^7 A_2 \\ A_4 \rightharpoonup p_4 \uparrow^8 A_2* \downarrow^8 r_1 \uparrow^9 A_5* \downarrow^9 A_6 \\ A_5 \rightharpoonup p_4 \uparrow^{10} p_2 \uparrow^{11} A_k* \downarrow^{11} A_4* \downarrow^{10} p_2 \uparrow^{12} A_6* \downarrow^{12} A_2 \\ A_6 \rightharpoonup r_1 \uparrow^{13} A_2* \downarrow^{13} A_k \end{cases}$$

На данном этапе могут быть выполнены минимизирующие действия в соответствии с полной системой тождественных преобразований схемных формул. Кроме того, следует исключить повторяющиеся операторы путем их замены на безусловные переходы.

$$\begin{cases} A_0 \rightharpoonup r_2 \uparrow^1\downarrow^4 A_1 \downarrow^1 \neg p_1 \uparrow^2\downarrow^6 A_4 \downarrow^2 A_2 \\ A_1 \rightharpoonup p_1 \uparrow^3 \neg p_2 \uparrow^2 \neg r_1 \uparrow^5\downarrow^7 A_6 \downarrow^5 A_3 \downarrow^3 \neg r_1 \uparrow^5 \omega \uparrow^6 \\ A_2 \rightharpoonup\downarrow^9 A_k \\ A_3 \rightharpoonup p_3 \uparrow^2 \omega \uparrow^4 \\ A_4 \rightharpoonup \neg p_4 \uparrow^2 r_1 \uparrow^7 A_5 \\ A_5 \rightharpoonup p_4 \uparrow^8 p_2 \uparrow^6 \omega \uparrow^9\downarrow^8 p_2 \uparrow^2 \omega \uparrow^7 \\ A_6 \rightharpoonup r_1 \uparrow^9 \omega \uparrow^2 \end{cases}$$

После чего, в соответствии с вышеизложенным алгоритмом преобразования, может быть получена логическая схема объединенного алгоритма.

$$U = A_0 r_2 \uparrow^1 \downarrow^4 A_1 p_1 \uparrow^3 \neg p_2 \uparrow^2 \neg r_1 \uparrow^5 \downarrow^7 A_6 r_1 \uparrow^9 \omega \uparrow^2 \downarrow^5 A_3 p_3 \uparrow^2 \omega \uparrow^4$$
$$\downarrow^3 \neg r_1 \uparrow^5 \omega \uparrow^6 \downarrow^1 \neg p_1 \uparrow^2 \downarrow^6 A_4 \neg p_4 \uparrow^2 r_1 \uparrow^7 A_5 p_4 \uparrow^8 p_2 \uparrow^6 \omega \uparrow^9 \downarrow^8 p_2 \uparrow^2$$
$$\omega \uparrow^7 \downarrow^2 A_2 \downarrow^9 A_k$$

Список литературы

[1] Ахо, А. Построение и анализ вычислительных алгоритмов [Текст] : монография / А. Ахо, Дж. Хопкрофт, Дж. Ульман. – Москва : Мир, 1979. – 536 с.

[2] Гэри, М. Вычислительные машины и труднорешаемые задачи [Текст] : монография / М. Гэри, Д. Джонсон. – Москва : Мир, 1982. – 416 с.

[3] Дьяченко, В. Ф. Способ упрощения логических схем алгоритмов, учитывающий неиспользуемые наборы значений переменных [Текст] / В. Ф. Дьяченко, В. Г. Лазарев // Проблемы передачи информации. – 1966. – Т. 2, № 3. – С. 92– 96.

[4] Дьяченко, В. Ф. Управление на сетях связи [Текст] / В. Ф. Дьяченко, В. Г. Лазарев, Г. Г. Саввин. – Москва : Энергия, 1970. – 224 с.

[5] Калужнин, Л. А. Об алгоритмизации математических задач [Текст] / Л. А. Калужнин // Проблемы кибернетики. – 1959. – Вып. 1. – С. 58–63.

[6] Карпов, Ю. Г. Теория автоматов [Текст] : учеб. для студентов вузов / Ю. Г. Карпов. – Санкт-Петербург : Питер, 2003. – 208 с.

[7] Катленд, Н. Вычислимость. Введение в теорию рекурсивных функций [Текст] / Н. Катленд ; пер. с англ. А. А. Мучника ; под ред. С. Ю. Маслова. – Москва : Мир, 1983. – 256 с.

[8] Кузнецов, О. П. Дискретная математика для инженера [Текст] / О. П. Кузнецов. – 3-е изд., перераб. и доп. – Санкт-Петербург : Лань, 2005. – 400 с.

[9] Лазарев, В. Г. Синтез управляющих автоматов [Текст] / В. Г. Лазарев, Е. И. Пийль. – Москва : Энергия, 1970. – 328 с.

[10] Ляпунов, А. А. О логических схемах программ [Текст] / А. А. Ляпунов // Проблемы кибернетики : сб. ст. Вып. 1. – Москва : Физматгиз, 1959. – С. 46–74.

[11] Матрос, Д. Ш. Теория алгоритмов [Текст] : учебник / Д. Ш. Матрос, Т. Б. Поднебесова. – Москва : БИНОМ : Лаборатория знании, 2008. – 202 с.

[12] Успенский, В. А. Машина Поста [Текст] / В. А. Успенский. – 2-е изд., испр. – Москва : Наука, 1988. – 96 с.

[13] Успенский, В. А. Лекции о вычислимых функциях [Текст] / В. А. Успенский. – Москва : Физматгиз, 1960. – 492 с.

[14] Хопкрофт, Дж. Введение в теорию автоматов, языков и вычислений [Текст] / Дж. Хопкрофт, Р. Мотвани, Дж. Ульман. – Москва : Вильямс, 2002. – 528 с.

[15] Эббинхауз, Г. Д. Машины Тьюринга и рекурсивные функции [Текст] / Г. Д. Эббинхауз, К. Якобс, Ф. К. Ман. – Москва : Мир, 1972. – 262 с.

[16] Янов, Ю. И. О логических схемах алгоритмов [Текст] / Ю. И. Янов // Проблемы кибернетики. Вып. 1. – Москва : Физматгиз, 1958. – С. 75–127.

Учебное издание

Чистяков Геннадий Андреевич

ОСНОВЫ ТЕОРИИ АЛГОРИТМОВ

Учебное пособие

Авторская редакция

Технический редактор А. Е. Свинина

Подписано в печать 17.10.2018. Печать цифровая. Бумага для офиснои техники.
Усл. печ. л. 5,75. Тираж 50 экз. Заказ № 5456.

Федеральное государственное бюджетное образовательное учреждение высшего образования «Вятский государственный университет».

610000, г. Киров, ул. Московская, 36, тел.: (8332) 74-25-63, http://vyatsu.ru