УЧЕБНИК

ДЛЯ ВУЗОВ

СПЕЦИАЛЬНОСТЬ

# ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ

3-е издание

Горячая линия–Телеком

Стариченко Б. Е.

Стариченко Б. Е.

# ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ

*Допущено УМО вузов по университетскому политехническому образованию в качестве учебника для студентов высших учебных заведений, обучающихся по направлению подготовки «Информационные системы и технологии»*

*3-е издание, переработанное и дополненное*

**Москва**
**Горячая линия – Телеком**
**2016**

УДК 004:621.391(075)
ББК 32.81
С77

**Стариченко Б. Е.**

**С77** Теоретические основы информатики. Учебник для вузов. – 3-е изд. перераб. и доп. – М.: Горячая линия – Телеком, 2016. – 400 с.: ил.

**ISBN 978-5-9912-0462-0.**

Рассмотрены вопросы теории информации Шеннона, теории кодирования, криптографии, элементы теории алгоритмов и теории конечных автоматов, а также общие вопросы моделирования и описания систем. Отбор материала произведен в соответствии с программой подготовки студентов высших учебных заведений, обучающихся по направлению подготовки «Информационные системы и технологии». Каждая глава содержит многочисленные примеры решения задач, а также вопросы и задания для самоконтроля.

Для студентов вузов, изучающих информатику в качестве профильной дисциплины, а также школьных учителей информатики.

**ББК 32.81**

*Адрес издательства в Интернет WWW.TECHBOOK.RU*

ISBN 978-5-9912-0462-0

# Предисловие

Любая наука начинается с определения круга рассматриваемых и решаемых в ней проблем теоретического и практического характера. Далее логика развития научного знания требует построения строгого понятийного аппарата — языка, принятого в данной науке и обеспечивающего однозначную трактовку дефиниций специалистами. В научном языке определение новых терминов возможно двумя путями: *аксиоматическим* и *операционным*. В первом случае, принятом, в частности, в математике, в виде постулатов вводится некоторый набор исходных определений, а уже через них выражаются все остальные понятия. Экспериментальные науки предложили другой тип определения — операционный, когда понятие задается описанием способа нахождения количественной меры, характеризующей это понятие. Например, когда определяют скорость как перемещение, совершаемое материальной точкой за единицу времени, то подразумевается, что имеются способы измерения перемещения и времени, а их отношение определит новую величину. Как правило, в естественнонаучных дисциплинах считается целесообразным вводить в научный язык только те величины (понятия), которые могут быть определены операционно.

После того как понятия и характеризующие их величины определены, необходимо выяснить наличие и характер связей между ними — в этом состоит основная задача любой науки. Связи эти могут иметь характер законов, закономерностей или тенденций. Использование этих законов для решения задач практики неизбежно связано еще с одним этапом научного исследования — выделением естественных или абстрактных систем и построением их моделей. Все прикладные, «технические» аспекты и решения основываются на построенном таким образом теоретическом фундаменте и, если угодно, являются его следствием.

Особенность информатики как научной и учебной дисциплины состоит в том, что прикладная ее составляющая оказывается востребованной многими людьми, в том числе весьма далекими от научной сферы. Такую ситуацию ни в коем случае не следует считать неправильной или нежелательной — напротив! Успехи информатики и технологий привели к возможности создания

уникального устройства — компьютера (и, в первую очередь, персонального), а также удобного программного обеспечения, позволяющего работать с ним в режиме *черного ящика*, т. е. решать с его помощью прикладные задачи, не вникая в реализацию механизма решения. Именно благодаря этому обстоятельству компьютер и получил столь широкое распространение в современном обществе; появился даже термин «*пользовательский уровень владения компьютером*». По-видимому, этим объясняется тот факт, что многие пособия по информатике ограничиваются лишь рассмотрением элементов устройства компьютера, его программного обеспечения и, в последнее время, работы в сети Интернет. Не умаляя важности и необходимости подобных книг, автор данного учебника поставил перед собой иную задачу — обсудить теоретические основы науки информатики (что отражено в названии), оставляя за рамками рассмотрения прикладные аспекты. При этом особое внимание автор постарался уделить терминологическому аппарату информатики, предлагая возможно более строгие и корректные определения базовых понятий и демонстрируя логику их взаимосвязи, развития и взаимообусловленности.

Данное пособие написано на основе курса лекций, прочитанного студентам 1–2-го курсов Института математики, информатики и информационных технологий Уральского государственного педагогического университета (г. Екатеринбург). Это сказалось, в первую очередь, на использовании математического аппарата — он сведен к минимально приемлемому, с точки зрения автора, уровню, когда доказательность соотношений не является самоцелью, а в качестве приоритетного выдвигается существо вопроса. Для интересующихся доказательной стороной, а также процессом развития идей теории информации учебник содержит ссылки на работы «классиков» — Шеннона, Бриллюэна, Хэмминга, Бауэра и Гооза, Брассара, Глушкова и др; современные подходы легко найти самостоятельно в Интернете, но, как правило, они сводятся к трактовке и пересказу прошлых работ.

Элементы теории вероятностей и математической логики, необходимые для понимания основного материала, изложены в приложениях.

Информатика — наука мировоззренческая уже хотя бы потому, что предметом ее изучения является одна из исходных категорий мироздания — *информация*. Учащиеся (любого уровня) должны осознать фундаментальность и универсальность законов

информатики. Автор надеется, что данное пособие будет способствовать решению этой задачи.

Пособие ориентировано, в первую очередь, на студентов, изучающих информатику в качестве профильной дисциплины, а также школьных учителей информатики.

Для акцентирования внимания читателя на наиболее важных положениях в тексте приняты следующие обозначения:

*таким образом выделяются определения понятий;*

**таким образом выделены формулировки и наиболее важные утверждения.**

Помимо теоретического материала каждая глава содержит примеры решения задач, иллюстрирующих рассматриваемые положения; в конце главы содержатся контрольные вопросы и задания для самостоятельной работы читателя — их выполнение может служить критерием успешности освоения теории.

Следует также отметить, что данный теоретический курс поддержан компьютерным лабораторным практикумом, который охватывает практически все основные разделы дисциплины.

Предыдущее издание данного учебника состоялось довольно давно — более 10 лет назад. Все эти годы курс читался студентам и, естественно, накапливались пожелания по его совершенствованию. Автор получил от читателей ряд замечаний, касавшихся содержания, изменилась учебная программа дисциплины, возникла необходимость рассмотреть некоторые новые разделы (например, элементы криптографии) — все это послужило причинами переработки, исправления и дополнения материала.

Автор будет признателен читателю за отзывы, замечания и предложения по улучшению содержания учебника.

620219, г. Екатеринбург, ул. К. Либкнехта, 9, Уральский государственный педагогический университет, Ин-т математики, информатики и информационных технологий, кафедра информационно-коммуникационных технологий в образовании.

e-mail: bes@uspu.ru

# Введение

По современным представлениям информация является одной из исходных категорий мироздания наряду с материей и энергией. Эти категории взаимосвязаны между собой; усмотреть такие связи можно как в природных явлениях, так и в явлениях и процессах, порожденных человеком. Примерами природных явлений, в которых проявляются связи между материей, энергией и информацией, могут служить:

- фазовый переход твердого тела из кристаллического состояния в жидкое — в нем, наряду с материальными преобразованиями и энергетическими затратами, происходит и потеря информации относительно расположения атомов (сопровождаемая ростом энтропии);
- передача наследственных признаков в живой природе посредством информации, заключенной в молекулы ДНК, что обеспечивает, благодаря разным наборам хромосом, с одной стороны, передачу доминантных признаков данного вида животных или растений, а с другой стороны, адаптацию к изменениям внешних условий существования;
- сохраняется в мозге животного благодаря материальным и энергетическим воздействиям внешней среды.

Примерами связей материя — энергия — информация в обществе людей являются:

- любое производство включает материальную составляющую (исходные материалы), энергетические ресурсы, необходимые для преобразования материальных объектов, а также информационное обеспечение в виде описания технологий, различной документации и пр.;
- подготовка новых членов общества — образование — информационный процесс, требующий как материального, так и энергетического обеспечения;
- управление в любой сфере состоит в выработке решений на основе имеющейся информации, которые могут иметь конкретные материальные и энергетические проявления, например падение с велосипеда или отключение электроэнергии.

Которая из трех перечисленных категорий важнее для человека? Такая постановка вопроса кажется бессмысленной, поскольку всегда можно привести множество примеров приоритетности каждой из этих категорий в конкретных ситуациях. Вместе с тем прогресс человечества неизбежно влечет увеличение общего объема информации, которым оно располагает, причем объем этот растет с течением времени гораздо быстрее, чем население земного шара и его материальные потребности. Таким образом, можно утверждать, что значимость информации по отношению к остальным категориям возрастает. Именно по этой причине дальнейшее развитие человечества связывают с построением и переходом к новой формации — *информационному обществу**.

Почему роль информатики и информационных компьютерных технологий обработки информации стала заметной только в последние несколько десятилетий? Почему информатика как самостоятельная научная дисциплина существует чуть более полувека, хотя с информационными процессами и переработкой информации человечество имело дело всегда? Для ответа на эти вопросы необходимо совершить краткий экскурс в историю, оценивая ее с точки зрения уровня развития информационных процессов. Стоит заметить, что само выделение первобытных людей из мира животных и формирование начальных общественных формаций связано с их коммуникацией в ходе решения совместных задач — охота, борьба со стихиями и пр., т. е. с информационным обменом. Важность коммуникации людей нисколько не изменилась за прошедшие с тех пор тысячелетия — и сейчас любые совместные действия, решение любой задачи, в котором участвует более одного человека, требуют обмена информацией, представленной в понятной всем участникам форме. Для характеристики информационного обеспечения исторических эпох мы выделим несколько параметров:

- организация передачи информации *в пространстве*, т. е. распространение информации с целью обеспечения доступа к ней людей (потребителей), удаленных друг от друга, в относительно небольшом временном интервале;

---

* Интересующихся этой проблемой можно адресовать к книгам К. Поппера, Ф. Хайека, Г. Громова. См. например, К. Поппер. Логика и рост научного знания. М.: Прогресс, 1983; Ф. Хайек. Контрреволюция науки (этюды о злоупотребления разумом). М.: Три квадрата, 2003; Г. Громов. От гиперкниги к гипермозгу. Информационные технологии эпохи Интернета. — М.: Радио и связь, 2004.

- организация передачи информации *во времени*, т. е. накопление и хранение информации, накопленной предыдущими поколениями людей, в интересах будущих;
- организация *обработки* информации, т. е. преобразование имеющейся информации с целью ее использования для решения задач практики — управления, обучения, создания новой информации (наука и искусство) и пр.

По ходу истории человечества улучшение показателей, характеризующих уровень развития перечисленных процессов, происходило неравномерно, что повлекло возникновение и затем преодоление нескольких *информационных барьеров*. Информационные барьеры образовывались в результате противоречий между информационными запросами общества и техническими возможностями их обеспечения. Эти барьеры оказывались препятствием на пути прогресса общества, и потому, как и в случае барьеров материальных или энергетических, человечество всегда находило способ их преодоления. Таких информационных барьеров можно указать три.

*I информационный барьер* был преодолен приблизительно в V тысячелетии до н.э. До того времени единственным хранилищем информации был мозг человека. Передача информации была связана с механическим перемещением самого человека, и, следовательно, скорость передачи оказывалась весьма низкой, а передача ненадежной. Обработка информации также производилась человеком. Противоречие состояло в том, что человечеству требовалась возможность сохранять во времени опыт и знания, накопленные предыдущими поколениями, с тем, чтобы они могли быть переданы поколениям последующим. Барьер был преодолен благодаря появлению письменности. Носителями информации стали камни, глиняные таблички, папирус, пергамент, береста, материя; позднее (во II веке н.э.) появилась бумага.

*II информационный барьер* сформировался к XV в. из-за того, что в связи с развитием производства — появлением цехов, мануфактур — возникла потребность в большом числе образованных людей, способных этим производством управлять. Противоречие состояло в том, что количество источников информации — рукописей, рукописных книг — не могло обеспечить обучение большого количества людей. Изобретение книгопечатания (т. е. тиражирования информации на бумаге) в Европе в XV в. И. Гутенбергом и в России в XVI в. И. Федоровым позволило преодолеть данное противоречие. В то время скорость передачи

информации определялась скоростью механического перемещения ее бумажного носителя. Обработка производилась человеком. Поскольку основным носителем информации являлась бумага и именно этим определялись технологии накопления и распространения информации, по определению В.М. Глушкова, это состояние можно назвать *бумажной информатикой*.

К началу XX в. ситуация изменилась, в первую очередь, в отношении скорости распространения информации: сначала появилась почта; в XIX в. — телеграф, затем телефон; в 1905 г. — радио; в 1920–1930-е годы — телевидение. В результате этих изобретений информация практически мгновенно могла быть доставлена в любую точку земного шара. Появились и новые устройства, обеспечивавшие иные (по сравнению с бумагой) принципы записи информации для хранения — фотография, затем кино, затем магнитная запись. Без существенных изменений оставалась лишь ситуация, связанная с переработкой информации, — эту функцию по-прежнему выполнял только человек.

*К III информационному барьеру* человечество подошло во второй половине XX столетия, когда общий объем информации, которым оно располагало, вырос настолько, что суммарной пропускной способности человеческого мозга оказалось недостаточно для ее переработки. Прогресс человечества стал зависеть от того, удастся ли решить проблему автоматизации обработки информации. Варианты решения появились в 1945–46 гг., когда американские инженеры П. Экерт и Дж. Мокли построили первую цифровую вычислительную машину ЭНИАК*, математик Дж. фон Нейман описал принципы работы автоматических вычислительных устройств, а в 1948 г. К. Шеннон опубликовал знаменитую работу «Математическая теория связи», где изложил математические принципы кодирования и передачи информации, а также предложил метод объективного измерения количества информации — именно эти идеи и составили основу новой науки — ин-

---

* Ранее, в 1943 году, британскими инженерами Т. Флауэрсом и М. Ньюманом была создана вычислительная машина «Колосс» (Colossus) для расшифровки сообщений немецких шифровальных машин, но из-за режима секретности изобретение Colossus не было упомянуто в трудах по истории. Однако еще ранее, в 1942 г., Дж. Атанасов и К. Берри разработали первый в США цифровой компьютер ABC. В 1973 г. федеральный суд США аннулировал патент, ранее принадлежавший Экерту и Мокли, признав Атанасова изобретателем первого электронного компьютера. — *Прим. ред.*

форматики. Итак, преодоление нового информационного барьера породило необходимость создания устройств, обеспечивающих автоматизированную обработку информации, а это, в свою очередь, вызвало к жизни науку, которая определила бы принципы работы таких устройств и общие принципы представления и преобразования информации.

Термин «*информатика*», обозначающий название новой науки, появился и прижился не сразу. В нашей стране в 60-е гг. XX в. вопросы, связанные с разработкой, функционированием и применением автоматизированных систем обработки информации, объединялись термином «*кибернетика*», хотя это было не вполне верно, поскольку, по определению Н. Винера, *кибернетика — это наука о законах управления в живой и неживой природе*, т. е. сфера ее интересов охватывает лишь часть (хотя обширную и важную) используемых человеком информационных процессов и систем. Более общую научную дисциплину, связанную с исследованием информации, в англоязычных странах стали называть *Computer Science* — «*вычислительная наука*». На самом деле это название охватывает группу дисциплин, занимающихся различными аспектами применения и разработки компьютерной техники, — программирование, прикладная математика, операционные системы и языки программирования, архитектура компьютера и т. п.

В 1962 г. во Франции впервые появился термин *Informatique* — «*информатика*», он был позаимствован и с середины 1970-х гг. прочно вошел сначала в научно-технический обиход, а затем стал общеизвестным и общепринятым. Однако предметную область дисциплины *информатика* установившейся считать нельзя. Международный конгресс по информатике 1978 г. предложил следующее определение: «*Понятие информатики охватывает области, связанные с разработкой, созданием, использованием и материально-техническим обслуживанием систем обработки информации, включая машины, оборудование, математическое обеспечение, организационные аспекты, а также комплекс промышленного, коммерческого, административного и социального воздействия*». Сращивание информатики со средствами телекоммуникаций, как отмечает академик В.М. Глушков [14], привело к появлению нового термина — *телематика*, хотя пока он не получил такого же распространения, как *информатика*. Для нашего рассмотрения вполне приемлемым можно считать определение академиков

А.П. Ершова и Б.Н. Наумова:

> ***Информатика** — фундаментальная естественная наука, изучающая общие свойства информации, процессы, методы и средства ее обработки (сбор, хранение, преобразование, перемещение, выдача).*

Отнесение информатики к фундаментальным наукам означает, что она имеет общенаучную значимость, т. е. ее понятия, законы и методы применимы не только в рамках самой науки, но и в иных научных и прикладных дисциплинах.

В информатике выделяются два направления — теоретическое и прикладное. Исследования в области теоретической информатики обеспечивают выявление и формулировку общих законов, касающихся информации и информационных процессов, определение принципов функционирования технических систем, связанных с информационными процессами и обработкой дискретной информации, а также построение методологии создания и использования информационных моделей. Прикладная информатика обеспечивает непосредственное создание информационных систем и программного обеспечения для них, а также их применение для решения задач практики.

Теоретическая информатика — дисциплина, использующая методы математики. Поскольку любая информация может быть представлена в *дискретной форме* (этот термин будет раскрыт далее), для описания информационных процессов может быть использован аппарат дискретной математики. Однако в теоретической информатике этот аппарат наполняется конкретным и специфическим содержанием, поскольку применяется к информационным объектам. Теоретическая информатика объединяет следующие дисциплины: теория информации, теория алгоритмов, теория кодирования, теория систем и моделей, теория конечных автоматов, вычислительная математика, математическое программирование и целый ряд других. В данном учебнике решается задача изложения некоторых фундаментальных основ науки информатики, поэтому из приведенного выше перечня дисциплин будут затронуты лишь некоторые и в той мере, которая определяется понятием «минимальная достаточность». С другой стороны, рассмотрение вопросов прикладного характера и конкретных технических или программных решений также выведено за рамки данного пособия — эти вопросы освещены во многих иных источниках (см., например, библиографию, приведенную в учебном пособии А.В. Могилева, Н.И. Пака, Е.К. Хеннера [30]).

# Часть I

# ТЕОРИЯ ИНФОРМАЦИИ

Теория информации как самостоятельная дисциплина возникла в ходе решения следующей задачи: *обеспечить надежную и эффективную передачу информации от источника к приемнику при условии, что передаче этой препятствуют помехи.* Сама формулировка этой задачи нуждается в ряде уточнений:

- «... *передачу информации*...» предполагает, что информация должна быть каким-то образом оценена количественно (иначе невозможно установить, произошла ли ее потеря или она передана полностью);
- «... *надежную* ...» означает, что в процессе передачи не должно происходить потери информации — приемник полностью, без искажений должен получить информацию, отправленную источником;
- «... *эффективную* ...» означает, что передача должна осуществляться наиболее быстрым способом, поскольку время эксплуатации линии связи — экономический фактор, который требуется минимизировать;
- помехи присутствуют в любой *реальной* линии связи; таким образом, поставленная выше задача имеет четкую практическую направленность.

Решение этой задачи ведется по двум направлениям, которые условно можно назвать *техническим* и *математическим.* Технический поиск связан с практической разработкой линий связи, в которых передача может идти с большой скоростью; обеспечением защиты от помех или уменьшения их воздействия; созданием технических устройств, обеспечивающих быструю и надежную связь. Однако в основе этих разработок лежат некоторые общие законы и принципы, применимые не к какой- то конкретной линии передачи информации, а к любым (во всяком

случае, многим) видам связи. Они определяют способы кодирования информации (в том числе такие, которые позволяют обнаружить и исправить ошибку передачи); условия надежной передачи информации; наконец, что очень важно, вводятся величины, позволяющие количественно описывать информационные процессы. Именно эти методы и составляют содержательную основу *теории информации*.

Теория информации является *математической теорией* с высокой степенью общности. Она основывается на теории случайных событий, для описания которых применяются понятия *вероятность* и энтропия. В рамках самой теории вводится понятие *информация* и устанавливается ее количественная мера — бит. Строится теория информации подобно другим теориям в математике: сначала аксиоматически определяются исходные понятия, а затем из них путем рассуждений доказывается справедливость новых положений или теорем — именно таким путем шел основоположник данной теории Клод Шеннон. В нашем изложении большее внимание будет уделено смыслу и значению теорем, нежели их доказательству. Интересующихся именно математическими аспектами теории можно адресовать к книгам А.М. Яглома и И.М. Яглома [50], Л. Бриллюэна [8], Р.Л. Стратоновича [38], А. Файнстейна [41] и, наконец, работам самого К. Шеннона [46, 47].

Отдельно следует остановиться на практической применимости положений и следствий, выводимых в теории. Примеры использования теории информации можно найти в информатике, технике, психологии, биологии, физике, педагогике, лингвистике и т. д. Однако, как и любая иная математическая теория, теория информации применима для решения конкретных задач практики в той мере, в какой описываемые материальные системы или процессы удовлетворяют исходным положениям теории. Неприменимость ее в остальных случаях ни в коем случае нельзя считать недостатком теории. Речь, в частности, идет о том, что сам исходный термин — *информация* — используется не только в данной теории; однако, если в других дисциплинах (например, философии) ему придается иной смысл, то нельзя требовать, чтобы теория информации была в них применима. Точно также механика Ньютона является теорией, описывающей движение, но не во всем многообразии значений этого термина, а лишь перемещение тел в пространстве с течением времени; другие виды движения — развитие растения, эволюция Вселенной,

изменения в общественном устройстве и т. п. — законами Ньютона, безусловно, не описываются, но это не уменьшает значимости последних.

Математическое понятие информации связано с возможностью ее количественного измерения. При этом в теории информации обосновывается *энтропийный* подход, когда количество информации в сообщении определяется тем, насколько уменьшается неопределенность исхода случайного события (например, появления конкретной буквы в некоторой последовательности символов) после получения сообщения. Сообщение несет полную информацию о событии, если оно целиком снимает исходную неопределенность. В технических приложениях используется иной способ оценки количества информации, основанный на простом подсчете числа знаков в сообщении — такой подход получил название *объемного*. В общем случае эти две меры количества информации не совпадают, в частности, в теории информации показывается, что энтропийная мера не превышает числа двоичных символов (бит) в сообщении. Одинаковым же в обоих подходах оказывается то, что количественная мера информации не привязывается к ее *семантической* (т. е. смысловой) основе. С бытовой точки зрения информация, лишенная смысла, лишена и какой-либо ценности для получателя. Однако устройство, предназначенное для передачи или хранения информации, оценить смысл передаваемого (или сохраняемого) не может (да и не должно!) — в этом случае главной оказывается задача надежной передачи и хранения информации независимо от ее семантической основы. Едва ли нас устроила бы ситуация, если бы почтальон стал оценивать содержание писем и в зависимости от своего понимания их значимости и ценности решать, какие из них доставлять, а какие нет. Почтальон, будучи звеном системы связи, обязан доставить пакет адресату, даже если в нем чистый лист бумаги. При этом важными (существенными) для передачи и хранения оказываются количественные характеристики информации и способы их оценки — именно их теория информации и устанавливает. Таким образом, оказывается, что теория информации применима для решения лишь тех практических задач, в которых допустимо игнорирование смысловой (содержательной) стороны информации.

**Теория информации имеет дело не со смыслом информации, а лишь с ее количеством.**

Еще одно соображение приводит в своей работе Р.В. Хэмминг: «*Теория информации устанавливает границы того, что можно сделать, однако мало помогает при проектировании конкретных систем*» [43, с. 7]. Другими словами, теорией устанавливается то, что допустимо и не может быть нарушено, позволяет получить количественные оценки информационных процессов, однако, она не дает способов реализации этих процессов. В дальнейшем изложении мы неоднократно столкнемся с этой ситуацией.

# 1 Исходные понятия информатики

## 1.1. Начальные определения

Любая наука начинается со строгих определений используемых ею понятий и терминов. Поэтому было бы вполне разумным начать изложение основ теории информации именно с ее точного определения. Определить какое-либо понятие — значит выразить его через другие понятия, уже определенные ранее. Сложность ситуации, однако, в том, что информация является одной из исходных категорий мироздания, и, следовательно, определение «*информации вообще*» невозможно свести к каким-то более простым, более «исходным» терминам. Что касается частных трактовок понятия «*информация*», то следует отметить значительное их расхождение в различных научных дисциплинах, в технике и на бытовом уровне. Такое положение не следует считать каким-то необычным — можно привести много аналогичных примеров, когда термин имеет и используется во множестве значений: движение, энергия, система, связь, язык и пр. Неоднозначность преодолевается тем, что в каждой «узкой» дисциплине дается свое определение термина — его следует считать *частным* — и именно оно используется. Но это, безусловно, не дает основания переносить такое определение и применять его вне рамок данной дисциплины. Например, в теоретической механике «*связь*» определяется как *некое внешнее воздействие, ограничивающее возможности перемещения (степени свободы) тела*; нет смысла такую трактовку пытаться применять, скажем, в телеграфии или социальных науках.

Аналогична ситуация и с термином «*информация*»: на бытовом уровне и во многих научных дисциплинах он ассоциируется с понятиями *сведения, знания, данные, известие, сообщение, управление* и др. Общим во всех перечисленных примерах является то, в них существенным и значимым для использования является содержательная сторона информации — с позиций

«здравого смысла» это представляется вполне естественным. Однако оценка содержания и ценности одной и той же информации различными людьми, вообще говоря, будет различной; объективная количественная мера смысловой стороны информации отсутствует. С другой стороны, можно привести примеры ситуаций, когда семантическая основа информации роли не играет, точнее, она принимается в виде атрибута (свойства, качества) информации, который не должен изменяться, а для этого следует обеспечить неизменность материального представления информации. По этой причине в ряде теоретических, технических и даже организационных приложений можно просто сосредоточиться на задаче обеспечения неизменности информации в процессах, с ней связанных (в первую очередь, это передача и хранение), а также поиске наилучших условий осуществления этих процессов безотносительно содержания самой информации. Например, задача библиотеки — обеспечить хранение, учет и доступ читателей к любым имеющимся в фонде книгам, независимо от их содержания; действия библиотекарей сводятся к тому, чтобы по формальным признакам — кодам, фамилии автора, названию — отыскать нужную книгу и выдать читателю; при этом содержание, полезность, новизну, значимость и т. п. книги оценивает именно читатель (т. е. лицо, использующее информацию), а не тот, кто ее хранит.

Отделив информацию от ее семантической основы, мы получаем возможность построить определение информации и параллельно ввести ее *объективную* количественную меру. Будет использован способ определения, который называется *операционным* и который состоит в описании метода измерения или вычисления значения определяемой величины — такому способу отдается предпочтение в научном знании, поскольку он обеспечивается однозначность и объективность, чего трудно добиться для категорий, не имеющих меры (например, попробуйте определить понятие «*доброта*»). Открытие такого способа определения информации является одной из главных заслуг теории информации.

Операционное определение информации будет нами рассмотрено в главе 2, поскольку оно требует освоения ряда предшествующих и сопутствующих понятий. Пока же мы обсудим особенность, которой обладает любая информация, — это то, что информация — категория *нематериальная*. Следовательно, для существования и распространения в нашем материальном мире она должна быть обязательно связана с какой-либо материальной

основой — без нее информация не может проявиться, передаваться и сохраняться, например восприниматься и запоминаться нами. Введем определение:

> *Материальный объект или среду, которые служат для представления или передачи информации, будем называть ее* ***материальным носителем****.*

Материальным носителем информации может быть бумага, воздух, лазерный диск, электромагнитное поле и пр. При этом *хранение* информации связано с некоторой характеристикой носителя, которая не меняется с течением времени, например намагниченные области поверхности диска или буква на бумаге, а *передача* информации — наоборот, с характеристикой, которая изменяется с течением времени, например амплитуда колебаний звуковой волны или напряжение в проводах. Другими словами, хранение информации связано с фиксацией *состояния* носителя, а распространение — с *процессом*, который протекает в носителе. Состояния и процессы могут иметь физическую, химическую, биологическую или иную основу — главное, что они материальны.

Однако не с любым процессом можно связать передачу информации. В частности, *стационарный* процесс, т. е. процесс с неизменными в течение времени характеристиками, информацию не переносит. Примером может служить постоянный электрический ток, ровное горение лампы или равномерный гул — они содержат лишь ту информацию, что процесс идет, т. е. что-то функционирует. Иное дело, если мы будем лампу включать и выключать, т. е. изменять ее яркость, — чередованием вспышек и пауз можно представить и передать информацию (например, посредством азбуки Морзе). Таким образом, для передачи необходим *нестационарный процесс*, т. е. процесс, характеристики которого могут изменяться; при этом информация связывается не с существованием процесса, а именно с изменением какой-либо его характеристики.

> *Изменение характеристики носителя, которое используется для представления информации, называется* ***сигналом****, а значение этой характеристики, отнесенное к некоторой шкале измерений, называется* ***параметром сигнала****.*

В табл. 1.1 приведены примеры процессов, используемых для передачи информации, и связанных с ними сигналов.

Однако одиночный сигнал, как мы увидим в дальнейшем, не может нести большое количество информации. Поэтому для

Таблица 1.1

| Способ передачи | Процесс | Параметры сигнала |
|---|---|---|
| Звук | Звуковые волны | Высота и громкость звука |
| Радио, телевидение | Радиоволны | Частота, амплитуда или фаза радиоволны |
| Изображение | Световые волны | Частота и амплитуда световых волн |
| Телефон, компьютерная сеть | Электрический ток | Частота и амплитуда электрических колебаний в линии связи |

передачи информации используется ряд следующих друг за другом сигналов.

*Последовательность сигналов называется* ***сообщением****.*

Таким образом, от источника к приемнику информация передается в виде сообщений. Можно сказать, что сообщение выступает в качестве *материальной оболочки* для представления информации при передаче. Следовательно, *сообщение служит переносчиком информации, а информация является содержанием сообщения.*

*Соответствие между сообщением и содержащейся в нем информацией называется* ***правилом интерпретации сообщения****.*

Это соответствие может быть *однозначным* и *неоднозначным*. В первом случае сообщение имеет лишь одно правило интерпретации. Например, по последовательности точек, тире и пауз в азбуке Морзе однозначно восстанавливается переданная буква. Неоднозначность соответствия между сообщением и информацией возможна в двух вариантах:

- одна и та же информация может передаваться различными сообщениями (например, прогноз погоды может быть получен по радио, из газеты, по телефону и пр.);
- одно и то же сообщение может содержать различную информацию для разных приемников (примером может служить передача в 1936 г. по радио фразы «*Над всей Испанией безоблачное небо*», которое для непосвященных людей имело смысл прогноза погоды, а для знакомых с правилом интерпретации — сигналом к началу военных действий).

Обсудим следующее исходное понятие — *информационный процесс*. Вообще термин «*процесс*» применяется в тех случаях, когда некоторое качество, характеризующее систему или объект, меняется с течением времени в результате внешних воздействий

или каких-то внутренних причин. Какие атрибуты могут изменяться с течением времени у нематериальной информации? Очевидно, только ее содержание и материальная оболочка, посредством которого информация представлена, т. е. сообщение. В связи с этим примем следующее определение:

> ***Информационный процесс*** *— это изменение с течением времени содержания информации или представляющего его сообщения.*

Различных видов информационных процессов оказывается немного:

- порождение (создание) новой информации;
- преобразование информации (т. е. порождение новой информации в результате обработки имеющейся);
- уничтожение информации;
- передача информации (распространение в пространстве).

На самом деле все перечисленные события происходят не непосредственно с самой информацией, а с сообщением, т. е. ее материальным представлением. И с этих позиций возможны лишь два типа процессов: изменение сообщения с сохранением содержащейся в нем информации и изменение сообщения, сопровождающееся преобразованием информации. К процессам первого типа относится передача информации без потерь и обратимая перекодировка; к процессам второго типа — создание-уничтожение информации, необратимая перекодировка, передача с потерями, обработка с появлением новой информации (нового содержания).

Отдельно следует остановиться на хранении информации. Как уже было сказано, хранение связывается с фиксацией параметра материального носителя, который далее с течением времени не меняется. Следовательно, запись информации на носитель (непосредственно момент фиксации параметра) и ее последующее считывание подпадают под определение информационного процесса, но само хранение — нет. Хранение следовало бы назвать *информационным состоянием*, однако такое понятие в информатике не используется.

С передачей информации связана еще одна пара исходных сопряженных понятий — *источник* и *приемник* информации.

> ***Источник информации*** *— это субъект, процесс или устройство, порождающие информацию и представляющие ее в виде сообщения.*

> ***Приемник информации*** *— это субъект или устройство, принимающие сообщение и способные правильно его интерпретировать.*

В этих определениях сочетание «субъект, процесс или устройство» означает, что источники и приемники информации могут быть одушевленными (человек, животные) или неодушевленными (технические устройства, природные явления). Для того чтобы объект (или субъект) считался источником информации, он должен не только ее породить, но и иметь возможность инициировать какой-то нестационарный процесс и связать информацию с его параметрами, т. е. создать сообщение. Например, если человек что-то придумал, но держит это в своем мозге, он не является источником информации; однако он им становится, как только свою идею изложит на бумаге (в виде текста, рисунка, схемы и пр.) или выскажет словами.

В определении приемника информации важным представляется то, что факт приема сообщения еще не означает получение информации; информация может считаться полученной только в том случае, если приемнику известно правило интерпретации сообщения. Другими словами, понятия «*приемник сообщения*» и «*приемник информации*» не тождественны. Например, слыша речь на незнакомом языке, человек оказывается приемником сообщения, но не приемником информации.

Для связи с внешним миром у человека, как известно, имеются пять органов чувств. Следовательно, воспринимать сообщение мы можем только посредством одного из них (или группой органов). Это не означает, однако, что человек не может использовать для передачи и приема информации какие-то иные процессы, им не воспринимаемые, например радиоволны. В этом случае человек-источник использует промежуточное устройство, преобразующее его сообщение в радиоволны — радиопередатчик, а человек-приемник — другое промежуточное устройство — радиоприемник, преобразующий радиоволны в звук. Такой подход заметным образом расширяет возможности человека в осуществлении передачи и приема информации. Промежуточные устройства-преобразователи получили название *технические средства связи:*

> ***Технические средства связи*** *— устройства, осуществляющие преобразование сообщения из одной формы представления в другую.*

Технические средства связи в совокупности с соединяющей их средой они образуют *линию связи*. К ним относятся телеграф, телефон, радио и телевидение, компьютерные телекоммуникации и пр. При использовании таких средств возникает необходимость преобразования сообщения из одного вида в другой без существенной для получателя потери информации, а также увязки скорости передачи сообщения (т. е. интервала следования и величины отдельных сигналов) с возможностями линии связи и приемника. Таким образом, поставленная ранее проблема передачи информации без потерь дополняется еще одной проблемой: *возможно ли преобразование сообщений без потерь содержащейся в них информации*? Обе эти проблемы оказываются центральными в теории информации.

## 1.2. Формы представления информации

В предыдущем разделе было сказано, что информация передается с помощью сигналов, а самим сигналом является изменение некоторой характеристики носителя с течением времени. При этом в зависимости от особенностей изменения этой характеристики (т. е. параметра сигнала) с течением времени выделяют два типа сигналов: *непрерывные* и *дискретные*.

*Сигнал называется* ***непрерывным*** *(или аналоговым), если его параметр может принимать любое значение в пределах некоторого интервала.*

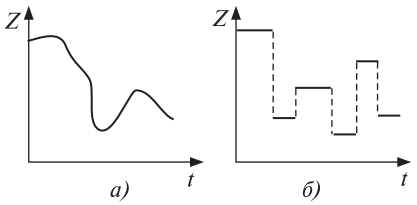



**Рис. 1.1.** Непрерывные (*а*) и дискретные (*б*) сигналы

Если обозначить через $Z$ параметр сигнала, а через $t$ — время, то зависимость $Z(t)$ будет непрерывной функцией (рис. 1.1,*а*).

Примерами непрерывных сигналов являются речь и музыка, изображение, показание термометра (параметр сигнала — высота столба спирта или ртути — имеет непрерывный ряд значений) и пр.

*Сигнал называется* ***дискретным****, если его параметр может принимать конечное число значений в пределах некоторого интервала.*

Пример дискретных сигналов представлен на рис. 1.1,*б*. Как следует из определения, дискретные сигналы могут быть описа-

ны *дискретным* и *конечным* множеством значений параметров $\{Z\}$. Примерами устройств, использующих дискретные сигналы, являются часы (электронные и механические), цифровые измерительные приборы, книги, табло и пр.

Поскольку последовательность сигналов есть сообщение, качество прерывности-непрерывности сигналов переносится и на сообщение — существуют понятия «*непрерывное сообщение*» и «*дискретное сообщение*». Очевидно, что дискретным будет считаться сообщение, построенное из дискретных сигналов. Гораздо меньше оснований приписывать данное качество самой информации, поскольку информация — категория нематериальная и не может обладать свойством дискретности или непрерывности. С другой стороны, одна и та же информация, как уже было сказано, может быть представлена посредством различных сообщений, в том числе и отличающихся характером сигналов. Например, речь, которую мы слышим, можно записать в аналоговом виде с помощью магнитофона, а можно и законспектировать посредством дискретного набора букв. По этой причине в информатике существуют и используются сочетания «*непрерывная информация*» и «*дискретная информация*». Их нужно понимать только как сокращение полных фраз: «*информация, представленная посредством непрерывных сигналов*» и «*информация, представленная посредством дискретных сигналов*» — именно в таком контексте эти понятия будут использоваться в дальнейшем изложении. Поэтому когда заходит речь о видах информации, правильнее говорить о формах ее представления в сообщении или о видах сообщений.

Принципиальным и важнейшим различием непрерывных и дискретных сигналов является то, что дискретные сигналы можно *обозначить*, т. е. приписать каждому из конечного числа возможных значений сигнала *знак*, который будет отличать данный сигнал от другого.

> ***Знак** — это элемент некоторого конечного* множества отличных друг от друга сущностей.*

Природа знака может любой — жест, рисунок, буква, сигнал светофора, определенный звук и т. д. — она определяется носи-

---

* Теоретически можно было бы обойтись без требования конечности, однако это не имело бы никакого практического значения, поскольку за конечное время всегда можно передать только сообщения, построенные из конечного числа знаков.

телем сообщения и формой представления информации в сообщении.

Вся совокупность знаков, используемых для представления дискретной информации, называется *набором знаков*. Таким образом, *набор есть дискретное множество знаков*.

*Набор знаков, в котором установлен порядок их следования, называется* ***алфавитом***.

Следовательно, *алфавит — это упорядоченная совокупность знаков*. Порядок следования знаков в алфавите называется *лексикографическим*. Благодаря этому порядку между знаками устанавливаются отношения «больше—меньше»: для двух знаков $\xi$ и $\psi$ принимается, что $\xi < \psi$, если порядковый номер у $\xi$ в алфавите меньше, чем у $\psi$.

Примером алфавита может служить совокупность арабских цифр 0,1,..., 9 — с его помощью можно записать любое целое число в системах счисления от двоичной до десятичной. Если к этому алфавиту добавить знаки «+» и «−», то сформируется набор знаков, применимый для записи любого целого числа, как положительного, так и отрицательного; правда, этот набор нельзя считать алфавитом, поскольку в нем не определен порядок следования знаков. Наконец, если добавить знак разделителя разрядов («.» или «,»), то такой набор позволит записать любое вещественное число.

Поскольку при передаче сообщения параметр сигнала должен меняться, очевидно, что минимальное количество различных его значений равно двум и, следовательно, *алфавит содержит минимум два знака* — такой алфавит называется *двоичным*. Верхней границы числа знаков в алфавите не существует; примером могут служить иероглифы, каждый из которых обозначает целое понятие, и общее их количество исчисляется тысячами.

Знаки, используемые для обозначения фонем человеческого языка, называются буквами, а их совокупность — *алфавитом языка*.

Сами по себе знак или буква не несут никакого смыслового содержания. Однако такое содержание им может быть приписано — в этом случае знак будет называться *символом*. Например, массу в физике принято обозначать буквой $m$ — следовательно, $m$ является символом физической величины «масса» в формулах. Другим примером символов могут служить *пиктограммы*, обозначающие в компьютерных программах объекты или действия.

Таким образом, понятия «*знак*», «*буква*» и «*символ*» нельзя считать тождественными, хотя весьма часто различия между ними не проводят, поэтому в информатике существуют понятия «*символьная переменная*», «*кодировка символов алфавита*», «*символьная информация*» — во всех приведенных примерах вместо термина «*символьный*» корректнее было бы использовать «*знаковый*» или «*буквенный*».

Представляется важным еще раз подчеркнуть, что *понятия знака и алфавита можно отнести только к дискретным сообщениям.*

## 1.3. Преобразование сообщений

Вернемся к обсуждению информационных процессов, связанных с преобразованием одних сигналов в другие. Ясно, что технически это осуществимо. Ранее сигналы и их последовательности — сообщения — были названы нами «*материальными оболочками для информации*», и, естественно, встает вопрос: при изменении «оболочки» что происходит с его содержимым, т. е. с информацией? Попробуем найти ответ на него.

Поскольку имеются два типа сообщений, между ними, очевидно, возможны четыре варианта преобразований (рис. 1.2).

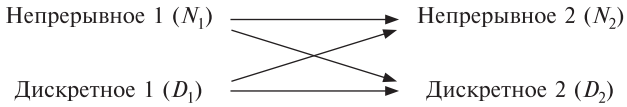



**Рис. 1.2.** Преобразования сообщений

Осуществимы и применяются на практике все четыре вида преобразований. Рассмотрим примеры устройств и ситуаций, связанных с такими преобразованиями, и одновременно попробуем отследить, что при этом происходит с информацией.

Примерами устройств, в которых осуществляется преобразование типа $N_1 \to N_2$, являются микрофон (звук преобразуется в электрические сигналы); магнитофон и видеомагнитофон (чередование областей намагничения ленты превращается в электрические сигналы, которые затем преобразуются в звук и изображение); телекамера (изображение и звук превращаются в электрические сигналы); радио- и телевизионный приемник (радиоволны преобразуются в электрические сигналы, а затем в звук и изображение); аналоговая вычислительная машина (одни электрические сигналы преобразуются в другие). Особенностью данного ва-

рианта преобразования является то, что *оно всегда сопровождается частичной потерей информации*. Потери связаны с помехами (шумами), которые порождает само информационное техническое устройство и которые воздействуют извне. Эти помехи примешиваются к основному сигналу и искажают его. Поскольку параметр сигнала может иметь любые значения (из некоторого интервала), то невозможно отделить ситуации: был ли сигнал искажен или он изначально имел такую величину. В ряде устройств искажение происходит в силу особенностей преобразования в них сообщения, например в черно-белом телевидении теряется цвет изображения; телефон пропускает звук в более узком частотном интервале, чем интервал человеческого голоса; кино- и видеоизображение оказываются плоскими, они утрачивают объемность.

Теперь обсудим общий подход к преобразованию типа $N \to D$. С математической точки зрения перевод сигнала из аналоговой формы в дискретную означает замену описывающей его непрерывной функции времени $Z(t)$ на некотором отрезке $[t_1, t_2]$ конечным множеством (массивом) $\{Z_i, t_i\}$, $i = 0, \ldots, n$, где $n$ — количество точек разбиения временно́го интервала). Подобное преобразование называется *дискретизацией* непрерывного сигнала и осуществляется посредством двух операций: *развертки по времени* и *квантования по величине* сигнала.

*Развертка по времени* состоит в том, что наблюдение за значением величины $Z$ осуществляется не непрерывно, а лишь в определенные моменты времени с интервалом $\Delta t = (t_n - t_0)/n$.

*Квантование по величине* — это отображение вещественных значений параметра сигнала в конечное множество чисел, кратных некоторой постоянной величине — *шагу квантования* ($\Delta Z$).

Совместное выполнение обоих операций эквивалентно нанесению масштабной сетки на график $Z(t)$, как показано на рис. 1.3. Далее, в качестве пар значений $\{Z_i, t_i\}$ выбираются узлы сетки, расположенные наиболее близко к $Z(t_i)$. Полученное таким образом множество узлов оказывается дискретным представлением исходной непрерывной функции. Таким образом, любое сообщение, связанное с ходом $Z(t)$, может быть преобразовано в дискретное, т. е. представлено посредством некоторого алфавита.

При такой замене довольно очевидно, что чем меньше $n$ (больше $\Delta t$), тем меньше число узлов, но и точность замены $Z(t)$ значениями $Z_i$ будет меньшей. Другими словами, при дискрети-

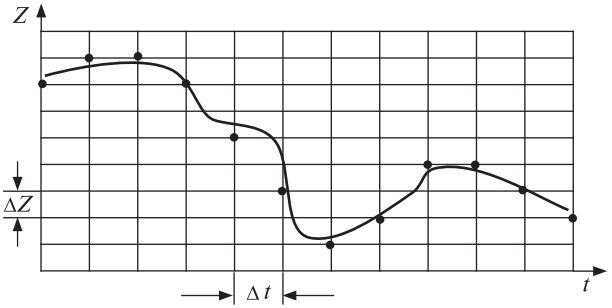



**Рис. 1.3.** Дискретизация аналогового сигнала за счет операций развертки по времени и квантования по величине

зации может происходить *потеря* части информации, связанной с особенностями функции $Z(t)$. На первый взгляд кажется, что увеличением количества точек $n$ можно улучшить соответствие между получаемым массивом и исходной функцией, однако полностью избежать потерь информации все равно не удастся, поскольку $n$ — величина конечная. Ответом на эти сомнения служит так называемая *теорема отсчетов*, доказанная в 1933 г. В.А. Котельниковым (по этой причине ее иногда называют его именем), значение которой для решения проблем передачи информации было осознано лишь в 1948 г. после работ К. Шеннона. Теорема, которую мы примем без доказательства, но результаты будем в дальнейшем использовать, гласит:

**Непрерывный сигнал можно полностью отобразить и точно воссоздать по последовательности измерений (отсчетов) величины этого сигнала через одинаковые интервалы времени, меньшие или равные половине периода максимальной частоты, имеющейся в сигнале.**

Комментарии к теореме:

1. Теорема касается только тех линий связи, в которых для передачи используются колебательные или волновые процессы. Это не должно восприниматься как заметное ограничение, поскольку действие большинства практических устройств связи основано именно на этих процессах.

2. Любое подобное устройство использует не весь спектр частот колебаний, а лишь какую-то его часть; например, в аналоговых телефонных линиях используются колебания с частотами от 300 до 3400 Гц. Согласно теореме отсчетов определяющим является значение верхней границы частоты — обозначим его $\nu_m$.

Смысл теоремы в том, что дискретизация не приведет к потере информации и по дискретным значениям можно будет полностью восстановить исходный аналоговый сигнал, если развертка по времени выполнена в соответствии со следующим соотношением:

$$\Delta t \leqslant \frac{1}{2\nu_m}. \tag{1.1}$$

Можно переформулировать теорему отсчетов:

**Развертка по времени может быть осуществлена без потери информации, связанной с особенностями непрерывного (аналогового) сигнала, если шаг развертки не будет превышать $\Delta t$, определяемый в соответствии с (1.1).**

Например, для оцифровки аналогового телефонного сигнала требуется реализовать 6800 отсчетов в секунду; известно, что человеческое ухо воспринимает звуковые сигналы частотой до 20 кГц, что тредует записи с отсчетами не менее 40000 в секунду — именно поэтому в промышленном стандарте дла записи звука на компакт-диске используется частота 44,1 кГц).

Операция перевода аналогового сообщения в дискретную форму представления осуществляется посредством специального устройства — *аналого-цифрового препобразователя* (АЦП, в англоязычном обозначении — ADC, Analog to Digital Convertor, A/D).

Устройства для обратного перевода — из дискретной формы в аналоговую — называются цифро-аналоговыми преобразователями — ЦАП (или DAC, Digital to Analog Convertor, D/A). Если шаг развертки удовлетворял (1.1), ЦАП полностью восстановит аналоговые сигналы по дискретным.

Однако, помимо временной развертки, дискретизация имеет и другую составляющую — квантование. Какими соображениями определяется шаг квантования $\Delta Z$? Любой получатель сообщения — человек или устройство — всегда имеют конечную предельную точность распознавания величины сигнала. Например, человеческий глаз в состоянии различить около 16 миллионов цветовых оттенков; это означает, что при квантовании цвета нет смысла делать большее число градаций. При передаче речи достаточной оказывается гораздо меньшая точность в определении громкости — около 1 %; следовательно, для амплитуды звуковых колебаний $\Delta Z \approx 0{,}01 Z_{\max}$, а алфавит для обозначения всех градаций громкости должен содержать 100 знаков. Мы приходим

к заключению, что *шаг квантования определяется чувствительностью приемного устройства.*

Указанные соображения по выбору шага развертки по времени и квантования по величине сигнала лежат в основе оцифровки звука и изображения. Примерами устройств, в которых происходят такие преобразования, являются сканер, модем, устройства для *цифровой записи* звука и изображения, лазерный проигрыватель, графопостроитель. Термины «*цифровая запись*», «*цифровой сигнал*» следует понимать как *дискретное представление с применением двоичного цифрового алфавита.*

Таким образом, мы приходим к очень важному выводу: *преобразование сигналов типа $N \to D$, как и обратное $D \to N$, может осуществляться без потери содержащейся в них информации.*

Преобразование типа $D_1 \to D_2$ состоит в переходе при представлении сигналов от одного алфавита к другому — такая операция носит название *перекодировка* и может осуществляться без потерь. Примерами ситуаций, в которых осуществляются подобные преобразования, могут быть: запись-считывание с компьютерных носителей информации; шифровка и дешифровка текста; вычисления на калькуляторе.

Таким образом, за исключением $N_1 \to N_2$ в остальных случаях оказывается возможным преобразование сообщений без потерь содержащейся в них информации. При этом на первый взгляд непрерывные и дискретные сообщения оказываются равноправными. Однако на самом деле это не так. Сохранение информации в преобразованиях $N \to D$ и $D \to N$ обеспечивается именно благодаря участию в них дискретного представления. Другими словами, *преобразование сообщений без потерь информации возможно только в том случае, если хотя бы одно из них является дискретным.* В этом проявляется несимметричность видов сообщений и преимущество дискретной формы; к другими ее достоинствам следует отнести:

- высокую помехоустойчивость;
- простоту и, как следствие, надежность и относительную дешевизну устройств по обработке информации;
- точность обработки информации, которая определяется количеством обрабатывающих элементов и не зависит от точности их изготовления;
- универсальность устройств.

Последнее качество — *универсальность* — оказывается следствием того обстоятельства, что любые дискретные сообщения, составленные в различных алфавитах, посредством обратимого кодирования можно привести к единому алфавиту. Это позволяет выделить некоторый алфавит в качестве *базового* (из соображений удобства, простоты, компактности или каких-либо иных) и *представлять в нем любую дискретную информацию*. Тогда устройство, работающее с информацией в базовом алфавите, оказывается универсальным в том отношении, что оно может быть использовано для переработки любой исходной дискретной информации. Таким базовым алфавитом, как мы увидим в дальнейшем, является *двоичный алфавит*, а использующим его универсальным устройством — *компьютер*.

Несимметричность непрерывной и дискретной информации имеет более глубокую основу, чем просто особенности представляющих сигналов. Дело в том, что информация, порождаемая и существующая в природе, связана с материальным миром — это размеры, форма, цвет и другие физические, химические и прочие характеристики и свойства объектов. Данная информация передается, как было сказано, посредством физических и иных взаимодействий и процессов. Бессмысленно ставить вопросы: для чего существует такая информация и кому она полезна? Эту *природную* информацию можно считать хаотической и неупорядоченной, поскольку никем и ничем не регулируется ее появление, существование, использование. Чаще всего она непрерывна по форме представления. Напротив, дискретная информация — это информация, прошедшая обработку — отбор, упорядочение, преобразование; она предназначена для дальнейшего применения человеком или техническим устройством. Дискретная информация даже может не иметь прямого отношения к природе и материальным объектам, например представления и законы математики. Другими словами, дискретная — это уже частично осмысленная информация, т. е. имеющая для кого-то смысл и значение и, как следствие, более высокий (с точки зрения пользы) статус, нежели непрерывная, хаотичная. Однако в информатике, как было сказано, этот смысл не отслеживается, хотя и подразумевается. Эту же мысль можно выразить иначе: информатика имеет дело не с любой информацией и не с информацией вообще, а лишь с той, которая кому-то необходима; при этом не ставятся и не обсуждаются вопросы «*Зачем она нужна*?» и «*Почему именно эта*?» — это определяет потребитель информации.

Отсюда становится понятной приоритетность дискретной формы представления информации по отношению к непрерывной в решении глобальной задачи автоматизации обработки информации. Приведенные в данном разделе соображения позволяют нам в дальнейшем исследовать только дискретную информацию, а для ее представления (фиксации) использовать некоторый алфавит. При этом нет необходимости рассматривать физические особенности передачи и представления, т. е. характер процессов и виды сигналов, — полученные результаты будут справедливы для любой дискретной формы представления информации независимо от физической реализации сообщения, с которым она связана. С этого момента и начинается наука информатика.

## Контрольные вопросы и задания к гл. 1

1. Приведите примеры терминов, имеющих несколько трактовок в различных науках, технике, быту.

2. Приведите примеры процессов, используемых для передачи информации, и связанных с ними сигналов, помимо указанных в тексте.

3. Приведите примеры неоднозначного и однозначного соответствия между сообщением и содержащейся в нем информацией.

4. Может ли существовать информация, если она не представлена в форме сообщения? Может ли существовать сообщение, не содержащее информации?

5. Почему хранение информации нельзя считать информационным процессом?

6. В чем состоит различие понятий «*приемник сообщения*» и «*приемник информации*»?

7. Органы чувств человека ориентированы на восприятие аналоговых сообщений. Означает ли это, что мы не можем воспринимать информацию в дискретной форме представления?

8. Приведите примеры знаков-символов. Могут ли символы образовывать алфавит?

9. В шестнадцатиричной системе счисления используются цифры *A, B, C, D, E* и *F*. Следует ли эти знаки считать символами?

10. В тексте данной главы разграничиваются понятия «*знак*», «*буква*», «*символ*». Как соотносится с ними понятие «*цифра*»? «*нота*»?

11. В чем состоит смысл и значение теоремы отсчетов В.А. Котельникова?

12. Какое количество отсчетов за 1 с необходимо производить цифровому звукозаписывающему устройству, если требуется обеспечить качество записи (a) телефона; (b) лазерного диска.

13. Как следует понимать термины «*оцифровка изображения*» и «*оцифровка звука*»? Какими устройствами производятся данные операции?

14. Приведите примеры преобразований типа $D_1 \to D_2$, при которых информация, содержащаяся в исходном сообщении, может не сохраняться.

15. Почему для представления дискретных сообщений в качестве базового выбирается двоичный алфавит?

16. Почему компьютер является универсальным устройством по обработке информации?

17. В чем состоит и как проявляется несимметричность непрерывной и дискретной форм представления информации?

# 2 Понятие информации в теории Шеннона

Материал данной главы опирается на понятия и соотношения теории вероятностей. Для тех, кто не знаком с этой теорией, необходимые ее элементы изложены в приложении А. Текст главы содержит ссылки на некоторые формулы из этого приложения — они в качестве первого индекса имеют букву «А».

## 2.1. Понятие энтропии

### 2.1.1. Энтропия как мера неопределенности

Случайные события могут быть описаны с использованием понятия «*вероятность*» (см. приложение А). Соотношения теории вероятностей позволяют найти (вычислить) вероятности как одиночных случайных событий, так и сложных опытов, объединяющих несколько независимых или связанных между собой событий. Однако описать случайные события можно не только в терминах вероятностей.

То, что событие случайно, означает отсутствие полной уверенности в его наступлении, что, в свою очередь, создает *неопределенность* в исходах опытов, связанных с данным событием. Безусловно, *степень неопределенности* различна для разных ситуаций. Например, если опыт состоит в определении возраста случайно выбранного студента 1-го курса дневного отделения вуза, то с большой долей уверенности можно утверждать, что он окажется менее 30 лет; хотя по положению на дневном отделении могут обучаться лица в возрасте до 35 лет, чаще всего очно учатся выпускники школ ближайших нескольких выпусков. Гораздо меньшую определенность имеет аналогичный опыт, если проверяется, будет ли возраст произвольно выбранного студента меньше 18 лет. Для практики важно иметь возможность произвести численную оценку неопределенности разных опытов. Попробуем ввести такую количественную меру неопределенности.

Начнем с простой ситуации, когда опыт имеет $n$ равновероятных исходов. Очевидно, что неопределенность каждого из них зависит от $n$, т. е. *мера неопределенности является функцией числа исходов* $f(n)$.

Можно указать некоторые свойства этой функции:

1) $f(1) = 0$, поскольку при $n = 1$ исход опыта не является случайным и, следовательно, неопределенность отсутствует;

2) $f(n)$ возрастает с ростом $n$, поскольку, чем больше число возможных исходов, тем более затруднительным становится предсказание результата опыта.

Для определения явного вида функции $f(n)$ рассмотрим два *независимых* опыта $\alpha$ и $\beta$* с количествами равновероятных исходов, соответственно $n_\alpha$ и $n_\beta$. Пусть имеет место сложный опыт, который состоит в одновременном выполнении опытов $\alpha$ и $\beta$; число возможных его исходов равно $n_\alpha n_\beta$, причем, все они равновероятны. Очевидно, неопределенность исхода такого сложного опыта $\alpha \wedge \beta$ будет больше неопределенности опыта $\alpha$, поскольку к ней добавляется неопределенность $\beta$; мера неопределенности сложного опыта равна $f(n_\alpha n_\beta)$. С другой стороны, меры неопределенности отдельных $\alpha$ и $\beta$ составляют, соответственно, $f(n_\alpha)$ и $f(n_\beta)$. В первом случае (сложный опыт) проявляется общая (суммарная) неопределенность совместных событий, во втором — неопределенность каждого из событий в отдельности. Однако из независимости $\alpha$ и $\beta$ следует, что в сложном опыте они никак не могут повлиять друг на друга и, в частности, $\alpha$ не может оказать воздействия на неопределенность $\beta$ и наоборот. Следовательно, мера суммарной неопределенности должна быть равна сумме мер неопределенности каждого из опытов, т. е. *мера неопределенности аддитивна*:

$$f(n_\alpha n_\beta) = f(n_\alpha) + f(n_\beta). \tag{2.1}$$

Теперь задумаемся о том, каким может быть явный вид функции $f(n)$, чтобы он удовлетворял свойствам (1) и (2) и соотношению (2.1)? Легко увидеть, что такому набору свойств удовлетворяет функция $\log n$, причем можно доказать, что она *единственная* из всех существующих классов функций. Таким образом:

---

* Для обозначения *опытов* со случайными исходами будем использовать греческие буквы ($\alpha, \beta$ и т. д.), а для обозначения отдельных **исходов** опытов (событий) — латинские заглавные ($A, B$ и т.д).

**За меру неопределенности опыта с $n$ равновероятными исходами можно принять величину $\log n$.**

Следует заметить, что выбор основания логарифма в данном случае значения не имеет, поскольку в силу известной формулы преобразования логарифма от одного основания к другому:

$$\log_b n = \log_b a \log_a n,$$

переход к другому основанию состоит во введении одинакового для обеих частей выражения (2.1) постоянного множителя $\log_b a$, что равносильно изменению масштаба (т. е. размера единицы) измерения неопределенности. Поскольку это так, мы имеет возможность выбрать удобное для нас (из каких-то дополнительных соображений) основание логарифма. Таким удобным основанием оказывается 2, поскольку в этом случае за единицу измерения принимается неопределенность, содержащаяся в опыте, имеющем лишь два равновероятных исхода, которые можно обозначить, например, ИСТИНА (*True*) и ЛОЖЬ (*False*) и использовать для анализа таких событий аппарат математической логики.

*Единица измерения неопределенности при двух возможных равновероятных исходах опыта называется* ***бит****.

Таким образом, нами установлен явный вид функции, описывающей меру неопределенности опыта, имеющего $n$ равновероятных исходов:

$$f(n) = \log_2 n = H. \tag{2.2}$$

Эта величина получила название *энтропия*. В дальнейшем будем обозначать ее $H$.

Вновь рассмотрим опыт с $n$ равновероятными исходами. Поскольку каждый исход случаен, он вносит свой вклад в неопределенность всего опыта, но так как все $n$ исходов равнозначны, разумно допустить, что и их неопределенности одинаковы. С другой стороны, общая (суммарная) неопределенность согласно (2.2) равна $\log_2 n$ — представляется вполне очевидным, что неопределенность, связанная с каждым отдельным исходом, составляет

$$\frac{1}{n}\log_2 n = -\frac{1}{n}\log_2 \frac{1}{n} = -p \log_2 p,$$

где $p = 1/n$ — вероятность любого из отдельных исходов.

---

* Название *бит* происходит от английского *binary digit*, что в дословном переводе означает «двоичный разряд» или «двоичная единица».

Таким образом, вклад в энтропию всего опыта, вносимый каждым отдельным из равновероятных исходов,

$$H_i = -p \log_2 p. \tag{2.3}$$

Теперь попробуем обобщить формулу (2.3) на ситуацию, когда исходы опыта *независимы,* но *неравновероятны*, например $p(A_1)$ и $p(A_2)$. Тогда

$$H_2 = -p(A_2) \log_2 p(A_2) H_1 = -p(A_1) \log_2 p(A_1)$$

и энтропия опыта

$$H = H_1 + H_2 = -p(A_1) \log_2 p(A_1) - p(A_2) \log_2 p(A_2).$$

Обобщая это выражение на ситуацию, когда опыт $\alpha$ имеет $n$ неравновероятных независимых исходов $A_1, A_2, \ldots, A_n$, получим

$$H(\alpha) = -\sum_{i=1}^{n} p(A_i) \log_2 p(A_i). \tag{2.4}$$

Введенная таким образом величина, как уже было сказано, называется *энтропией опыта* $\alpha$, реализуемого через случайные события $A_i$, $i = 1, \ldots, n$.

Сопоставляя выражение (2.4) с формулой для среднего значения дискретных случайных величин (А.11), можно его интерпретировать следующим образом: будем считать, что с исходом $A_i$ связана энтропия $H(A_i) = -\log_2 p(A_i)$; тогда энтропия опыта $H(\alpha)$ оказывается равной среднему значению $H(A_i)$ по всем возможным исходам:

$$H(\alpha) = \langle -\log_2 p(A^{(\alpha)}) \rangle = \langle H(A_i) \rangle, \quad i = 1, ..., n,$$

где $A^{(\alpha)}$ обозначает все исходы, возможные в опыте $\alpha$.

> ***Энтропия** является мерой неопределенности опыта, который реализуется через случайные события, и равна средней энтропии всех возможных его исходов.*

Для практики формула (2.4) важна тем, что позволяет сравнить неопределенности различных опытов со случайными исходами.

**Пример 2.1.**

Имеются два ящика, в каждом из которых по 12 шаров. В первом — 3 белых, 3 черных и 6 красных; во втором — каждого цвета по 4. Опыты состоят в вытаскивании по одному шару из каждого ящика. Что можно сказать относительно неопределенностей исходов этих опытов?

Согласно (2.4) находим энтропии обоих опытов:

$$H_\alpha = -\frac{3}{12}\log_2\frac{3}{12} - \frac{3}{12}\log_2\frac{3}{12} - \frac{6}{12}\log_2\frac{6}{12} = 1{,}50 \text{ бит};$$

$$H_\beta = -\frac{4}{12}\log_2\frac{4}{12} - \frac{4}{12}\log_2\frac{4}{12} - \frac{4}{12}\log_2\frac{4}{12} = 1{,}58 \text{ бит},$$

$H_\beta > H_\alpha$, т. е. неопределенность результата в опыте $\beta$ выше и, следовательно, предсказать его можно с меньшей долей уверенности, чем результат $\alpha$.

### 2.1.2. Свойства энтропии

1. Как следует из (2.4), $H = 0$ только в двух случаях:

a) какая-либо вероятность из $p(A_j) = 1$; однако при этом из (A.7) следует, что все остальные $p(A_i) = 0$, $i \neq j$, т. е. реализуется ситуация, когда один из исходов является *достоверным* (и общий итог опыта перестает быть случайным);

b) все $p(A_i) = 0$, т. е. никакие из рассматриваемых исходов опыта невозможны, поскольку нетрудно показать, что

$$\lim_{p\to 0}(p\log p) = 0.$$

Во всех остальных случаях $H > 0$.

2. Следствием (2.1) будет утверждение, что для двух *независимых* опытов $\alpha$ и $\beta$

$$H(\alpha \wedge \beta) = H(\alpha) + H(\beta). \tag{2.5}$$

**Энтропия сложного опыта, состоящего из нескольких независимых, равна сумме энтропий отдельных опытов.**

В справедливости (2.5) можно убедиться непосредственно. Пусть опыт $\alpha$ имеет $n$ исходов $A_1, A_2, \ldots A_n$, которые реализуются с вероятностями $p(A_1), p(A_2), \ldots, p(A_n)$, а событие $\beta$ — $m$ исходов $B_1, B_2, \ldots, B_m$ с вероятностями $p(B_1), p(B_2), \ldots, p(B_m)$. Сложный опыт $\alpha \wedge \beta$ имеет $nm$ исходов типа $A_iB_j$, $i = 1, \ldots, n$, $j = 1, \ldots, m$. Следовательно,

$$H(\alpha \wedge \beta) = -\sum_{i=1}^{n}\sum_{j=1}^{m} p(A_i \wedge B_j)\log_2 p(A_i \wedge B_j). \tag{2.6}$$

Поскольку $\alpha$ и $\beta$ — независимы, то независимыми окажутся события в любой паре $A_i \wedge B_j$. Тогда согласно (A.9)

$p(A_i \wedge B_j) = p(A_i)p(B_j)$; $\quad \log_2 p(A_i \wedge B_j) = \log_2 p(A_i) + \log_2 p(B_j)$

и

$$H(\alpha \wedge \beta) = -\sum_{i=1}^{n}\sum_{j=1}^{m} p(A_i)p(B_j)\{\log_2 p(A_i) + \log_2 p(B_j)\} =$$

$$= -\sum_{i=1}^{n}\sum_{j=1}^{m} p(A_i)p(B_j)\log_2 p(A_i) - \sum_{i=1}^{n}\sum_{j=1}^{m} p(A_i)p(B_j)\log_2 p(B_j) =$$

$$= -\sum_{i=1}^{n} p(A_i)\log_2 p(A_i)\sum_{j=1}^{m} p(B_j) - \sum_{j=1}^{m} p(B_j)\log_2 p(B_j)\sum_{i=1}^{n} p(A_i).$$

В слагаемых изменен порядок суммирования в соответствии со значениями индексов. Далее, по условию нормировки (А.7)

$$\sum_{j=1}^{m} p(B_j) = 1; \quad \sum_{i=1}^{n} p(A_j) = 1,$$

а из (2.4)

$$-\sum_{j=1}^{m} p(B_j)\log_2 p(B_j) = H(\beta); \quad -\sum_{i=1}^{n} p(A_i)\log_2 p(A_i) = H(\alpha),$$

и окончательно имеем

$$H(\alpha \wedge \beta) = H(\alpha) + H(\beta),$$

что и требовалось доказать.

3. Теперь рассмотрим ситуацию, когда имеется два опыта с одинаковым числом исходов $n$, но в одном случае они равновероятны, а в другом — нет. Каково соотношение энтропий опытов? Примем без доказательства* следующее утверждение:

$$-\sum_{i=1}^{n} p(A_i)\log_2 p(A_i) \leqslant \log_2 n. \tag{2.7}$$

**При прочих равных условиях наибольшую энтропию имеет опыт с равновероятными исходами.**

Другими словами, энтропия максимальна в опытах, где все исходы равновероятны. Здесь усматривается аналогия (имеющая глубинную первооснову!) с понятием энтропии, используемой в физике. Впервые понятие энтропии было введено в 1865 г. немецким физиком Рудольфом Клаузиусом как функции состояния термодинамической системы, определяющей направленность самопроизвольных процессов в системе. Клаузиус сформулировал 2-е начало термодинамики: *все самопроизвольные процессы в замкнутой термодинамической системе идут в направлении роста энтропии*; в состоянии равновесия системы энтропия

* При необходимости доказательство можно найти, например, в книгах А.М. Яглома и И.М. Яглома [49, с. 73–75]; Л. Бриллюэна [7, с. 34–36].

достигает максимума. Позднее (в 1872 г.) Людвиг Больцман, развивая статистическую теорию, связал энтропию системы с вероятностью ее состояния, дал статистическое (вероятностное) толкование 2-му началу термодинамики и, в частности, показал, что вероятность максимальна у полностью разупорядоченной (равновесной) системы, причем энтропия и термодинамическая вероятность оказались связанными логарифмической зависимостью. Другими словами, в физике энтропия оказывается *мерой беспорядка в системе*. При этом беспорядок понимается как *отсутствие знания* о характеристиках объекта (например, координат и скорости молекулы); с ростом энтропии уменьшается порядок в системе, т. е. наши знания о ней. Сходство понятий и соотношений между ними в теории информации и статистической термодинамике, как оказалось позднее, совершенно не случайно*.

Кстати, результат, полученный в рассмотренном выше примере 2.1, иллюстрирует справедливость формулы (2.7).

### 2.1.3. Условная энтропия

Найдем энтропию сложного опыта $\alpha \wedge \beta$ в том случае, если опыты не являются независимыми, т. е. если на исход $\beta$ оказывает влияние результат опыта $\alpha$. Например, если в ящике всего два разноцветных шара и $\alpha$ состоит в извлечении первого, а $\beta$ — второго из них, то $\alpha$ полностью снимает неопределенность сложного опыта $\alpha \wedge \beta$, т. е. оказывается $H(\alpha \wedge \beta) = H(\alpha)$, а не сумме энтропии, как следует из (2.5).

Связь между $\alpha$ и $\beta$ состоит в том, что какие-то из исходов $A^{(\alpha)}$ могут оказывать влияние на исходы из $B^{(\beta)}$, т. е. некоторые пары событий $A_i \wedge B_j$ не являются независимыми. Но тогда в (2.6) $p(A_i \wedge B_j)$ следует заменять не произведением вероятностей, а согласно (А.14)

$$p(A_i \wedge B_j) = p(A_i) p_{A_i}(B_j)$$

где $p_{A_i}(B_j)$ — вероятность наступления исхода $B_j$ при условии, что в первом опыте имел место исход $A_i$. Тогда

$$\log_2 p(A_i \wedge B_j) = \log_2 p(A_i) + \log_2 p_{A_i}(B_j).$$

При подстановке в (2.6) получаем

$$H(\alpha \wedge \beta) = -\sum_{i=1}^{n} \sum_{j=1}^{m} p(A_i) p_{A_i}(B_j) \{\log_2 p(A_i) + \log_2 p_{A_i}(B_j)\} =$$

---

* Подробнее об этом можно прочитать в книгах Л. Бриллюэна [7] и Р.Л. Стратоновича [39].

$$= -\sum_{i=1}^{n}\sum_{j=1}^{m} p(A_i)p_{A_i}(B_j)\log_2 p(A_i) -$$
$$-\sum_{i=1}^{n}\sum_{j=1}^{m} p(A_i)p_{A_i}(B_j)\log_2 p_{A_i}(B_j) =$$
$$= -\underbrace{\sum_{i=1}^{n} p(A_i)\log_2 p(A_i)}_{H_{A_i}(\beta)}\underbrace{\sum_{j=1}^{m} p_{A_i}(B_j)}_{1} -$$
$$-\underbrace{\sum_{i=1}^{n} p(A_i)\sum_{j=1}^{m} p_{A_i}(B_j)\log_2 p_{A_i}(B_j)}_{H(\alpha)} = H_{A_i}(\beta) - H(\alpha).$$

В первом слагаемом индекс $j$ имеется только у $B$; изменив порядок суммирования, получим члены вида

$$\sum_{j=1}^{m} p_{A_i}(B_j).$$

Их смысл — сумма вероятностей всех исходов опыта $\beta$ при том, что ему предшествовал исход $A_i$ в опыте $\alpha$. Ясно, что для любого $A_i$ какой-либо из исходов опыта $\beta$ все равно реализуется; тогда из условия нормировки вероятностей

$$\sum_{j=1}^{m} p_{A_i}(B_j) = 1,$$

следовательно, первое слагаемое оказывается равным

$$-\sum_{i=1}^{n} p(A_i)\log_2 p(A_i) = H(\alpha).$$

Во втором слагаемом члены вида

$$-\sum_{j=1}^{m} p_{A_i}(B_j)\log_2 p_{A_i}(B_j) = H_{A_i}(\beta) \tag{2.8}$$

имеют смысл энтропии опыта $\beta$ при условии, что в опыте $\alpha$ реализовался исход $A_i$ — будем называть ее *условной энтропией*. Если ввести данное понятие и использовать его обозначение, то второе слагаемое будет иметь вид

$$\sum_{i=1}^{n} p(A_i)H_{A_i}(\beta) = H_\alpha(\beta), \tag{2.9}$$

где $H_\alpha(\beta)$ есть *средняя условная энтропия опыта $\beta$ при условии выполнении опыта $\alpha$*. Окончательно получаем для энтропии сложного опыта

$$H(\alpha \wedge \beta) = H(\alpha) + H_\alpha(\beta). \tag{2.10}$$

Полученное выражение представляет собой общее правило нахождения энтропии сложного опыта. Совершенно очевидно, что выражение (2.5) является частным случаем (2.10) при условии независимости опытов $\alpha$ и $\beta$.

Относительно условной энтропии можно высказать следующие утверждения:

1. Условная энтропия является величиной *неотрицательной*. $H_\alpha(\beta) = 0$ только в том случае, если *любой* исход $\alpha$ полностью определяет исход $\beta$ (как в примере с двумя шарами), т. е.

$$H_{A_i}(\beta) = H_{A_2}(\beta) = \ldots = H_{A_n}(\beta) = 0.$$

В этом случае $H(\alpha \wedge \beta) = H(\alpha)$.

2. Если опыты $\alpha$ и $\beta$ независимы, то $H_\alpha(\beta) = H(\beta)$, причем это оказывается *наибольшим* значением условной энтропии. Другими словами, опыт $\alpha$ не может повысить неопределенность опыта $\beta$; он может либо не оказать никакого влияния (если опыты независимы), либо понизить энтропию $\beta$.

Приведенные утверждения можно объединить одним неравенством:

$$0 \leqslant H_\alpha(\beta) \leqslant H(\beta), \tag{2.11}$$

т. е. *условная энтропия не превосходит безусловную*.

3. Из соотношений (2.10) и (2.11) следует, что

$$H(\alpha \wedge \beta) \leqslant H(\alpha) + H(\beta), \tag{2.12}$$

причем равенство реализуется только в том случае, если опыты $\alpha$ и $\beta$ независимы.

**Пример 2.2.**

В ящике имеются 2 белых шара и 4 черных. Из ящика извлекают последовательно два шара без возврата. Найти энтропию, связанную с первым и вторым извлечениями, а также энтропию обоих извлечений.

Будем считать опытом $\alpha$ извлечение первого шара. Он имеет два исхода: $A_1$ — вынут белый шар; его вероятность $p(A_1) = 2/6 = 1/3$; исход $A_2$ — вынут черный шар; его вероятность $p(A_2) = 1 - p(A_1) = 2/3$. Эти данные позволяют с помощью (2.4) сразу найти $H(\alpha)$:

$$H(\alpha) = -p(A_1)\log_2 p(A_1) - p(A_2)\log_2 p(A_2) =$$
$$= -\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} = 0{,}918 \text{ бит}.$$

Опыт $\beta$ — извлечение второго шара также имеет два исхода: $B_1$ — вынут белый шар; $B_2$ — вынут черный шар, однако их вероятности будут зависеть от того, каким был исход опыта $\alpha$. В частности:

$$\text{при } A_1: \quad p_{A_1}(B_2) = \frac{4}{5} p_{A_1}(B_1) = \frac{1}{5};$$
$$\text{при } A_2: \quad p_{A_2}(B_2) = \frac{3}{5} p_{A_2}(B_1) = \frac{2}{5}.$$

Следовательно, энтропия, связанная со вторым опытом, является условной и согласно (2.8) и (2.9)

$$H_{A_1}(\beta) = -\frac{1}{5}\log_2\frac{1}{5} - \frac{4}{5}\log_2\frac{4}{5} = 0{,}722 \text{ бит};$$
$$H_{A_2}(\beta) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0{,}971 \text{ бит};$$
$$H_\alpha(\beta) = p(A_1)H_{A_1}(\beta) + p(A_2)H_{A_2}(\beta) =$$
$$= \frac{1}{3}\cdot 0{,}722 + \frac{2}{3}\cdot 0{,}971 = 0{,}888 \text{ бит}.$$

Наконец, из (2.10) $H(\alpha \wedge \beta) = 0{,}918 + 0{,}888 = 1{,}806$ бит.

**Пример 2.3.**

Имеется три тела с одинаковыми внешними размерами, но с разными весами $x_1$, $x_2$ и $x_3$. Необходимо определить энтропию, связанную с нахождением наиболее тяжелого из них, если сравнивать веса тел можно только попарно.

Последовательность действий достаточно очевидна: сравниваем вес двух любых тел, определяем из них более тяжелое, затем с ним сравниваем вес третьего тела и выбираем наибольший из них. Поскольку внешне тела неразличимы, выбор номеров тел при взвешивании будет случаен, однако общий результат от этого выбора не зависит. Пусть опыт $\alpha$ состоит в сравнении веса двух тел, например 1-го и 2-го. Этот опыт, очевидно, может иметь два исхода: $A_1$ — $x_1 > x_2$; его вероятность $p(A_1) = 1/2$; исход $A_2$ — $x_1 < x_2$; также его вероятность $p(A_2) = 1/2$. Получаем

$$H(\alpha) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1 \text{ бит}.$$

Опыт $\beta$ — сравнение весов тела, выбранного в опыте $\alpha$, и 3-го — имеет четыре исхода: $B_1$ — $x_1 > x_3$, $B_2$ — $x_1 < x_3$, $B_3$ — $x_2 > x_3$, $B_4$ — $x_2 < x_3$; вероятности исходов зависят от реализовавшегося исхода $\alpha$ — для удобства представим их в виде таблицы:

| | $B_1$ | $B_2$ | $B_3$ | $B_4$ |
|---|---|---|---|---|
| $A_1$ | 1/2 | 1/2 | 0 | 0 |
| $A_2$ | 0 | 0 | 1/2 | 1/2 |

Вновь, воспользовавшись формулами (2.8) и (2.9) и с учетом свойства (1) п. 2.1.2, находим:

$$H_{A_2}(\beta) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1 \text{ бит};$$

$$H_{A_1}(\beta) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1 \text{ бит};$$

$$H_\alpha(\beta) = p(A_1)H_{A_1}(\beta) + p(A_2)H_{A_2}(\beta) = \frac{1}{2}\cdot 1 + \frac{1}{2}\cdot 1 = 1 \text{ бит}.$$

Следовательно, энтропия сложного опыта, т. е. всей процедуры испытаний

$$H(\alpha \wedge \beta) = H(\alpha) + H_\alpha(\beta) = 2 \text{ бита}.$$

## 2.2. Энтропия и информация

Поучительность только что рассмотренного примера в том, что из него отчетливо видно, как предшествующий опыт ($\alpha$) может уменьшить количество исходов и, следовательно, неопределенность последующего опыта ($\beta$). Разность $H(\beta)$ и $H_\alpha(\beta)$, очевидно, показывает, какие новые сведения относительно $\beta$ мы получаем, произведя опыт $\alpha$. Эта величина называется *информацией относительно опыта $\beta$, содержащейся в опыте $\alpha$*:

$$I(\alpha, \beta) = H(\beta) - H_\alpha(\beta). \tag{2.13}$$

Данное выражение открывает возможность *вычисления количества информации*, поскольку оценивать энтропию мы уже умеем.

**Пример 2.4.**

В урне находятся 4 белых и 4 черных шара. Случайным образом извлекается один шар. Какое количество информации связано с его извлечением?

Если бы извлечения не было (опыт $\beta$), вероятности получения шаров составляли бы $p_б = p_ч = 1/2$; тогда энтропия согласно (2.4) была бы равна

$$H(\beta) = -p_б \log_2 p_б - p_ч \log_2 p_ч = 2 \text{ бита}.$$

После извлечения шара (опыт $\alpha$) вероятности получения шаров во втором опыте ($\beta$) изменятся: если первый шар был белым, то $p_б = 3/7$; $p_ч = 4/7$ и

$$H_б(\beta) = -\frac{3}{7}\log_2\frac{3}{7} - \frac{4}{7}\log_2\frac{4}{7} = 2{,}03\text{бит};$$

если первый шар был черным, то $p_б = 4/7$; $p_ч = 3/7$ и $H_ч(\beta) = 2{,}03$ бит.

Условная энтропия

$$H_\alpha(\beta) = \frac{3}{8}H_б(\beta) + \frac{3}{8}H_ч(\beta) = 1{,}52 \text{ бит};$$

$$I(\alpha, \beta) = H(\beta) - H_\alpha(\beta) = 0{,}48 \text{ бит}.$$

Таким образом, с извлечением первого шара связана информация 0,48 бит.

Из (2.13) легко получить ряд следствий:

**Следствие 1**. Поскольку единицей измерения энтропии является *бит*, то в этих же единицах может быть измерено количество информации.

**Следствие 2**. Пусть опыт $\alpha = \beta$, т. е. просто произведен опыт $\beta$. Поскольку он несет полную информацию о себе самом, неопределенность его исхода полностью снимается, т. е. $H_\beta(\beta) = 0$. Тогда $I(\beta, \beta) = H(\beta)$, т. е. можно считать, что

**энтропия равна информации относительно опыта, которая содержится в нем самом.**

Можно построить уточнение:

**Энтропия опыта равна той информации, которую мы получаем в результате его осуществления.**

Отметим ряд свойств информации:

1. $I(\alpha, \beta) \geqslant 0$, причем $I(\alpha, \beta) = 0$ тогда и только тогда, когда опыты $\alpha$ и $\beta$ независимы. Это свойство непосредственно вытекает из (2.10) и (2.13).

2. Легко доказать, что $I(\alpha, \beta) = I(\beta, \alpha)$ (см. контрольное задание 11), т. е. информация симметрична относительно последовательности опытов — именно по этой причине $I(\alpha, \beta)$ называют также *взаимной информацией*.

3. Рассмотрим два конкретных события (исхода) — в опыте $\alpha$ пусть это будет $A_i$, а в опыте $\beta$ — $B_j$. В соответствии с определением (2.13) можно записать, что взаимная информация между $A_i$ и $B_j$

$$I(A_i, B_j) = H(B_j) - H_{A_i}(B_j).$$

Однако для отдельного исхода $B_j$ энтропия $H(B_j) = -\log_2 p(B_j)$. Аналогично $H_{A_i}(B_j) = -\log_2 p_{A_i}(B_j)$.

Следовательно, признаком того, что событие $A_i$ не содержит никакой информации относительно события $B_j$ (т. е. между ними отсутствует взаимная информация), будет равенство условной и безусловной вероятности:

$$p(B_j) = p_{A_i}(B_j).$$

4. Следствие 2 и представление энтропии в виде (2.4) позволяют записать для опыта $\alpha$

$$I(\alpha) = -\sum_{i=1}^{n} p(A_i) \log_2 p(A_i). \tag{2.14}$$

Из этого выражения, в частности, вытекает ряд следствий:

а) информация, связанная с опытом, есть величина неотрицательная ($I(\alpha) \geqslant 0$); информация оказывается равной 0 только в случаях, если все исходы опыта невероятны или один из них достоверен, остальные невероятны;

б) из аддитивности энтропии вытекает аддитивность информации, т. е. для независимых опытов $I(\gamma \wedge \delta) = I(\gamma) + I(\delta)$.

в) (2.14) можно дать другую интерпретацию — считать, что с каждым исходом $A_i$ связана информация $I(A_i) = -\log_2 p(A_i)$; тогда (2.14) можно переписать:

$$I(\alpha) = \sum_{i=1}^{n} p(A_i) I(A_i) = \langle I(A^{(\alpha)}) \rangle,$$

т. е.

**информация опыта равна среднему значению количества информации, содержащейся во всех отдельных его исходах.**

5. Из (2.14) для ситуации, когда все $n$ исходов опыта равновероятны и, следовательно, $p(A_i) = 1/n$, получаем

$$I(\alpha) = -\sum_{i=1}^{n} \frac{1}{n} \log_2 \frac{1}{n} = \log_2 n; \quad I = \log_2 n. \tag{2.15}$$

Эта формула была выведена в 1928 г. американским инженером Р. Хартли и носит его имя. Она связывает *количество равновероятных исходов ($n$) и количество информации в сообщении ($I$) о том, что любой из этих исходов реализовался*. Ее смысл в том, что, если некоторое множество содержит $n$ элементов и $x$ принадлежит данному множеству, то для его выделения (однозначной идентификации) среди прочих требуется количество информации, равное $\log_2 n$.

6. Выражение (2.13) можно интерпретировать следующим образом: если начальная энтропия опыта $H_1$, а в результате сообщения информации $I$ энтропия становится равной $H_2$ (очевидно, $H_1 \geqslant H_2$), то

$$I = H_1 - H_2,$$

т. е. *информация равна убыли энтропии*. В частном случае, если изначально равновероятных исходов было $n_1$, а в результате передачи информации $I$ неопределенность уменьшилась и число исходов стало $n_2$ (естественно, $n_2 \leqslant n_1$), то из (2.15) легко получить:

$$I = \log_2 n_1 - \log_2 n_2 = \log_2 \frac{n_1}{n_2}.$$

Таким образом, можно построить следующее определение:

***Информация** — это содержание сообщения, понижающего неопределенность некоторого опыта с неоднозначными исходами; убыль связанной с ним энтропии является количественной мерой информации.*

Рассмотрим ряд примеров применения формул (2.14)–(2.15).

**Пример 2.5.**

Какое количество информации требуется, чтобы узнать исход броска монеты?

В данном случае $n = 2$ и события равновероятны, т. е. $p_1 = p_2 = 0{,}5$. Согласно формуле Хартли (2.15)

$$I = \log_2 2 = 1 \text{ бит}.$$

**Пример 2.6.**

Игра «*Угадай-ка-4*». Некто задумал целое число в интервале от 0 до 3. Наш опыт состоит в угадывании этого числа. На наши вопросы Некто может отвечать лишь «*Да*» или «*Нет*». Какое количество информации мы должны получить, чтобы узнать задуманное число, т. е. полностью снять начальную неопределенность? Как правильно построить процесс угадывания?

Исходами в данном случае являются: $A_1$ — «*задуман 0*», $A_2$ — «*задумана 1*», $A_3$ — «*задумана 2*», $A_4$ — «*задумана 3*». Конечно, предполагается, что вероятности быть задуманными у всех чисел одинаковы. Поскольку $n = 4$, следовательно, $p(A_i) = 1/4$, $\log_2 p(A_i) = -2$ и $I = 2$ бита. Таким образом, для полного снятия неопределенности опыта (угадывания задуманного числа) нам необходимо 2 бита информации.

Теперь можно задуматься, какие вопросы и в какой последовательности необходимо задать, чтобы процесс угадывания был оптимальным, т. е. содержал минимальное их число. Непосредственно из теории информации способ никак не вытекает (теория проделала свою работу — дала количественную оценку, но не способ!). Конечно, можно пойти путем перебора: «Это 0?», «Это 1?» и т. д. В лучшем случае ответ будет получен с 1-го раза, в худшем — с 4-го. Едва ли этот вариат можно считать оптимальным. Для решения задачи следует воспользоваться *методом половинного деления* (так он называется в математике) или *выборочным каскадом* (рис. 2.1).

Таким образом, задача гарантированно решается посредством двух вопросов независимо от того, какое число было задумано. Совпадение между количеством информации и числом вопросов с бинарными ответами неслучайно.

В рассмотренном примере два полученных ответа в выборочном каскаде полностью сняли начальную неопределенность. *Подобная процедура позволяет определить количество информации в любой задаче, интерпретация которой может*

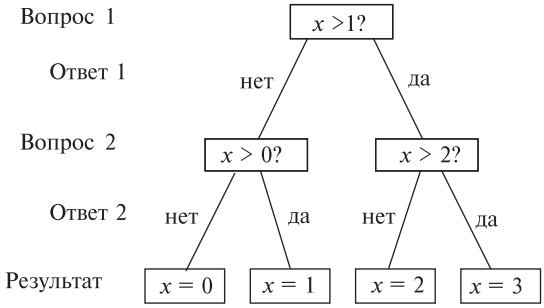



**Рис. 2.1.** Выборочный каскад

*быть сведена к парному выбору*. Например, определение символа некоторого алфавита, использованного для представления сообщения. Приведенное утверждение перестает быть справедливым в том случае, если каждый из двух возможных ответов имеет разную вероятность — такая ситуация будет рассмотрена позднее.

**Количество информации численно равно* числу вопросов с равновероятными бинарными вариантами ответов, которые необходимо задать, чтобы полностью снять неопределенность задачи.**

Очевидно, такое равенство будет справедливо только в случае, если число исходов равно целочисленной степени 2 ($n = 2^k$). Тогда из формулы Хартли

$$I = \log_2 2^k = k, \tag{2.16}$$

т. е. $I = k$ бит.

Именно эта ситуация реализовалась в рассмотренных ранее примерах 2.4 и 2.5. Анализируя результаты решения, можно прийти к выводу, что $k$ как раз равно количеству вопросов с бинарными равновероятными ответами, которые определяли количество информации в задачах.

**Пример 2.7.**

Случайным образом вынимается карта из колоды в 32 карты. Какое количество информации требуется, чтобы угадать, что это за карта? Как построить угадывание?

---

* Речь идет только о равенстве числовых значений, а не о полном равенстве, поскольку количество информации имеет размерность (бит), а число вопросов — величина безразмерная.

Для данной ситуации $n = 2^5$, значит, $k = 5$ и, следовательно, $I = 5$ бит. Последовательность вопросов придумайте самостоятельно.

**Пример 2.8.**

В некоторой местности имеются две близкорасположенные деревни: А или В. Известно, что жители А всегда говорят правду, а жители В — всегда лгут. Известно также, что жители обеих деревень любят ходить друг к другу в гости, поэтому в каждой из деревень можно встретить жителя соседней деревни. Путешественник, сбившись ночью с пути оказался в одной из двух деревень и, заговорив с первым встречным, захотел выяснить, в какой деревне он находится и откуда его собеседник. Какое минимальное количество вопросов с бинарными ответами требуется задать путешественнику?

Количество возможных комбинаций, очевидно, равно 4 (путешественник в А, собеседник из А; путешественник в А, собеседник из В; и т. д.), т. е. $n = 2^2$ и, следовательно, $k = 2$. Порядок вопросов придумайте самостоятельно.

Как уже отмечалось, во всех рассмотренных примерах посредством формул (2.14)–(2.16) можно определить количество информации, необходимое для решения задачи, но они не указывают способ решения.

Попытаемся понять смысл полученных в данном разделе результатов. Необходимо выделить ряд моментов.

1. Выражение (2.14) является *статистическим* определением понятия «*информация*», поскольку в него входят вероятности возможных исходов опыта. По сути дано *операционное* определение новой величины, т. е. устанавлена *процедура* (способ) *вычисления* величины. Как отмечалось ранее, в науке (научном знании) именно такой метод введения новых терминов считается предпочтительным, поскольку то, что не может быть измерено, не может быть проверено и, следовательно, заслуживает меньшего доверия.

2. Начнем с рассмотрения примера.

**Пример 2.9.**

Отгадывается случайным образом задуманное двузначное число. Первый раз методом половинного деления, во второй раз — по 1-й и 2-й цифре. Как соотносятся количества информации, необходимые для решения задачи (полного снятия неопределенности)?

При отгадывании числа целиком осуществляется выбор из $n = 90$ равновероятных исходов; следовательно, $I^{(1)} = \log_2 90$.

Для отгадывания 1-й цифры нужно осуществить выбор из 9 возможностей (от 1 до 9), 2-й цифры — выбор из 10 возможностей (от 0 до 9);

поскольку 1-я и 2-я цифры независимы, можно воспользоваться аддитивностью информации:

$$I^{(2)} = \log_2 9 + \log_2 10 = \log_2 90,$$

т. е. $I^{(1)} = I^{(2)}$.

Полученный результат неслучаен и несет глубокий смысл:

**Количество информации является характеристикой задачи и не зависит от способа ее решения.**

В этом отношении информация схожа с энергией в физике — она также характеризует состояние системы и не зависит от того, как система в это состояние перешла.

3. Вернемся к утверждению о том, что количество информации может быть измерено числом вопросов с двумя равновероятными ответами ответа (будем далее называть такие вопросы *бинарными*). Означает ли это, что $I$ должна быть всегда величиной целой? Из формулы (2.16), как было показано, $I = k$ бит (т. е. целому числу бит) только в случае $n = 2^k$. А в остальных ситуациях? Например, при игре «*Угадай-ка-7*» (угадать число от 0 до 6) нужно было бы в выборочном каскаде задать $k \geqslant \log_2 7 = 2{,}807$, т. е. необходимо задать три вопроса, поскольку количество вопросов — это целое число. Однако представим, что мы одновременно играем в шесть таких же игр. Тогда необходимо отгадать одну из возможных комбинаций, которых всего $N = n_1 n_2 \cdots n_6 = 7^6 = 117649 < 2^{17} = 131072$. Следовательно, для угадывания всей шестизначной комбинации требуется $I = 17$ бит информации, т. е. нужно задать 17 вопросов. *В среднем* же на один элемент (одну игру) приходится $17/6 = 2{,}833$ вопроса, что близко к значению $\log_2 7$. Таким образом,

**Величина $I$, определяемая по числу ответов, показывает, сколько *в среднем* необходимо сделать парных выборов для установления результата (полного снятия неопределенности), если опыт повторить многократно ($n \to \infty$): $I = \lim\limits_{n\to\infty} \langle k \rangle$.**

Для отдельного угадывания (распознавания) необходимо задать $k \geqslant \log_2 n$ бинарных вопросов (дробное число округляется до ближайшего целого, превышающего $\log_2 n$).

4. Необходимо понимать также, что не всегда с каждым из ответов на вопрос, имеющий только два варианта, связан ровно 1 бит информации. Рассмотрим опыт, реализующийся посредством двух случайных событий; поскольку их всего два, очевидно,

они являются дополнительными друг другу. Если эти события равновероятны, то $p_1 = p_2 = 1/2$ и $I = 1$ бит, как следует из формулы Хартли и (2.14). Однако, если их вероятности различны: $p_1 = p$, то согласно (А.8) $p_2 = 1 - p$ и из (2.14) получаем выражение для $I$, которое можно рассматривать как кофункцию $p$:

$$I(p) = -p \log_2 p - (1 - p) \log_2 (1 - p).$$

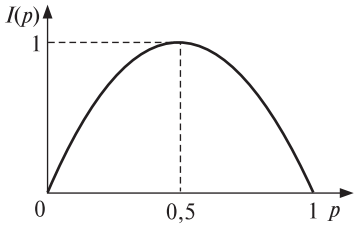



**Рис. 2.2.** График функции $I(p)$

Легко показать, что при $p \to 0$ и при $p \to 1$ функция $I(p) \to 0$. Ситуация может быть проиллюстрирована графиком (рис. 2.2), из которого, в частности, видно, что кривая симметрична относительно $p = 0{,}5$ и достигает максимума при этом значении.

Если теперь считать, что событие *1* — это утвердительный ответ на бинарный вопрос, а событие *2* — отрицательный, то мы приходим к заключению:

**Ответ на бинарный вопрос может содержать не более 1 бит информации; информация равна 1 бит только для равновероятных ответов; в остальных случаях она меньше 1 бит.**

**Пример 2.10.**

При угадывании результата броска игральной кости задается вопрос «*Выпало 6?*». Какое количество информации содержит ответ?

$p = 1/6$, $1 - p = 5/6$, следовательно, из (2.14)

$$I = -1/6 \log_2 1/6 - 5/6 \log_2 5/6 \approx 0{,}65 \text{ бит} < 1 \text{ бит}.$$

Из этого примера видна также *ошибочность* утверждения, встречающегося в некоторых учебниках информатики, что 1 бит является минимальным количеством информации — никаких оснований для такого утверждения в теории информации нет.

5. Формула (2.14) приводит нас еще к одному выводу. Пусть некоторый опыт имеет два исхода $A$ и $B$, причем $p_A = 0{,}99$, а $p_B = 0{,}01$. В случае исхода $A$ мы получим количество информации $I_A = -\log_2 0{,}99 = 0{,}0145$ бит. В случае исхода $B$ количество информации оказывается равным $I_B = -\log_2 0{,}01 = 6{,}644$ бит. Другими словами, *больше информации связано с теми исходами, которые менее вероятны.* Действительно, то, что наступит именно $A$, мы почти наверняка знали и до опыта; поэтому реализация такого исхода очень мало добавляет к нашей

осведомленности. Наоборот, исход $B$ — весьма редкий; информации с ним связано больше (осуществилось трудно ожидаемое событие). Однако такое большое количество информации мы будем при повторах опыта получать редко, поскольку мала вероятность $B$. Среднее же количество информации согласно (2.14) $I = 0{,}99I_A + 0{,}01I_B \approx 0{,}081$ бит.

6. Нами рассмотрен *вероятностный* подход к определению количества информации. Он не является единственным. Как будет показано далее, количество информации можно связать с числом знаков в дискретном сообщении — такой способ измерения называется *объемным*. Можно доказать, что при любом варианте представления информации $I_{\text{вер}} \leqslant I_{\text{об}}$.

7. Объективность информации. При использовании людьми одна и та же информация может иметь различную оценку с точки зрения значимости (важности, ценности). Определяющим в такой оценке оказывается содержание (смысл) сообщения для конкретного потребителя. Однако нельзя предложить единой для всех меры ценности информации. При решении же задач технического характера содержание сообщения может не играть роли. Например, задача телеграфной (и любой другой) линии связи состоит в точной и безошибочной передаче сообщения без анализа того, насколько ценной для получателя оказывается связанная с сообщением информация. Техническое устройство не может оценить важности информации — его задача без потерь передать или сохранить информацию.

Выше мы определили информацию как результат выбора. Такое определение не зависит от того, кто и каким образом осуществляет выбор, и связанная с ним количественная мера информации будет одинаковой для любого потребителя. Следовательно, появляется возможность *объективного* измерения информации, при этом результат измерения абсолютен. Это служит предпосылкой для решения технических задач. Определение количества информации основано на вероятностях, а не на смысле, который могут нести для человека знаки сообщения. Жертвуя смысловой (семантической) стороной информации, мы получаем объективные методы измерения ее количества, а также обретаем возможность описывать информационные процессы математическими уравнениями. Это является приближением и в то же время условием применимости законов теории информации к анализу и описанию информационных процессов.

8. Информация и знание. На бытовом уровне, в науках социальной направленности, например в педагогике, «*информация*» отождествляется с «*информированностью*», т. е. человеческим *знанием*, которое, в свою очередь, связано с оценкой смысла информации. В теории информации же, напротив, информация является *мерой нашего незнания* чего-либо (но что в принципе может произойти); как только это происходит и мы узнаем результат, информация, связанная с данным событием, исчезает. Состоявшееся событие не несет информации, поскольку пропадает его неопределенность (энтропия становится равной нулю) и согласно (2.13) $I = 0$. Если поставить вопрос: «Какая книга содержит больше всего информации» (при фиксированном числе знаков)? Ответ теории информации будет неожиданным с точки зрения «здравого смысла»: «Больше всего информации содержит книга, текст которой состоит из полностью случайного набора знаков» — если каждый знак текста будет обладать наибольшей энтропией, то с этим текстом оказывается связаной максимальная информация.

Последние два замечания представляются весьма важными, поскольку недопонимание указанных в них обстоятельств порождает попытки применения законов теории информации в тех сферах, где условия ее применимости не выполнены. Это, в свою очередь, порождает отрицательные результаты, которые служат причиной разочарования в самой теории. Однако любая теория, в том числе и теория информации, справедлива лишь в рамках своих исходных ограничений. С иными, в том числе философскими, аспектами понятия «*информация*» можно ознакомиться в книге А.В. Могилева, Н.И. Пака, Е.К. Хеннера [30, с. 27–30]; в ней же приводится список литературы по данному вопросу.

## 2.3. Информация и алфавит

Рассматривая формы представления информации, мы отметили то обстоятельство, что, хотя естественной для органов чувств человека является аналоговая форма, универсальной все же следует считать дискретную форму представления информации с помощью некоторого набора знаков. В частности, именно таким образом представленная информация обрабатывается компьютером, передается по компьютерным и некоторым иным линиям связи. Сообщение есть последовательность знаков алфавита. При их передаче возникает проблема *распознавания знака*:

каким образом *прочитать* сообщение, т. е. по полученным сигналам установить исходную последовательность знаков первичного алфавита. В устной речи это достигается использованием различных фонем (основных звуков разного звучания), по которым мы и отличаем знаки речи. В письменности это достигается различным начертанием букв и дальнейшим нашим анализом написанного. Как данная задача может решаться техническим устройством, мы рассмотрим позднее. Сейчас для нас важно, что можно реализовать некоторую процедуру (механизм), посредством которой выделить из сообщения тот или иной знак. Но появление конкретного знака (буквы) в конкретном месте сообщения — событие случайное. Следовательно, узнавание (отождествление) знака требует получения некоторой порции информации. Можно связать эту информацию с самим знаком и считать, что знак несет в себе (содержит) определенное количество информации. Попробуем оценить это количество.

Начнем с самого грубого приближения (будем называть его *нулевым*, что отражается индексом у получаемых величин) — предположим, что появление всех знаков (букв) алфавита в сообщении *равновероятно*. Тогда для английского алфавита $n_e = 27$ (с учетом пробела как самостоятельного знака); для русского алфавита $n_r = 34$. Из формулы Хартли (2.15) находим

$$I_0^{(r)} = \log_2 27 = 4{,}755 \text{ бит}; \quad I_0^{(e)} = \log_2 34 = 5{,}087 \text{ бит}.$$

Получается, что в нулевом приближении со знаком русского алфавита *в среднем* связано больше информации, чем со знаком английского. Например, в русской букве «$a$» информации больше, чем в «$a$» английской! Это, безусловно, не означает, что английский язык — язык Шекспира и Диккенса — беднее, чем язык Пушкина и Достоевского. Лингвистическое богатство языка определяется количеством слов и их сочетаний, а это никак не связано с числом букв в алфавите. С точки зрения техники это означает, что сообщения из равного количества знаков будет иметь разную длину (и соответственно, время передачи) и большими они окажутся у сообщений на русском языке.

В качестве следующего (*первого*) приближения, уточняющего исходное, попробуем учесть то обстоятельство, что относительная частота, т. е. вероятность появления различных букв в тексте (или сообщении) различна. Эти вероятности получили название *априорных*; они определяются только особенностями языка, используемого источником.

Таблица 2.1

| Буква | $p_i$ | Буква | $p_i$ | Буква | $p_i$ | Буква | $p_i$ |
|---|---|---|---|---|---|---|---|
| Пробел | 0,174 | р | 0,040 | я | 0,018 | х | 0,009 |
| о | 0,090 | в | 0,038 | ы | 0,016 | ж | 0,007 |
| е | 0,072 | л | 0,035 | з | 0,016 | ю | 0,006 |
| а | 0,062 | к | 0,028 | ь, ъ | 0,014 | ш | 0,006 |
| и | 0,062 | м | 0,026 | б | 0,014 | ц | 0,004 |
| т | 0,053 | д | 0,025 | г | 0,013 | щ | 0,003 |
| н | 0,053 | п | 0,023 | ч | 0,012 | э | 0,003 |
| с | 0,045 | у | 0,021 | й | 0,010 | ф | 0,002 |

Рассмотрим таблицу средних частот букв для русского алфавита, в который включен также знак «пробел» для разделения слов (из книги А.М. Яглома и И.М. Яглома [50, с. 238]); с учетом неразличимости букв «е» и «ё», а также «ь» и «ъ» (так принято в телеграфном кодировании), получим алфавит из 32 знаков со вероятностями их появления в русских текстах (табл. 2.1).

Для оценки информации, связанной с выбором одного знака алфавита с учетом неравной вероятности их появления в сообщении (текстах), можно воспользоваться формулой (2.14). Из нее, в частности, следует, что если $p_i$ — вероятность (относительная частота) знака номер $i$ данного алфавита из $N$ знаков, то *среднее количество информации, приходящейся на один знак*,

$$I = -\sum_{i=1}^{n} p_i \log_2 p_i. \tag{2.17}$$

Это и есть знаменитая формула К. Шеннона*, с работы которого «Математическая теория связи» (1948) принято начинать отсчет возраста информатики как самостоятельной науки [46]. Объективности ради следует заметить, что и в нашей стране практически одновременно с Шенноном велись подобные исследования, например в том же 1948 г. вышла работа А.Н. Колмогорова «Математическая теория передачи информации».

Применение формулы (2.17) к алфавиту русского языка дает значение средней информации на знак $I_1^{(r)} = 4{,}36$ бит, а для английского языка $I_1^{(e)} = 4{,}04$ бит, для французского $I_1^{(f)} = 3{,}96$ бит,

---

* На самом деле формула Шеннона, как и формула Хартли, изначально была записана для энтропии. Однако для нашего изложения более удобной представляется форма записи через понятие информации.

для немецкого $I_1^{(d)} = 4{,}10$ бит, для испанского $I_1^{(s)} = 3{,}98$ бит. Как мы видим, и для русского, и для английского языков учет вероятностей появления букв в сообщениях приводит к уменьшению среднего *информационного содержания* буквы, что, кстати, подтверждает справедливость формулы (2.7). Несовпадение значений средней информации для английского, французского и немецкого языков, основанных на одном алфавите, связано с тем, что частоты появления одинаковых букв в них различаются.

В рассматриваемом приближении по умолчанию предполагается, что вероятность появления любого знака в любом месте сообщения остается одинаковой и не зависит от того, какие знаки или их сочетания предшествуют данному. Такие сообщения называются *шенноновскими* (или *сообщениями без памяти*).

> *Сообщения, в которых вероятность появления любого отдельного знака алфвита в любом месте сообщения остается неизменной и не зависит от того, какие знаки предшествовали данному, называются* ***шенноновскими****, а порождающий их отправитель —* ***шенноновским источником****.*

Каким образом можно проверить, является ли некоторое сообщение шенноновским? Необходимо произвести частотное исследование текста и определить вероятности (относительные частоты) каждого из $N$ знаков алфавита $p_i$, $i = 1, \ldots, N$, а также вероятности появления всех возможных двухбуквенных сочетаний $p_{ik}^{(\mathrm{fact})}$, $k = 1, \ldots, N$. Если случайное событие — появление в тексте знака $a_i$ — не оказывает влияние на другое случайное событие — появление после него знака $a_k$, то согласно (А.9) должно выполняться

$$p_{ik}^{(\mathrm{rand})} = p_i p_k. \tag{2.18}$$

Критерием того, что сообщение является шенноновским, очевижно будет выполнение соотношения $p_{ik}^{(\mathrm{fact})} \approx p_{ik}^{(\mathrm{rand})}$ *для всех пар* $ik$. Равенства могут не быть строгими из-за конечности длины текста.

Заметные отличия этих двух величин будут свидетельствовать о несправедливости модели независимых знаков в сообщении. Это, в свою очередь, означает, что между знаками имеются связи и появление (или непоявление) какого-то знака оказывает влияние на вероятность появления последующего знака.

Если сообщение оказывается шенноновским, то набор знаков (алфавит) и вероятности их появления в сообщении могут

считаться известными заранее. В этом случае, с одной стороны, можно предложить *оптимальные способы кодирования*, уменьшающие суммарную длину сообщения при передаче по каналу связи. С другой стороны, интерпретация сообщения, представляющего собой последовательность сигналов, сводится к задаче *распознавания знака*, т. е. выявлению, какой именно знак находится в данном месте сообщения. А такая задача, как мы уже убедились в предыдущем разделе, может быть решена серией парных выборов. При этом количество информации, содержащееся в знаке, служит мерой затрат по его выявлению.

Последующие (*второе* и далее) приближения при оценке значения информации, приходящейся на знак алфавита, строятся с учетом *корреляций*, т. е. связей между буквами в словах. Дело в том, что в словах естественных языков буквы появляются не в любых сочетаниях; это понижает неопределенность угадывания следующей буквы после одной или нескольких предыдущих, например в русском языке нет слов, в которых встречается сочетание щц или фъ. И напротив, после некоторых сочетаний можно с большей определенностью, чем чистый случай, предполагать появление следующей буквы, например после распространенного сочетания пр всегда следует гласная буква, а их в русском языке 10 и, следовательно, вероятность угадывания буквы оказывается равной $1/10$, а не $1/33$. В связи с этим примем следующее определение:

> *Сообщения (а также источники, их порождающие), в которых существуют статистические связи (корреляции) между знаками или их сочетаниями, называются* ***сообщениями*** *(источниками)* ***с памятью****.*

Согласно данным из книги Л. Бриллюэна [8, с. 46], учет в английских словах двухбуквенных сочетаний понижает среднюю информацию на знак до значения $I_2^{(e)} = 3{,}32$ бит, учет трехбуквенных — до $I_3^{(e)} = 3{,}10$ бит. Шеннон сумел приблизительно оценить $I_5^{(e)} \approx 2{,}1$ бит и $I_8^{(e)} \approx 1{,}9$ бит. Аналогичные исследования для русского языка дают: $I_2^{(r)} = 3{,}52$ бит; $I_3^{(r)} = 3{,}01$ бит.

Последовательность $I_0$, $I_1$, $I_2$,... является убывающей в любом языке. Экстраполируя ее на учет бесконечного числа корреляций, можно оценить *предельную* информацию на знак в данном языке $I_\infty$, которая будет отражать *минимальную неопределенность, связанную с выбором знака алфавита без учета семантических особенностей языка*, в то время как $I_0$ явля-

ется другим предельным случаем, поскольку характеризует *наибольшую информацию*, которая может быть связана со знаком данного алфавита. Шеннон ввел характеристику, которую назвал *относительной избыточностью языка*:

$$R = 1 - \frac{I_\infty}{I_0}. \tag{2.19}$$

Избыточность является мерой бесполезно совершаемых альтернативных выборов при передаче сообщений, обусловленных лексическими особенностями языка. Эта величина показывает, какую долю лишней информации содержат тексты данного языка; лишней в том отношении, что она определяется структурой самого языка и, следовательно, может быть восстановлена без явного указания в буквенном виде.

Исследования Шеннона для английского языка дали значение $I_\infty \approx 1{,}4...1{,}5$ бит, что по отношению к $I_0 = 4{,}755$ бит создает избыточность около 0,68. Подобные оценки показывают, что и для других европейских языков, в том числе русского, избыточность составляет 60...70 %. Это означает, что в принципе возможно почти трехкратное (!) сокращение текстов без ущерба для их содержательной стороны и выразительности. Однако такое «экономичное» представление слов снижает разборчивость языка, уменьшает возможность понимания речи при наличии шума (а это одна из проблем передачи информации по реальным линиям связи), а также исключает возможность локализации и исправления ошибки (написания или передачи) при ее возникновении. Именно избыточность языка позволяет легко восстановить текст, даже если он содержит большое число ошибок или неполон (например, при отгадывании кроссвордов или при игре в «Поле чудес»). В этом смысле избыточность есть определенная страховка и гарантия разборчивости (обеспечения надежности передачи и хранения информации).

Вместе с тем именно избыточность языка и существующие в нем корреляционные связи используется в криптоанализе (т. е. «взламывании» шифрованных сообщений) — эти вопросы будут затронуты в гл. 7.

На практике учет корреляций в сочетаниях знаков сообщения весьма затруднителен, поскольку требует объемных статистических исследований текстов. Кроме того, корреляционные вероятности зависят от характера текстов и целого ряда иных их особенностей. По этим причинам в дальнейшем мы ограничим себя рассмотрением только шенноновских сообщений, т. е. будем

учитывать различную (априорную) вероятность появления знаков в тексте, но не их корреляции.

## Контрольные вопросы и задания к гл. 2

1. Почему в определении энтропии как меры неопределенности выбрана логарифмическая зависимость между $H$ и $n$? Почему выбран $\log_2$?

2. Какова энтропия следующих опытов:

а) бросок монеты;

б) бросок игральной кости;

в) вытаскивание наугад одной игральной карты из 36;

г) бросок двух игральных костей.

3. Алфавит русского языка содержит 34 буквы (с пробелом), английского — 27. Если считать вероятность появление всех букв в тексте одинаковой, то как соотносятся энтропии, связанные с угадыванием случайно выбранной буквы текста?

4. Опыт имеет два исхода. Докажите, что энтропия такого опыта максимальна, если вероятности исходов будут обе равны 0,5.

5. По условиям задачи 12 приложения А определите, с результатом выстрела которого из стрелков — А или В — связана большая неопределенность.

6. Докажите, что для двух опытов $\alpha$ и $\beta$ справедливо соотношение

$$H(\alpha) + H_\alpha(\beta) = H(\beta) + H_\beta(\alpha).$$

7. Решите задачу, рассмотренную в примере 2.2, при условии, что шары возвращаются в ящик после извлечения.

8. Опыты $\alpha$ и $\beta$ состоят в последовательном извлечении без возврата двух шаров из ящика, в котором изначально находились $n$ белых шаров и $m$ черных. Найдите $H(\alpha)$, $H(\beta)$, $H_\alpha(\beta)$ и $H_\beta(\alpha)$.

9. Какое количество информации связано с исходом следующих опытов:

а) бросок игральной кости;

б) бросок двух монет;

в) вытаскивание наугад одной игральной карты из 36;

г) бросок двух игральных костей.

10. Мы отгадываем задуманное кем-то трехзначное число. Какое количество информации требуется для отгадывания всего числа?

11. Какова оптимальная последовательность вопросов при отгадывании? Каково их минимальное число?

12. Как соотносятся количества бинарных вопросов, обеспечивающих отгадывание числа целиком и по цифрам (сначала 1-ю цифру числа, затем — 2-ю, затем — 3-ю)? Какой способ оказывается оптимальным?

13. Одинакова ли информация, необходимая для отгадывания 1-й, 2-й и 3-й цифр?

14. Как соотносятся количества информации при отгадывании числа целиком или по цифам? Почему?

15. Докажите, что $I(\alpha, \beta) = I(\beta, \alpha)$.

16. Решите задачу, описанную в примере 2.8, при условии, что помимо деревень A и B имеется деревня C, жители которой дают по очереди то правдивые, то ложные ответы, причем неизвестно, с какого они начинают.

17. По условиям задачи 12 приложения A определите, с результатом выстрела которого из стрелков — A или B — связана большая информация.

18. Вопрос имеет два варианта ответа. Возможно ли, чтобы с каждым из вариантов была связано различное количество информации? Как при этом найти полную информацию, связанную с ответом?

19. Возможно ли, чтобы ответ на бинарный вопрос содержал меньше 1 бит информации? Не содержал информации?

20. Какое количество информации содержит каждый из ответов на вопрос, если их 3 и все они равновероятны? Какое количество информации связано со всем ответом? А если равновероятных ответов $n$?

21. Источник порождает множество шестизнаковых сообщений, каждое из которых содержит 1 знак «*», 2 знака «%» и 3 знака «!». Какое количество информации содержится в каждом (одном) из таких сообщений?

22. С какой буквой русского алфавита «а» или «б» связано больше информации? Найдите эту информацию.

23. Средняя длина слова в русском языке 5,3 буквы, в английском — 4,5. Найдите вероятности появления в соответствующих текстах пробелов. Какое количество информации связано с пробелом в обоих языках?

24. По данным таблицы 2.1 с помощью Excel вычислите $I_1$ для русского алфавита и проверьте приведенное в тексте значение.

25. Дайте объяснение тому, что количество информации на знак алфавита выражается нецелым числом.

26. Что такое «*шенноновские сообщения*»? Почему теория информации имеет дело именно с такими сообщениями?

27. Как соотносятся $p_{ik}^{(\mathrm{fact})}$ и $p_{ik}^{(\mathrm{rand})}$ в сообщениях с памятью? Почему?

28. Почему мы используем «*избыточный*» язык?

29. Одинакова ли на Ваш взгляд избыточность литературных и деловых текстов? Почему?

# 3 Кодирование символьной информации

Исторически теория кодирования развивалась как самостоятельное научное направление. В ней решались две основные задачи:

1) разработка принципов *наиболее экономичного* (оптимального) первичного кодирования информации;

2) разработка кодов, обеспечивающих *надежность* передачи информации по *реальным* каналам связи.

Вообще говоря, разработка кодов может осуществляться вне связи с другими теориями. Однако в рамках теории кодирования нельзя ответить на вопрос: чем определяется длина оптимального кода? Ответ на него дает теория информации — в этом усматривается связь обеих теорий. Именно теория информации устанавливает границы возможностей методов кодирования.

В системах связи, используемых на практике, кодирование происходит в два этапа. Сначала устройство, называемое *первичным кодером*, осуществляет такой вариант кодирования, которое не предполагает возможности потерь информации при ее передаче (или хранении) — данный код также называется *первичным*. Поскольку можно предложить много способов кодирования одного и того же исходного сообщения, критерием оптимальности первичного кода оказывается длина закодированного сообщения (или отношение длины закодированного сообщения к длине исходного).

На следующем этапе, который предшествует непосредственно передаче сообщения по каналу связи, *вторичный кодер* дополняет первичный код дополнительными *проверочными знаками*, которые позволяют приемному устройству обнаружить или даже исправить возможную ошибку передачи. Критерием оптимальности вторичного кодирования, как мы увидим далее, является избыточность кода (т. е. соотношение длин его информационной и проверочной частей), кратность обнаруживаемых (или

исправляемых) ошибок, а также вероятность безошибочной передачи.

Помехоустойчивое кодирование связано с процессом передачи информации — оно будет рассмотрено в главе 6. Мы же начнем с обсуждения вопросов первичного кодирования.

## 3.1. Постановка задачи первичного кодирования. Первая теорема Шеннона

Как отмечалось при рассмотрении исходных понятий информатики, для представления дискретной информации используется некоторый алфавит. Однако однозначное соответствие между информацией и алфавитом отсутствует. Другими словами, одна и та же информация может быть представлена посредством различных дискретных сообщений и, следовательно, различных алфавитов. В связи с такой возможностью возникает проблема перехода от одного алфавита к другому, причем подобное преобразование не должно приводить к потере информации.

Введем ряд определений.

Будем считать, что источник представляет информацию в форме дискретного сообщения, используя для этого алфавит, который в дальнейшем условимся называть *первичным*. Далее это сообщение попадает в устройство, преобразующее и представляющее его в другом алфавите, — этот алфавит назовем *вторичным*.

> ***Код*** *— (1) правило, описывающее соответствие знаков или их сочетаний первичного алфавита знакам или их сочетаниям вторичного алфавита;*
> *(2) совокупности знаков вторичного алфавита, используемые для представления знаков или их сочетаний первичного алфавита.*

> ***Кодирование*** *— перевод информации, представленной посредством первичного алфавита, в последовательность кодов.*

> ***Декодирование*** *— операция, обратная кодированию, т. е. восстановление информации в первичном алфавите по имеющейся последовательности кодов.*

> *Операции кодирования и декодирования называются* ***обратимыми****, если их последовательное применение обеспечивает возврат к исходной информации без каких-либо ее потерь.*

***Кодер** — устройство, обеспечивающее выполнение операции кодирования.*

***Декодер** — устройство, производящее декодирование.*

Примером обратимого кодирования является представление знаков в телеграфном коде и их восстановление после передачи. Примером кодирования необратимого может служить перевод с одного естественного языка на другой — обратный перевод, вообще говоря, не восстанавливает исходного текста. Безусловно, для практических задач, связанных со знаковым представлением информации, возможность восстановления информации по ее коду является необходимым условием применимости кода, поэтому в дальнейшем изложении ограничим себя рассмотрением только обратимого кодирования.

Рассмотрим схему передачи информации от источника к приемнику (рис. 3.1).

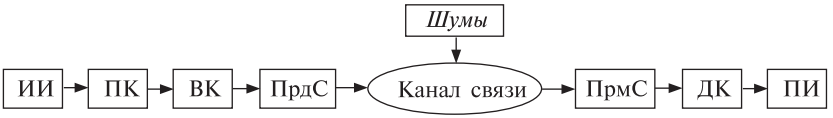



**Рис. 3.1.** Схема передачи информации

Без технической детализации можно выделить следующие модули:

ИИ — источник информации;
ПК — первичный кодер;
ВК — вторичный кодер;
ПрдС — передатчик сообщения в канал связи;
ПрмС — приемник сообщения из канала связи;
ДК — декодер сообщения;
ПИ — приемник информации.

Взаимодействуют модули следующим образом. От источника информация, представленная в первичном алфавите, поступает в первичный кодер — на его выходе получается последовательность кодов; эта последовательность поступает во вторичный кодер, действие которого состоит в построении помехоустойчивых кодов, которые передатчиком направляются в канал связи, подверженный воздействию шумов (помех), которые могут исказить передаваемые коды. После приема сообщения из канала связи она направляется в декодер, который выполняет две функции: проверку правильности передачи и перевод помехоустойчивых кодов в обычные. Для получения информации в первичном

алфавите (алфавите источника) приемник информации должен осуществить декодирование.

Таким образом, задача кодирования решается дважды — сначала строится первичный код, затем на его основе помехоустойчивый. В данном разделе, как было сказано, обсуждаются вопросы первичного кодирования.

Кодирование предшествует передаче и хранению информации. При этом, как указывалось ранее, хранение связано с фиксацией некоторого состояния носителя информации, а передача — с изменением состояния с течением времени (т. е. процессом). Эти состояния или сигналы будем называть *элементарными сигналами* — именно их совокупность и составляет алфавит кодов.

Не обсуждая технических сторон передачи и хранения сообщения (т. е. того, каким образом фактически реализованы передача-прием последовательности сигналов или фиксация состояний), попробуем дать математическую постановку задачи кодирования, привлекая представления теории информации.

Пусть первичный алфавит $A$ состоит из $N$ знаков со средней информацией на знак $I^{(A)}$, а вторичный алфавит $B$ — из $M$ знаков со средней информацией на знак $I^{(B)}$. Пусть также исходное сообщение, представленное в первичном алфавите, содержит $n$ знаков, а закодированное сообщение — $m$ знаков. Если количество информации в исходном сообщении $I_{\text{st}}(A)$, а в закодированном — $I_{\text{cod}}(B)$, то *условие обратимости кодирования*, т. е. *неисчезновения* информации при кодировании, очевидно, может быть записано следующим образом:

$$I_{\text{st}}(A) \leqslant I_{\text{cod}}(B),$$

смысл которого в том, что *операция обратимого кодирования может увеличить количество информации в сообщении, но не может его уменьшить*. Однако каждая из величин в данном неравенстве может быть заменена произведением числа знаков на среднее информационное содержание знака, т. е.

$$nI^{(A)} \leqslant mI^{(B)}, \quad \text{или} \quad I^{(A)} \leqslant \frac{m}{n} I^{(B)}.$$

Отношение $m/n$, очевидно, характеризует *среднее число знаков вторичного алфавита, которое приходится использовать для кодирования одного знака первичного алфавита* — будем называть его *средней длиной кода* или *длиной ко-*

*довой комбинации* и обозначим $K(A,B)$. Следовательно,

$$K(A,B) \geqslant \frac{I^{(A)}}{I^{(B)}}. \tag{3.1}$$

Поскольку обычно $n > m$ и, следовательно, $I^{(A)} > I^{(B)}$, оказывается $K(A,B) > 1$, т. е. один знак перчиного алфавита представляется несколькими знаками вторичного.

Способов построения кодов при фиксированных алфавитах $A$ и $B$ существует множество; в связи с этим возникает проблема выбора (или построения) наилучшего варианта — будем называть его *оптимальным кодом*. Выгодность кода при передаче и хранении информации — это экономический фактор, поскольку более эффективный код позволяет затратить на передачу сообщения меньше энергии, а также времени и, соответственно, сделать передачу дешевле; при хранении используется меньше площади поверхности (объема) носителя. При этом следует сознавать, что выгодность кода по длине не идентична временно́й выгодности всей цепочки кодирование — передача — декодирование; возможна ситуация, когда за использование эффективного кода при передаче придется расплачиваться тем, что операции кодирования и декодирования будут занимать больше времени и иных ресурсов (например, места в памяти технического устройства, если эти операции производятся с его помощью).

Каково минимально возможное значение $K(A,B)$? Как следует из (3.1), оно достигается при следующих условиях: во-первых, реализуется равенство правой и левой частей, во-вторых, числитель дроби, т. е. $I^{(A)}$, должен принимать наименьшее значение, а $I^{(B)}$ в знаменателе — наоборот — наибольшее. Таким образом, условие минимальности длины кодовой комбинации может быть записано в следующем виде:

$$K^{\min}(A,B) = \frac{\min(I^{(A)})}{\max(I^{(B)})}. \tag{3.2}$$

Данное выражение следует воспринимать как соотношение оценочного характера, устанавливающее нижний предел длины кода, однако из него неясно, в какой степени в реальных схемах кодирования возможно приближение $K(A,B)$ к $K^{\min}(A,B)$. По этой причине для теории кодирования и теории связи важнейшее значение имеют две теоремы, доказанные К. Шенноном. Первая — ее мы сейчас рассмотрим — затрагивает ситуацию с кодированием при отсутствии помех, искажающих сообщение. Вто-

рая теорема относится к реальным линиям связи с помехами и будет обсуждаться в гл. 6.

Первая теорема Шеннона, которая называется *основной теоремой о кодировании при отсутствии помех*, формулируется следующим образом:

**При отсутствии помех всегда возможен такой вариант кодирования сообщения, при котором среднее число знаков кода, приходящихся на один знак первичного алфавита, будет сколь угодно близко к отношению средних информаций на знак первичного и вторичного алфавитов.**

Приведенное утверждение является теоремой и, следовательно, должно доказываться. Мы опустим его, адресовав интересующихся именно доказательной стороной к книге А.М. Яглома и И.М. Яглома [50, с. 212–217]. Для нас важно, что теорема открывает принципиальную возможность оптимального кодирования, т. е. построения кода со средней длиной сколь угодно приближенной к $K^{\min}(A, B)$. Однако необходимо сознавать, что из самой теоремы никоим образом не следует, как такое кодирование осуществить практически, — для этого должны привлекаться какие-то дополнительные соображения, что и станет предметом нашего последующего обсуждения.

Из (3.2) видно, что имеются два пути сокращения $K^{\min}(A, B)$:

1) уменьшение числителя (т. е. сокращение количества информации, подлежащей кодированию); это возможно, если учесть различную частоту появления разных знаков в исходном сообщении, корреляции двухбуквенные, трехбуквенные и т. п. (как показано в п. 2.3, при этом $I_0 > I_1 > I_2 > \ldots > I_\infty$);

2) увеличение знаменателя — для этого необходимо применить такой способ кодирования, при котором появление знаков вторичного алфавита было бы близко к равновероятному, т. е. к $I^{(B)} = \log_2 M$.

В частной ситуации, рассмотренной подробно К. Шенноном, при кодировании сообщения в первичном алфавите учитывается различная вероятность появления знаков (то, что в п. 2.3 мы назвали «*первым приближением*»), однако их корреляции не отслеживаются — источники подобных сообщений называются *источниками без памяти* или *шенноновскими*. Если при этом обеспечена равная вероятность появления знаков вторичного алфавита, то, как следует из (3.2), для минимальной средней длины

кода оказывается справедливым соотношение

$$K^{\min}(A,B) = \frac{I_1^{(A)}}{\log_2 M}. \tag{3.3}$$

В качестве показателя превышения $K(A,B)$ над $K^{\min}(A,B)$ можно ввести *относительную избыточность кода*

$$Q(A,B) = \frac{K(A,B) - K^{\min}(A,B)}{K^{\min}(A,B)} = \frac{K(A,B)}{K^{\min}(A,B)} - 1. \tag{3.4}$$

Данная величина отражает, в какой мере операция кодирования увеличила длину исходного сообщения. Очевидно, $Q(A,B) \to 0$ при $K(A,B) \to K^{\min}(A,B)$. Следовательно, решение проблемы оптимизации кода состоит в нахождении таких схем (алгоритмов) кодирования, которые обеспечили бы приближение средней длины кода к значению $K^{\min}(A,B)$. Легко показать, что чем меньше $Q(A,B)$, тем $I_{\text{cod}}(B)$ ближе к $I_{\text{st}}(A))$, т. е. возникает меньше информации, связанной с кодированием, более выгодным оказывается код и более эффективной операция кодирования. Оптимальным при прочих равных условиях (алфавитах $A$ и $B$) будет код с наименьшей избыточностью.

Используя понятие избыточности кода, можно дать более короткую формулировку первой теоремы Шеннона:

**При отсутствии помех передачи всегда возможен такой вариант кодирования сообщения, при котором избыточность кода будет сколь угодно близкой к нулю.**

Наиболее важной для практики оказывается ситуация, когда $M = 2$, т. е. для представления кодов в линии связи используется лишь два типа сигналов (их называют *элементарными сигналами*) — технически это наиболее просто реализуемый вариант (например, существование напряжения в проводе (будем называть это *импульсом*) или его отсутствие (*пауза*); наличие или отсутствие отверстия на перфокарте или намагниченной области на диске); подобное кодирование называется *двоичным*. Знаки двоичного алфавита принято обозначать «0» и «1», но нужно воспринимать их как буквы, а не цифры. Удобство двоичных кодов и в том, что при равных длительностях и вероятностях каждый элементарный сигнал (0 или 1) несет в себе 1 бит информации ($\log_2 M = 1$); тогда из (3.3)

$$K^{\min}(A,2) = I_1^{(A)},$$

и первая теорема Шеннона получает следующую интерпретацию:

**При отсутствии помех средняя длина двоичного кода может быть сколь угодно близкой к средней информации, приходящейся на знак первичного алфавита.**

Применение формулы (3.4) для двоичных сообщений источника без памяти при кодировании знаками равной вероятности дает

$$Q(A,2) = \frac{K(A,2)}{I_1^{(A)}} - 1. \qquad (3.5)$$

При декодировании двоичных сообщений возникает проблема выделения из потока сигналов (последовательности импульсов и пауз) кодовых слов (групп элементарных сигналов), соответствующих отдельным знакам первичного алфавита. При этом приемное устройство фиксирует *интенсивность* и *длительность* сигналов, а также может соотносить некоторую последовательность сигналов с эталонной (*таблицей кодов*).

Возможны следующие особенности вторичного алфавита, используемого при кодировании:

- элементарные сигналы (0 и 1) могут иметь одинаковые длительности ($\tau_0 = \tau_1$) или разные ($\tau_0 \neq \tau_1$);
- длина кода может быть одинаковой для всех знаков первичного алфавита (в этом случае код называется *равномерным*) или же коды разных знаков первичного алфавита могут иметь различную длину (*неравномерный* код);
- коды могут строиться для отдельного знака первичного алфавита (*алфавитное кодирование*) или для их комбинаций (*кодирование блоков, слов*).

Сочетания перечисленных особенностей определяют основу конкретного способа кодирования, однако даже при одинаковой основе возможны различные варианты построения кодов, отличающихся своей эффективностью. Нашей ближайшей задачей будет рассмотрение различных схем кодирования для некоторых основ.

## 3.2. Способы построения двоичных кодов

### 3.2.1. Равномерное алфавитное двоичное кодирование. Байтовый код

Пожалуй, самым простым способом кодирования является алфавитное равномерное, в котором всем знакам первичного алфавита ставятся в соответствие коды, содержащие одинаковое число двоичных разрядов (элементарных сигналов). Длина кода

зависит от числа знаков первичного алфавита ($N$). Как известно, с помощью $k$ двоичных разрядов можно построить $2^k$ различных комбинаций. Поскольку $k$ — целое, при кодировании общее число комбинаций должно быть не меньше количества знаков первичного алфавита, т. е. выполняться соотношение $2^k \geqslant N$ или $k \geqslant \log_2 N$. Кодирование состоит в построении таблицы кодов, в которой каждому знаку первичного алфавита указывается своя комбинация двоичных разрядов. Поскольку длины кодов одинаковы, размещение знаков в кодовой таблице может быть осуществлено по-разному (не однозначно — это вопрос соглашения); для обеспечения возможности прочтения информации на различных устройствах их кодовые таблицы, безусловно, должны быть согласованы — для этого вводятся *стандарты кодов*.

В кодер подается последовательность первичных знаков; для каждого из них кодер в соответствии с кодовой таблицей определяет и выдает код. В результате получается сообщение, состоящее из непрерывной последовательности нулей и единиц. Приемное устройство просто отсчитывает из этого сообщения оговоренное заранее количество элементарных сигналов и интерпретирует цепочку (устанавливает, какому знаку она соответствует), соотнося ее с таблицей кодов.

Примером равномерного алфавитного кодирования является телеграфный *код Бодо*, пришедший на смену азбуке Морзе. Исходный алфавит должен содержать не более 32 знаков; тогда $k = \log_2 32 = 5$, т. е. каждый знак первичного алфавита содержит 5 бит информации и кодируется цепочкой из 5 двоичных знаков. Условие $N \leqslant 32$, очевидно, выполняется для языков, основанных на латинском алфавите ($N = 27 = 26 +$ «пробел»), однако в русском алфавите 34 буквы (с пробелом) — именно по этой причине пришлось «сжать» алфавит и объединить в один знак «е» и «ё», а также «ь» и «ъ». После такого сжатия $N = 32$, однако, не остается свободных кодов для знаков препинания, поэтому в телеграммах они отсутствуют или заменяются буквенными аббревиатурами; это не является заметным ограничением, поскольку, как указывалось выше, избыточность языка позволяет легко восстановить информационное содержание сообщения. Избыточность кода Бодо для русского языка $Q^{(\text{Бд})}(r, 2) = 0{,}148$, для английского $Q^{(\text{Бд})}(e, 2) = 0{,}239$.

Другим важным для нас примером использования равномерного алфавитного кодирования является представление символьной (знаковой) информации в компьютере. Чтобы определить

длину кода, необходимо начать с установления количество знаков в первичном алфавите. Компьютерный алфавит должен включать:

- $26{\times}2 = 52$ букв латинского алфавита (с учетом прописных и строчных);
- $33{\times}2 = 66$ букв русского алфавита;
- цифры $0,\ldots, 9$ — всего 10;
- знаки математических операций, знаки препинания, спецсимволы $\approx 20$.

Получаем, что общее число символов $N \approx 148$. Теперь можно оценить длину кодовой цепочки: $k \geqslant \log_2 148 \geqslant 7{,}21$. Поскольку длина кода выражается целым числом, очевидно, $k = 8$. Именно такой способ кодирования принят в компьютерных системах: любому знаку ставится в соответствие код из 8 двоичных разрядов (8 бит). Эта последовательность сохраняется и обрабатывается как единое целое (т. е. отсутствует доступ к отдельному биту) — по этой причине разрядность устройств компьютера, предназначенных для хранения или обработки информации, кратна 8. Совокупность восьми связанных бит получила название *байт*, а представление таким образом знаков — *байтовым кодированием*.

Байт наряду с битом может использоваться как единица измерения количества информации в сообщении. *Один байт соответствует количеству информации в одном знаке алфавита при их равновероятном распределении.* Этот способ измерения количества информации называется также *объемным*. Пусть имеется некоторое сообщение (последовательность знаков); оценка количества содержащейся в нем информации согласно рассмотренному ранее вероятностному подходу (с помощью формулы Шеннона (2.14)) дает $I_{\text{вер}}$, а объемная мера пусть равна $I_{\text{об}}$; соотношение между этими величинами вытекает из (2.7):

$$I_{\text{вер}} \leqslant I_{\text{об}}.$$

Именно байт принят в качестве единицы измерения количества информации в международной системе единиц СИ. 1 байт = = 8 бит. Наряду с байтом для измерения количества информации используются более крупные производные единицы:

1 Кбайт = $2^{10}$ байт = 1024 байт (килобайт);
1 Мбайт = $2^{20}$ байт = 1024 Кбайт (мегабайт);
1 Гбайт = $2^{30}$ байт = 1024 Мбайт (гигабайт);
1 Тбайт = $2^{40}$ байт = 1024 Гбайт (терабайт).

Использование 8-битных цепочек позволяет закодировать $2^8 = 256$ символов, что превышает оцененное выше $N$ и, следовательно, дает возможность употребить оставшуюся часть кодовой таблицы для представления дополнительных символов.

Однако недостаточно только условиться об определенной длине кода. Ясно, что способов кодирования, т. е. вариантов сопоставления знакам первичного алфавита восьмибитных цепочек, очень много. По этой причине для совместимости технических устройств и обеспечения возможности обмена информацией между многими потребителями требуется согласование кодов. Подобное согласование осуществляется в форме *стандартизации кодовых таблиц*. Первым таким международным стандартом, который применялся на больших вычислительных машинах, был *EBCDIC* (*Extended Binary Coded Decimal Interchange Code*) — «*расширенная двоичная кодировка десятичного кода обмена*». В персональных компьютерах и телекоммуникационных системах применяется международный байтовый код *ASCII* (*American Standard Code for Information Interchange* — «*американский стандартный код обмена информацией*»). Он регламентирует коды первой половины кодовой таблицы (номера кодов от 0 до 127, т. е. первый бит всех кодов 0). В эту часть попадают коды прописных и строчных английских букв, цифры, знаки препинания и математических операций, а также некоторые управляющие коды (номера от 0 до 31), вырабатываемые при использовании клавиатуры. В табл. 3.1 приведены некоторые ASCII-коды.

Таблица 3.1

| Знак, клавиша | ASC-коды | | Буква русского алфавита | ASC-коды | |
|---|---|---|---|---|---|
| | Двоичный | Десятичный | | Двоичный | Десятичный |
| Пробел | 00100000 | 32 | А | 10000000 | 128 |
| A (лат) | 01000001 | 65 | Б | 10000001 | 129 |
| B (лат) | 01000010 | 66 | Ю | 10011110 | 158 |
| Z | 01011010 | 90 | Я | 10011111 | 159 |
| 0 | 00110000 | 48 | а | 10100000 | 160 |
| 1 | 00110001 | 49 | п | 10101111 | 175 |
| 9 | 00111001 | 57 | р | 11100000 | 224 |
| [Esc] | 00011011 | 27 | с | 11100001 | 225 |
| [Enter] | 00001101 | 13 | я | 11101111 | 239 |

Вторая часть кодовой таблицы — она считается расширением основной — охватывает коды в интервале от 128 до 255 (первый бит всех кодов 1). Она используется для представления символов национальных алфавитов (например, русского), а также символов псевдографики. Для этой части также имеются стандарты, например для символов русского языка это КОИ-8, КОИ-7 и др.

Как в основной таблице, так и в ее расширении коды букв и цифр соответствуют их лексикографическому порядку (т. е. порядку следования в алфавите) — это обеспечивает возможность автоматизации обработки текстов и ускоряет ее.

В настоящее время появился и находит все более широкое применение еще один международный стандарт кодировки — *Unicode*. Его особенность в том, что в нем использовано 16-битное кодирование, т. е. для представления каждого символа отводится 2 байта. Такая длина кода обеспечивает включения в первичный алфавит 65536 знаков. Это, в свою очередь, позволяет создать и использовать единую для всех распространенных алфавитов кодовую таблицу.

Недостаток равномерных кодов в их большой избыточности. Даже приведенные выше оценки для кода Бодо, все 32 комбинации которого используются, дают величины 15...25 %. Для байтового кода избыточность оказывается еще значительней. За удобство использования кодов приходится расплачиваться их большой избыточностью. В то же время согласно первой теореме Шеннона возможно построение кодов, избыточность которых может быть сколь угодно близкой к нулю. К рассмотрению способов построения таких кодов мы переходим.

### 3.2.2. Алфавитное неравномерное двоичное кодирование сигналами равной длительности. Коды с разделителем

Как следует из названия, в способах кодировании, относящихся к этой группе, знаки первичного алфавита (например, русского) кодируются комбинациями символов двоичного алфавита (т. е. 0 и 1), причем длина кодовых комбинаций и, соответственно, длительность передачи отдельного кода, могут различаться. Длительности элементарных сигналов при этом одинаковы ($\tau_0 = \tau_1 = \tau$). Очевидно, для передачи информации, в среднем приходящейся на знак первичного алфавита, необходимо время $K(A, 2)\tau$. Таким образом, задачу оптимизации неравномерного кодирования можно сформулировать следующим образом: *построить такую схему кодирования, в которой суммарная*

*длительность кодов при передаче (или суммарное число элементарных сигналов при хранении) данного сообщения была бы наименьшей.* За счет чего возможна такая оптимизация? Очевидно, суммарная длительность сообщения будет меньше, если применить следующий подход: тем знакам первичного алфавита, которые встречаются в сообщении чаще, присвоить меньшие по длине коды, а тем, относительная частота которых меньше, — коды более длинные. Другими словами, коды знаков первичного алфавита, вероятность появления которых в сообщении выше, следует строить из возможно меньшего числа элементарных сигналов, а длинные коды использовать для знаков с малыми вероятностями.

Параллельно должна решаться проблема *различимости кодов*. Представим, что на выходе кодера получена следующая последовательность элементарных сигналов:

00100010000111010101110000110

Каким образом она может быть декодирована? Если бы код был равномерным, приемное устройство просто отсчитывало бы заданное (фиксированное) число элементарных сигналов (например, 5, как в коде Бодо) и интерпретировало их в соответствии с кодовой таблицей. При использовании неравномерного кодирования возможны два подхода к обеспечению различимости кодов.

Первый состоит в использовании специальной комбинации элементарных сигналов, которая интерпретируется декодером как *разделитель знаков*. Второй – в применении *префиксных кодов*. В данном разделе речь пойдет о построении неравномерного кода с разделителями знаков.

Условимся, что разделителем отдельных кодов букв будет последовательность 00 (признак конца знака), а разделителем слов-слов — 000 (признак конца слова — пробел). Довольно очевидными оказываются следующие правила построения кодов:

- код признака конца знака может быть включен в код буквы, поскольку не существует отдельно (т. е. коды всех букв будут заканчиваться 00);
- коды букв не должны содержать двух и более нулей подряд в середине (иначе они будут восприниматься как конец знака);
- код буквы (кроме пробела) всегда должен начинаться с 1;
- разделителю слов (000) всегда предшествует признак конца знака; при этом реализуется последовательность 00000 (т. е., если в конце кода встречается комбинация . . . 000 или . . . 0000,

Таблица 3.2

| Буква | $p_i$ | Код | $k_i$ | Буква | $p_i$ | Код | $k_i$ |
|---|---|---|---|---|---|---|---|
| пробел | 0,174 | 000 | 3 | я | 0,018 | 1011000 | 7 |
| о | 0,090 | 100 | 3 | ы | 0,016 | 1011100 | 7 |
| е | 0,072 | 1000 | 4 | з | 0,016 | 1101000 | 7 |
| а | 0,062 | 1100 | 4 | ь, ъ | 0,014 | 1101100 | 7 |
| и | 0,062 | 10000 | 5 | б | 0,014 | 1110000 | 7 |
| т | 0,053 | 10100 | 5 | г | 0,013 | 1110100 | 7 |
| н | 0,053 | 11000 | 5 | ч | 0,012 | 1111000 | 7 |
| с | 0,045 | 11100 | 5 | й | 0,010 | 1111100 | 7 |
| р | 0,040 | 101000 | 6 | х | 0,009 | 10101000 | 8 |
| в | 0,038 | 101100 | 6 | ж | 0,007 | 10101100 | 8 |
| л | 0,035 | 110000 | 6 | ю | 0,006 | 10110000 | 8 |
| к | 0,028 | 110100 | 6 | ш | 0,006 | 10110100 | 8 |
| м | 0,026 | 111000 | 6 | ц | 0,004 | 10111000 | 8 |
| д | 0,025 | 111100 | 6 | щ | 0,003 | 10111100 | 8 |
| п | 0,023 | 1010000 | 7 | э | 0,003 | 11010000 | 8 |
| у | 0,021 | 1010100 | 7 | ф | 0,002 | 11010100 | 8 |

они не воспринимаются как разделитель слов); следовательно, коды букв могут оканчиваться на 0 или 00 (до признака конца знака).

В соответствии с перечисленными правилами построим кодовую таблицу (табл. 3.2) для букв русского алфавита, основываясь на приведенных ранее (табл. 2.1) вероятностях появления отдельных букв.

Теперь по формуле (А.11) можно найти среднюю длину кода $K(r,2)$ для данного способа кодирования:

$$K(r,2) = \sum_{i=1}^{32} p_i k_i = 4{,}964.$$

Поскольку для русского языка, как указывалось в п. 2.3, $I_1^{(r)} = 4{,}356$ бит, избыточность данного кода согласно (3.5)

$$Q(r,2) = 4{,}964/4{,}356 - 1 \approx 0{,}14;$$

это означает, что при данном способе кодирования будет передаваться приблизительно на 14 % больше информации, чем содержит исходное сообщение. Аналогичные вычисления для английского языка дают значение $K(e,2) = 4{,}716$, что при $I_1^{(e)} = 4{,}036$ бит приводят к избыточности кода $Q(e,2) = 0{,}168$.

Полученные оценки избыточности ниже, чем у равномерных кодов, но они по-прежнему остаются весьма значительными и малопригодными для практического использования.

### 3.2.3. Алфавитное неравномерное двоичное кодирование сигналами равной длительности. Префиксные коды

Совершенно понятно, что причиной высокой избыточности кодов, построенных по описанной выше схеме, является наличие разделителей знаков, удлиняющих любой код на 2 бита. Естественно, возникает вопрос: можно ли предложить схему неравномерного кодирования, в которой был бы сохранен принцип «больше вероятность знака в тексте — меньше его код», но не требовалось использовать разделителей знаков? Параллельно можно было бы решить вопрос о существовании оптимального способа неравномерного двоичного кодирования.

Суть первой проблемы состоит в нахождении такого варианта кодирования сообщения, при котором последующее выделение из него каждого отдельного знака (т. е. декодирование) оказывается однозначным без специальных указателей разделения знаков. Наиболее простыми и употребимыми кодами такого типа являются так называемые *префиксные коды*, которые удовлетворяют следующему условию (*условию Фано*):

**Неравномерный код может быть однозначно декодирован, если никакой из кодов не совпадает с началом (префиксом*) какого-либо иного более длинного кода.**

Например, если имеется код 110, то уже не могут использоваться коды 1, 11, 1101, 110101 и пр. Если условие Фано выполняется, то при прочтении (расшифровке) закодированного сообщения путем сопоставления со списком кодов всегда можно точно указать, где заканчивается один код и начинается другой.

**Пример 3.1.**

Пусть имеется следующая таблица префиксных кодов:

| а | л | м | р | у | ы |
|---|---|---|---|---|---|
| 10 | 010 | 00 | 11 | 0110 | 0111 |

Требуется декодировать сообщение
00100010000111010101110000110

* В языковедении термин «*префикс*» означает «*приставка*».

Декодирование осуществляется циклическим повторением следующих действий:

а) отрезать от текущего сообщения крайний левый символ, присоединить справа к рабочему кодовому слову;

б) сравнить рабочее кодовое слово с кодовой таблицей; если совпадения нет, перейти к п. а);

в) декодировать рабочее кодовое слово, очистить его;

г) проверить, имеются ли еще знаки в сообщении; если «да», перейти к п. а).

Применение данного алгоритма дает:

| Шаг | Рабочее слово | Текущее сообщение | Распознанный знак | Декодированное сообщение |
|---|---|---|---|---|
| 0 | пусто | **0**0100010000111010101110000110 | – | – |
| 1 | 0 | **0**1000100001110101011100000110 | нет | – |
| 2 | **00** | **1**0001000011101010111000000110 | м | м |
| 3 | 1 | **0**0010000111010101110000110 | нет | м |
| 4 | **10** | **0**0100001110101011100000110 | ***а*** | ма |
| 5 | 0 | **0**1000011101010111000000110 | нет | ма |
| 6 | **00** | **1**0000111010101110000110 | ***м*** | мам |
| ... | | | | |

Доведя процедуру до конца, получим сообщение: «*мама мыла раму*».

Таким образом, использование префиксного кодирования позволяет делать сообщение более коротким, поскольку нет необходимости передавать разделители знаков. Однако условие Фано не устанавливает способа формирования префиксного кода и, в частности, наилучшего из возможных. Мы обсудим две схемы построения префиксных кодов.

Для возможности дальнейшего сопоставления в качестве исходного мы возьмем один и тот же код, состоящий из 8 знаков с вероятностями:

| Знак | $p_i$ | Знак | $p_i$ |
|---|---|---|---|
| a | 0,08 | e | 0,08 |
| b | 0,44 | f | 0,08 |
| c | 0,08 | g | 0,08 |
| d | 0,08 | h | 0,08 |

Ясно, что поскольку алфавит содержит 8 знаков, длина кода при равномерном кодировании составила бы 3, а его избыточность — около 17 %.

**Префиксный код Шеннона–Фано.** Данный вариант кодирования был предложен в 1948–1949 гг. независимо Р. Фано и К. Шенноном и по этой причине назван по их именам. Построение кода Шеннона–Фано начинается с переупорядочивания исходной кодовой таблицы *в порядке убывания* вероятностей.

Далее знаки делятся на две группы таким образом, чтобы суммы вероятностей в каждой из них были бы приблизительно одинаковыми. В нашем примере в первую группу попадут $b$ и $a$ с суммой вероятностей 0,52 — им присвоим первый знак кода 0. Сумма вероятностей для остальных четырех знаков составит 0,48 — первый знак кода им приписывается 1. Продолжим деление каждой из групп на подгруппы по этой же схеме, т. е. так, чтобы суммы вероятностей на каждом шаге в соседних подгруппах были бы возможно более близкими. Деление прекращается, когда в группе остается одно значение. Знаки кода появляются постепенно на каждом послежующем шаге деления. В результате получаем табл. 3.3.

Из процедуры построения кодов легко видеть, что они удовлетворяют условию Фано и, следовательно, код является префиксным.

Средняя длина кода $K(A,2) = \sum_{i=1}^{8} p_i k_i = 2{,}8$.

Средняя информация на знак $I(A,2) = -\sum_{i=1}^{8} p_i \log_2 p_i =$ $= 2{,}562$ бит.

Избыточность составила $Q^{(\text{Ш}-\Phi)}(A2) = 2{,}8/2{,}562 - 1 \approx 0{,}09$.

Таким образом, код Шеннона–Фано обеспечивает построение более компактного (сжатого) кода по сравнению с равномерным.

Построенные по изложенной схеме коды Шеннона–Фано для букв русского алфавита представлены в табл. 3.2. Средняя длина кода $K(r,2) = 4{,}395$, а его избыточность $Q(r,2) = 0{,}009$.

**Префиксный код Хаффмана.** Способ *оптимальноro* префиксного двоичного кодирования был предложен Д. Хаффманом (D.A. Huffman) в 1952 г. Построение кодов Хаффмана мы рассмотрим на том же примере. Процедура построения кода оказывается более громоздкой, чем в коде Шеннона–Фано. Она представляет собой две операции, первая из которых называется *свертывание алфавита* (сворачивание), а второе — *развертывание кодов*.

Таблица 3.3

| Знак (A) | $p_i$ | Знаки кода | | | | Код | $k_i$ | $k_i p_i$ |
|---|---|---|---|---|---|---|---|---|
| | | 1-й | 2-й | 4-й | 5-й | | | |
| b | 0,44 | 0 | 0 | | | 00 | 2 | 0,88 |
| a | 0,08 | 0 | 1 | | | 01 | 2 | 0,16 |
| c | 0,08 | 1 | 0 | 0 | | 100 | 3 | 0,24 |
| d | 0,08 | 1 | 0 | 1 | 0 | 101 | 4 | 0,32 |
| e | 0,08 | 1 | 0 | 1 | 1 | 1011 | 4 | 0,40 |
| f | 0,08 | 1 | 1 | 0 | | 110 | 3 | 0,24 |
| g | 0,08 | 1 | 1 | 1 | 0 | 1110 | 4 | 0,32 |
| h | 0,08 | 1 | 1 | 1 | 1 | 1111 | 4 | 0,32 |

Процедура вновь начинается с упорядочения знаков алфавита по убыванию вероятностей их появления тексте. Далее два последних знака заменяются одним — промежуточным, которому приписывается вероятность, равная сумме вероятностей объединяемых знаков. Сразу же оставшиеся $N-1$ знака переупорядочиваются в порядке убывания вероятностей. Затем вновь два нижние знака заменяются одним и т. д. до тех пор, пока в алфавите не останется два знака. Таким образом, сворачивание занимает $N-2$ шага (от $\mathrm{A}^{(1)}$ до $\mathrm{A}^{(6)}$). Оставшимся знакам одному приписывают код 0, другому 1, после чего выполняется обратная операция разворачивания, в ходе которой и выявляются знаки кодов (табл. 3.4).

Построение кодов осуществляется в ходе процедуры разворачивания. Смысл ее в следующем: первый код в вероятностью 0,56 порождает коды 0,32 и 0,24 — им приписывается «материнский» код 0 и индивидуальные — верхнему 0, нижнему 1. Код знака с вероятностью 0,44 не порождает никаких иных кодов, поэтому до самого начала у него сохраняется код 1. Код 0,32, в свою очередь, порождает два кода 0,16 — они получают коды 00+0 и 00+1. Оба прохода проиллюстрированы в таблице стрелками переходов.

Средняя длина кода

$$K(A,2) = \sum_{i=1}^{8} p_i k_i = 2{,}6.$$

Средняя информация на знак, как и в предыдущей оценке, равна 2,562 бит; следовательно, избыточность составит

$$Q^{(\mathrm{Хф})}(A,2) = 2{,}6/2{,}562 - 1 \approx 0{,}015.$$

Таблица 3.4

| Знак | $k_i$ | $p_i$ | Код | Свернутые алфавиты | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | A(1) | Код | A(2) | код | A(3) | Код | A(4) | Код | A(5) | Код | A(6) | Код | A(1) | Код | A(2) | код | A(3) | Код | A(4) | Код | A(5) | Код | A(6) | Код |
| b | 1 | 0,44 | 1 | 0,44 | 1 | 0,44 | 1 | 0,44 | 1 | 0,44 | 1 | 0,44 | 1 | 0,56 | 0 | 0,44 | 1 | 0,44 | 1 | 0,44 | 1 | 0,44 | 1 | 0,56 | 0 |
| a | 4 | 0,08 | 011 | 0,16 | 000 | 0,16 | 000 | 0,16 | 000 | 0,24 | 01 | 0,32 | 00 | 0,44 | 1 | 0,16 | 000 | 0,16 | 000 | 0,16 | 001 | 0,24 | 01 | 0,44 | 1 |
| c | 4 | 0,08 | 0100 | 0,08 | 0100 | 0,16 | 001 | 0,16 | 001 | 0,16 | 000 | 0,24 | 01 | | | 0,08 | 0100 | 0,16 | 001 | 0,16 | 010 | 0,16 | 001 | | |
| d | 3 | 0,08 | 0101 | 0,08 | 0101 | 0,08 | 0100 | 0,16 | 010 | 0,16 | 001 | | | | | 0,08 | 0101 | 0,08 | 0100 | 0,08 | 011 | | | | |
| e | 4 | 0,08 | 0010 | 0,08 | 011 | 0,08 | 0101 | 0,08 | 011 | | | | | | | 0,08 | 011 | 0,08 | 0101 | | | | | | |
| g | 4 | 0,08 | 0011 | 0,08 | 0010 | 0,08 | 011 | | | | | | | | | 0,08 | 0010 | 0,08 | 011 | | | | | | |
| f | 4 | 0,08 | 0000 | 0,08 | 0011 | | | | | | | | | | | 0,08 | 0011 | | | | | | | | |
| h | 4 | 0,08 | 0001 | | | | | | | | | | | | | | | | | | | | | | |

Из процедуры построения кодов вновь видно, что они удовлетворяют условию Фано и, следовательно, не требуют разделителя.

Результат, полученный на рассмотренном примере, не случаен. Оба префиксных кода обеспечивают сжатие информации при кодировании по сравнению с кодированием равномерным. Однако код Хаффмана имеет важное теоретическое значение — строго математически можно доказать, что код Хаффмана является *самым экономичным* (оптимальным) из всех возможных для заданного алфавита и набора вероятностей, т. е. *ни для какого метода алфавитного кодирования длина кода не может оказаться меньше, чем код Хаффмана* [50, с. 209–211]. При этом не следует думать, что он представляет чисто теоретический интерес. Метод Хаффмана и его модификация — метод адаптивного кодирования (*динамическое кодирование Хаффмана*) — нашли широчайшее применение в программах-архиваторах, программах резервного копирования файлов и дисков, в системах сжатия информации в модемах и факсах.

Рассмотренный выше пример носит скорее иллюстративный характер. При кодировании алфавитов естественного языка избыточности для обоих методов оказываются одинаковыми, о чем можно судить по табл. 3.5 для букв русского алфавита.

Поскольку длины кодов в обоих методах для каждой отдельной буквы совпадают, избыточности также окажутся одинаковыми:

$$Q^{(\text{Ш}-\Phi)}(r,2) = Q^{(\text{Х})}(r,2) = 0{,}0090.$$

### 3.2.4. Алфавитное кодирование с неравной длительностью элементарных сигналов. Код Морзе

В качестве примера использования данного варианта кодирования рассмотрим телеграфный код Морзе («*азбука Морзе*»). В нем каждой букве или цифре сопоставляется некоторая последовательность кратковременных импульсов — точек и тире, разделяемых паузами. Длительности импульсов и пауз различны: если продолжительность импульса, соответствующего точке, обозначить $\tau$, то длительность импульса тире составляет $3\tau$, длительность паузы между точкой и тире $\tau$, пауза между буквами слова $3\tau$ (длинная пауза), пауза между словами (пробел) — $6\tau$. Таким образом, под знаками кода Морзе следует понимать:

Таблица 3.5

| Знак | $p_i$ | Код Ш.–Ф. | Код Хаф. | $k_i$ | Знак | $p_i$ | Код Ш.–Ф. | Код Хаф. | $k_i$ |
|---|---|---|---|---|---|---|---|---|---|
| Пробел | 0,174 | 000 | 000 | 3 | я | 0,018 | 110110 | 001101 | 6 |
| о | 0,090 | 001 | 111 | 3 | ы | 0,016 | 110111 | 010110 | 6 |
| е | 0,072 | 0100 | 0100 | 4 | з | 0,016 | 111000 | 010111 | 6 |
| а | 0,062 | 0101 | 0110 | 4 | ь,ъ | 0,014 | 111001 | 100001 | 6 |
| и | 0,062 | 0110 | 0111 | 4 | б | 0,014 | 111010 | 101100 | 6 |
| т | 0,053 | 0111 | 1001 | 4 | г | 0,013 | 111011 | 101101 | 6 |
| н | 0,053 | 1000 | 1010 | 4 | ч | 0,012 | 111100 | 110011 | 6 |
| с | 0,045 | 1010 | 1101 | 4 | й | 0,010 | 1111010 | 0011001 | 7 |
| р | 0,040 | 10011 | 00101 | 5 | х | 0,009 | 1111011 | 1000000 | 7 |
| в | 0,038 | 10010 | 00111 | 5 | ж | 0,007 | 1111100 | 1000001 | 7 |
| л | 0,035 | 10110 | 01010 | 5 | ю | 0,006 | 1111101 | 1100101 | 7 |
| к | 0,028 | 10111 | 10001 | 5 | ш | 0,006 | 11111100 | 00110000 | 8 |
| м | 0,026 | 11000 | 10111 | 5 | ц | 0,004 | 11111101 | 11001000 | 8 |
| д | 0,025 | 11010 | 11000 | 5 | щ | 0,003 | 11111110 | 11001001 | 8 |
| п | 0,023 | 110010 | 001000 | 6 | э | 0,003 | 111111110 | 001100010 | 9 |
| у | 0,021 | 110011 | 001001 | 6 | ф | 0,002 | 111111111 | 001100011 | 9 |

«·» — «*короткий импульс* + *короткая пауза*», «**–**» — «*длинный импульс* + *короткая пауза*», «0» — «*длинная пауза*», т. е. код оказывается *троичным*.

Свой код Морзе разработал в 1838 г., т. е. задолго до исследований относительной частоты появления различных букв в текстах. Однако им был правильно выбран принцип кодирования — буквы, которые встречаются чаще, должны иметь более короткие коды, чтобы сократить общее время передачи. Относительные частоты букв английского алфавита он оценил простым подсчетом литер в ячейках типографской наборной машины. Поэтому самая распространенная английская буква «$E$» получила код «*точка*». При составлении кодов Морзе для букв русского алфавита учет относительной частоты букв не производился, что, естественно, повысило его избыточность.

Как и в рассмотренных ранее вариантах кодирования, произведем оценку избыточности. По-прежнему для удобства сопоставления данные представим в формате табл. 3.1 (см. табл. 3.6). Признак конца буквы («0») в их кодах не отображается, но учтен в величине $k_i$ — длине кода буквы $i$.

Среднее значение длины кода $K(r, 3) = 3{,}361$. Полагая появление знаков вторичного алфавита равновероятным, получаем

Таблица 3.6

| Буква | Код | $p_i$ | $k_i$ | Буква | Код | $p_i$ | $k_i$ |
|---|---|---|---|---|---|---|---|
| Пробел | 00 | 0,174 | 2 | я | ·−·− | 0,018 | 5 |
| о | −−− | 0,090 | 4 | ы | −·−− | 0,016 | 5 |
| е | · | 0,072 | 2 | з | −−·· | 0,016 | 5 |
| а | ·− | 0,062 | 3 | ь, ъ | −··− | 0,014 | 5 |
| и | ·· | 0,062 | 3 | б | −··· | 0,014 | 5 |
| т | − | 0,053 | 2 | г | −−· | 0,013 | 4 |
| н | −· | 0,053 | 3 | ч | −−−· | 0,012 | 5 |
| с | ··· | 0,045 | 4 | й | ·−−− | 0,010 | 5 |
| р | ·−· | 0,040 | 4 | х | ···· | 0,009 | 5 |
| в | ·−− | 0,038 | 4 | ж | ···− | 0,007 | 5 |
| л | ·−·· | 0,035 | 5 | ю | ··−− | 0,006 | 5 |
| к | −·− | 0,028 | 4 | ш | −−−− | 0,006 | 5 |
| м | −− | 0,026 | 2 | ц | −·−· | 0,004 | 5 |
| д | −·· | 0,025 | 4 | щ | −−·− | 0,003 | 5 |
| п | ·−−· | 0,023 | 5 | э | ··−·· | 0,003 | 6 |
| у | ··− | 0,021 | 4 | ф | ··−· | 0,002 | 5 |

среднюю информацию на знак $I^{(2)} = \log_2 3 = 1{,}585$ бит. Подставляя эти данные, а также $I_1^{(r)} = 4{,}356$ бит (для русского алфавита) в (3.4), получаем

$$Q(r, 2) = (3{,}361 \cdot 1{,}585)/4{,}356 - 1 \approx 0{,}223,$$

т. е. избыточность составляет около 22 % (для английского языка приблизительно 19 %). Тем не менее, код Морзе имел в недалеком прошлом весьма широкое распространение в ситуациях, когда источником и приемником сигналов являлся человек (не техническое устройство) и на первый план выдвигалась не экономичность кода, а удобство его восприятия.

### 3.2.5. Блочное двоичное кодирование

Вернемся к проблеме оптимального кодирования. Пока что наилучший результат (наименьшая избыточность) был получен при кодировании методом Хаффмана — для русского алфавита избыточность оказалась менее 1 %. При этом указывалось, что код Хаффмана улучшить невозможно. На первый взгляд это противоречит первой теореме Шеннона, утверждающей, что всегда можно предложить способ кодирования, при котором избыточность будет сколь угодно малой величиной. На самом деле это противоречие возникло из-за того, что до сих пор мы ограни-

чивали себя алфавитным кодированием. При алфавитном кодировании передаваемое сообщение представляет собой последовательность кодов отдельных знаков первичного алфавита. Однако возможны варианты кодирования, при которых кодовый знак относится сразу к нескольким буквам первичного алфавита (будем называть такую комбинацию *блоком*) или даже к целому слову первичного языка. Кодирование блоков понижает избыточность. В этом легко убедиться на простом примере.

**Пример 3.2.**

Пусть первичный алфавит состоит из двух знаков $a$ и $b$ с вероятностями 0,75 и 0,25 соответственно. Необходимо сравнить избыточность кода Хаффмана при алфавитном и блочном двухбуквенном кодировании.

При алфавитном кодировании:

| Знак | $p_i$ | Код |
|---|---|---|
| a | 0,75 | 0 |
| b | 0,25 | 1 |

$I(A) = 0{,}811$; $K(A,2) = 1$; $Q(A,2) = 0{,}233$.

При блочном двухбуквенном кодировании различных сочетаний может быть 4; если сообщение является шенновским, что для каждой пары $p_{ij} = p_i p_j$. В результате алфавит $(A')$ состоит из 4-х знаков с известными вероятностями; применяя к нему метод Хаффмана, получаем коды:

| Знак | $p_{ij}$ | Код |
|---|---|---|
| aa | 0,562 | 0 |
| ab | 0,188 | 11 |
| ba | 0,188 | 100 |
| bb | 0,062 | 101 |

$I(A') = 1{,}623$ (в пересчете на 1 знак — 0,811); $K(A',2) = 1{,}688$ (в пересчете на знак — 0,844); $Q(A',2) = 0{,}040$.

Таким образом, блочное кодирование обеспечивает построение более сжатого кода, чем алфавитное. При использовании блоков большей длины (трехбуквенных и более) избыточность стремится к нулю в полном соответствии с первой теоремой Шеннона.

Цель сжатия — уменьшение количества бит, необходимых для хранения или передачи заданной информации, что дает возможность передавать сообщения более быстро и хранить более

экономно и оперативно (последнее означает, что операция извлечения данной информации с устройства ее хранения будет проходить быстрее, что возможно, если скорость распаковки данных выше скорости считывания данных с носителя информации). Сжатие позволяет, например, записать больше информации на носитель, эффективно увеличить размер жесткого диска, ускорить работу с модемом и т. д. При работе с компьютерами широко используются программы-архиваторы данных формата ZIP, GZ, ARJ и других. Методы сжатия информации были разработаны как математическая теория, которая долгое время (до первой половины 80-х годов) мало использовалась в компьютерах на практике.

Однако следует сознавать, что, несмотря на кажущиеся преимущества, применение блочного кодирования создает дополнительные проблемы, поскольку требует, наряду с передачей информационных бит, пересылки на приемный конец и таблицы блочных кодов, без чего декодирование оказывается невозможным.

## Контрольные вопросы и задания к гл. 3

1. Приведите примеры обратимого и необратимого кодирования помимо рассмотренных в тексте.

2. В чем значение первой теоремы Шеннона для кодирования?

3. Первичный алфавит содержит 8 знаков с вероятностями: «пробел» — 0,25; «?» — 0,18; «&» — 0,15; «*» — 0,12; «+» — 0,1; «%» — 0,08; «#» — 0,07 и «!» — 0,05. В соответствии с правилами, изложенными в п. 3.2.1, предложите вариант неравномерного алфавитного двоичного кода с разделителем знаков, а также постройте коды Шеннона–Фано и Хаффмана; сравните их избыточности.

4. Постройте в виде блок-схемы последовательность действий устройства, производящего декодирование сообщения, коды которого удовлетворяют условию Фано. Реализуйте программно на каком-либо языке программирования.

5. Является ли кодирование по методу Шеннона–Фано и по методу Хаффмана однозначным? Докажите на примере алфавита $A$, описанного в п. 3.2.3.

6. С помощью электронных таблиц проверьте правильность данных о средней длине кода $K(A, 2)$ и избыточности кода для всех обсуждавшихся в п. 3.2.2 примерах неравномерного алфавитного кодирования (для русского и английского алфавитов).

7. Для алфавита, описанного в п. 3.2.3, постройте вариант минимального равномерного кода.

8. Проверьте данные об избыточность кода Бодо для русского и английских языков, приведенных в п. 3.2.1. Почему избыточность кода для русского языка меньше?

9. Почему в 1 байте содержится 8 бит?

10. Оцените, какое количество книг объемом 200 страниц может поместиться:

а) на дискете 1,44 Мб;

б) в ОЗУ компьютера 32 Мб?

в) на оптическом CD-диске емкостью 650 Мб?

г) на жестком магнитном диске винчестера емкостью 40 Гб?

11. Почему в компьютерных устройствах используется байтовое кодирование?

12. Что такое «*лексикографический порядок кодов*»? Чем он удобен?

13. Для цифр придумайте вариант байтового кодирования. Реализуйте процедуру кодирования программно (ввод — последовательность цифр; вывод — последовательность двоичных кодов в соответствии с разработанной кодовой таблицей).

14. Разработайте программу декодирования байтовых кодов из задачи 13.

15. Продолжите пример 3.2, определив избыточность при трехбуквенном блочном кодировании.

16. Почему код Морзе для русского алфавита имеет бóльшую избыточность, чем для английского?

17. Код Морзе для цифр следующий:

| | | | |
|---|---|---|---|
| 0 | − − − − − | 5 | · · · · · |
| 1 | · − − − − | 6 | − · · · · |
| 2 | · · − − − | 7 | − − · · · |
| 3 | · · · − − | 8 | − − − · · |
| 4 | · · · · − | 9 | − − − − · |

Считая алфавит цифр самостоятельным, а появление различных цифр равновероятным, найдите избыточность кода Морзе для цифрового алфавита.

18. В лексиконе «людоедки» Эллочки Щукиной из романа Ильфа и Петрова «12 стульев» было 17 словосочетаний («Хо-хо!», «Ого!», «Блеск!», «Шутишь, парниша», «У вас вся спина белая» и пр).

а) Определите длину кода при равномерном словесном кодировании.

б) Предложите вариант равномерного кодирования данного словарного запаса.

# 4 Представление и обработка чисел в компьютере

Безусловно, одним из основных направлений применения компьютеров были и остаются разнообразные вычисления. Обработка числовой информации ведется и при решении задач, на первый взгляд не связанных с какими-то расчетами, например при использовании компьютерной графики или звука. В связи с этим встает вопрос о выборе оптимального представления чисел в компьютере. Конечно, можно было бы использовать 8-битное (байтовое) кодирование отдельных цифр, а из них составлять числа. Однако такое кодирование не будет оптимальным, что легко увидеть из простого примера: пусть имеется двузначное число 13; при 8-битном кодировании отдельных цифр в кодах ASCII его представление выглядит следующим образом: 0011000100110011, т. е. код имеет длину 16 бит; если же определять это число посредством двоичного выборного каскада (например, используя выборочный каскад «*Угадай-ка-16*», подобный описанному в п. 2.2), то получим 4-битную цепочку 1101. Важно, что представление определяет не только способ записи данных (букв или чисел), но и допустимый набор операций над ними; в частности, буквы могут быть только помещены в некоторую последовательность (или исключены из нее) без изменения их самих; над числами же возможны операции, изменяющие само число, например извлечение корня или сложение с другим числом. Представление чисел в компьютере по сравнению с формами, известными всем со школы, имеет два важных отличия:

во-первых, числа записываются в двоичной системе счисления (в отличие от привычной десятичной);

во-вторых, для записи и обработки чисел отводится конечное количество разрядов (в «некомпьютерной» арифметике такое ограничение отсутствует).

Следствия, к которым приводят эти отличия, и рассматриваются в данной главе.

## 4.1. Системы счисления

Начнем с некоторых общих замечаний относительно понятия *число*. Можно считать, что любое число имеет *значение* (содержание) и *форму представления**. Значение числа задает его отношение к значениям других чисел («*больше*», «*меньше*», «*равно*») и, следовательно, порядок расположения чисел на числовой оси. Форма представления, как следует из названия, определяет порядок записи числа с помощью предназначенных для этого знаков. При этом значение числа является *инвариантом*, т. е. *не зависит от способа его представления*. Это подразумевает также, что число с одним и тем же значением может быть записано по-разному, т. е. *отсутствует взаимно однозначное соответствие между представлением числа и его значением*. В связи с этим возникают вопросы, во-первых, о формах представления чисел и, во-вторых, о возможности и способах перехода от одной формы к другой.

Способ представления числа определяется *системой счисления*.

> ***Система счисления*** *— это правило записи чисел с помощью заданного набора специальных знаков — цифр.*

Людьми использовались различные способы записи чисел, которые можно объединить в несколько групп: *унарная*, *непозиционные* и *позиционные*.

**Унарная** — это система счисления, в которой для записи чисел используется только один знак — I («*палочка*»). Следующее число получается из предыдущего добавлением новой I; их количество (сумма) равно самому числу. Именно такая система применяется для начального обучения счету детей (можно вспомнить «*счетные палочки*»); использование унарной системы оказывается важным педагогическим приемом для введения детей в мир чисел и действий с ними. Но, как мы увидим в дальнейшем, унарная система важна также в теоретическом отношении, поскольку в ней число представляется наиболее естественным способом и, следовательно, просты операции с ним. Кроме того, именно унарная система определяет значение целого числа количеством содержащихся в нем единиц, которое, как было сказано,

---

* Ситуация весьма напоминает порядок использованием переменных в программах — они тоже имеют значение и имя. Эта аналогия подчеркивает общность подхода к представлению данных независимо от того, кому (или чему) эти данные предназначены.

не зависит от формы представления. Для записи числа в унарной системе в дальнейшем будем использовать обозначение $Z_1$.

Из *непозиционных* наиболее распространенной в настоящее время можно считать *римскую* систему счисления. В ней некоторые базовые числа обозначены заглавными латинскими буквами: 1 — I, 5 — V, 10 — X, 50 — L, 100 — C, 500 — D, 1000 — M. Все другие числа строятся комбинаций базовых в соответствии со следующими правилами:

- если цифра меньшего значения стоит справа от большей цифры, то их значения суммируются; если слева — то меньшее значение вычитается из большего.
- цифры I, X, C и M могут следовать подряд не более трех раз каждая;
- цифры V, L и D могут использоваться в записи числа не более одного раза.

Например, запись XIX соответствует числу 19, MDXLIX — числу 1549. Запись чисел в такой системе громоздка и неудобна, но еще более неудобным оказывается выполнение в ней даже самых простых арифметических операций. Отсутствие нуля и знаков для чисел больше M не позволяют римскими цифрами записать любое число (хотя бы натуральное). По указанным причинам теперь римская система используется лишь для нумерации.

В настоящее время для представления чисел применяют *позиционные* системы счисления.

> ***Позиционными** называются системы счисления, в которых значение каждой цифры в изображении числа определяется ее положением (позицией) в ряду других цифр.*

Наиболее распространенной и привычной является система счисления, в которой для записи чисел используется 10 цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8 и 9. Число представляет собой краткую запись многочлена, в который входят степени некоторого другого числа — *основания системы счисления*. Например,

$$272{,}12 = 2 \cdot 10^2 + 7 \cdot 10^1 + 2 \cdot 10^0 + 1 \cdot 10^{-1} + 2 \cdot 10^{-2}.$$

В данном числе цифра 2 встречается трижды, однако значение (вклад в число) этих цифр различно и определяется их положением (*позицией*) в числе. Количество цифр для построения чисел, очевидно, равно основанию системы счисления. Также очевидно, что максимальная цифра на 1 меньше основания. Причина широкого распространения именно десятичной системы

счисления понятна — она происходит от унарной системы с пальцами рук в качестве «палочек». Однако в истории человечества имеются свидетельства использования и других систем счисления — пятиричной, шестиричной, двенадцатиричной, двадцатиричной и даже шестидесятиричной — об этом можно прочитать, например, в книге С.В. Фомина [42].

Общим для унарной и римской систем счисления является то, что значение числа в них определяется посредством операций сложения и вычитания базисных цифр, из которых составлено число, независимо от их позиции в числе. Такие системы получили название *аддитивных*. В отличие от них позиционное представление следует считать *аддитивно-мультипликативным*, поскольку значение числа определяется операциями умножения и сложения. Главной же особенностью позиционного представления является то, что в нем *посредством конечного набора знаков* (цифр, разделителя десятичных разрядов и обозначения знака числа) *можно записать неограниченное количество различных чисел*. Кроме того, в позиционных системах гораздо легче, чем в аддитивных, осуществляются операции умножения и деления. Именно эти обстоятельства обуславливают доминирование позиционных систем при обработке чисел как человеком, так и компьютером.

По принципу, положенному в основу десятичной системы счисления, очевидно, можно построить системы с иным основанием. Пусть $p$ — основание системы счисления. Тогда любое число $Z$ (пока ограничимся только целыми числами), удовлетворяющее условию $Z < p^k$ ($k \geqslant 0$, целое), может быть представлено в виде многочлена со степенями $p$ (при этом, очевидно, максимальный показатель степени будет равен $k-1$):

$$Z_p = a_{k-1}p^{k-1} + a_{k-2}p^{k-2} + ... + a_1p^1 + a_0p^0 = \sum_{j=0}^{k-1} a_j p^j. \qquad (4.1)$$

Из коэффициентов $a_j$ при степенях основания строится сокращенная запись числа:

$$Z_p = (a_{k-1}a_{k-2}\ldots a_1a_0).$$

Индекс $p$ у числа $Z$ указывает, что оно записано в системе счисления с основанием $p$; общее число цифр числа равно $k$. Все коэффициенты $a_j$ — целые числа, удовлетворяющие условию $0 \leqslant a_j \leqslant p-1$.

Уместно задаться вопросом: каково минимальное значение $p$? $p = 1$ невозможно, поскольку тогда все $a_j = 0$ и форма (4.1) теряет смысл. Первое допустимое значение $p = 2$ — оно и является минимальным для позиционных систем. Система счисления с основанием 2 называется *двоичной*. Цифрами двоичной системы являются 0 и 1, а форма (4.1) строится по степеням 2. Интерес именно к этой системе счисления связан с тем, что, как указывалось выше, любая информация в компьютерах представляется с помощью двух состояний — 0 и 1, которые легко реализуются технически. Наряду с двоичной в компьютерах используются 8-ричная и 16-ричная системы счисления — причины будут рассмотрены далее.

Необходимо еще раз подчеркнуть, что значение целого числа, т. е. *общее количество входящих в него единиц*, не зависит от способа его представления и остается одинаковым во всех системах счисления; различаются только формы представления одного и того же количественного содержания числа. Например, $\mathrm{IIIII}_1 = 5_{10} = 101_2 = 10_5 = 5_6 = 5_{16}$. Другими словами, размер стипендии или зарплаты не станет больше от того, что при ее начислении вместо десятичной будут использовать, например, двоичную систему счисления.

## 4.2. Представление чисел в различных системах счисления

### 4.2.1. Перевод целых чисел из одной системы счисления в другую

Поскольку одно и то же число может быть записано в различных системах счисления, встает вопрос о переводе представления числа из одной системы ($p$) в другую ($q$) — будем обозначать такое преобразование $Z_p \to Z_q$. Теоретически можно произвести его при любых $q$ и $p$. Однако подобный прямой перевод будет затруднен тем, что придется выполнять операции по правилам арифметики недесятичных систем счисления. По этой причине более удобными с практической точки зрения оказываются варианты преобразования с промежуточным переводом $Z_p \to Z_r \to Z_q$ с основанием $r$, для которого арифметические операции выполнить легко. Такими удобными основаниями являются $r = 1$ и $r = 10$, т. е. перевод осуществляется через унарную или десятичную систему счисления.

**Преобразование $Z_p \to Z_1 \to Z_q$.** Идея алгоритма перевода предельно проста: положим начальное значение $Z_q := 0$; из числа

$Z_p$ вычтем 1 по правилам вычитания системы $p$, т. е. $Z_p := Z_p - 1^*$ и добавим ее к $Z_q$ по правилам сложения системы $q$, т. е. $Z_q := Z_q + 1$; будем повторять эту последовательность действий, пока не достигнем $Z_p = 0$.

Правила сложения с 1 и вычитания 1 могут быть записаны следующим образом:

| Для системы $p$ | Для системы $q$ |
|---|---|
| $(p-1)-1=(p-2)$ | $0+1=1$ |
| $(p-2)-1=(p-3)$ | $1+1=2$ |
| ... | ... |
| $1-1=0$ | $(q-2)+1=(q-1)$ |
| $0-1=\overset{1}{\curvearrowright}(p-1)$ | $(q-1)+1=\overset{1}{\curvearrowleft}0$ |

Промежуточный переход к унарной системе счисления в данном случае осуществляется неявно — используется упоминавшееся выше свойство независимости значения числа от формы его представления. Рассмотренный алгоритм перевода может быть легко реализован программным путем, в частности машиной Тьюринга (см. п. 9.3.3).

Выполнить преобразование $22_3 \to Z_6$. Последовательность действий и промежуточные результаты для наглядности представим в виде таблицы:

| Шаг | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| $Z_3 - 1$ | 22 | 21 | 20 | 12 | 11 | 10 | 2 | 1 | 0 |
| $Z_6 + 1$ | 0 | 1 | 2 | 3 | 4 | 5 | 10 | 11 | **12** |

Следовательно, $22_3 = 12_6$.

**Преобразование $Z_p \to Z_{10} \to Z_q$.** Очевидно, первая и вторая часть преобразования не связаны друг с другом, что дает основание рассматривать их по отдельности.

Алгоритмы перевода $Z_{10} \to Z_q$ вытекают из следующих соображений. Многочлен (4.1) для $Z_q$ может быть представлен таким образом**:

$$Z_q = \sum_{j=0}^{m-1} b_j q^j = (...(b_{m-1}q + b_{m-2})q + ... + b_1)q + b_0, \qquad (4.2)$$

---

* Знак «:=» используется здесь и далее в смысле «*присвоить*» («*считать равным*»).

** Такое представление называется схемой Горнера.

где $m$ — число разрядов в записи $Z_q$; $b_j$, $j = 0, \ldots, m-1$, — цифры числа $Z_q$.

Разделим число $Z_q$ на две части по $i$-му разряду; число, включающее $m - i$ разрядов с $(m-1)$-го по $i$-й обозначим $\gamma_i$, а число с $i$-го по 0-й разряды — $\delta_i$:

$$Z_q = (\underbrace{b_{m-1}b_{m-2}...b_i}_{\gamma_i}\underbrace{b_{i-1}...b_1b_0}_{\delta_i})_q.$$

Очевидно, $I \in [0, m-1]$, $\gamma_0 = \delta_{m-1} = Z_q$. Для произвольного $i$

$$\gamma_i = \underbrace{(...((b_{m-1}q + b_{m-2})q + ... + b_{i+1})}_{\gamma_{i+1}}q + b_i = \gamma_{i+1}q + b_i.$$

Позаимствуем из языка PASCAL обозначение двух операций: div — результат целочисленного деления двух целых чисел и mod — остаток от целочисленного деления (13 div 4 = 3; 13 mod 4 = 1). Теперь если принять $\gamma_{m-1} = b_{m-1}$, то в (4.2) усматривается следующее рекуррентное соотношение: $\gamma_i = \gamma_{i+1}q + b_i$, из которого, в свою очередь, получаются выражения:

$$\gamma_{i+1} = \gamma_i \operatorname{div} q; \quad b_i = \gamma_i \bmod q. \tag{4.3}$$

Аналогично, если принять $\delta_0 = b_0$, то для правой части числа будет справедливо

$$\delta_i = b_i q^i + \underbrace{b_{i-1}q^{i-1} + ... + b_1 q + b_0}_{\delta_{i-1}} = b_i q^i + \delta_{i-1},$$

из чего следует другое рекуррентное соотношение: $\delta_i = \delta_{i-1} + b_i q^i$, из которого, в свою очередь, вытекает:

$$b_i = \delta_i \operatorname{div} q^i; \quad \delta_{i-1} = \delta_i \bmod q^i. \tag{4.4}$$

Из соотношений (4.3) и (4.4) непосредственно вытекают два способа перевода целых чисел из 10-й системы счисления в систему с произвольным основанием $q$.

**Способ 1** является следствием соотношений (4.3), из которых просматривается следующий алгоритм перевода:

1) целочисленно разделить исходное число $Z_{10}$ на основание новой системы счисления $q$ и найти остаток от деления — это будет цифра 0-го разряда числа $Z_q$;

2) частное от деления снова целочисленно разделить на $q$ с выделением остатка; процедуру продолжать до тех пор, пока частное от деления не окажется меньше $q$;

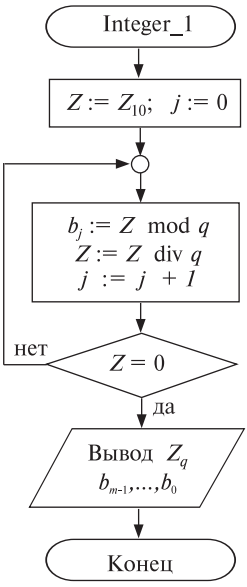



**Рис. 4.1.** Алгоритм перевода $Z \to q$

3) образовавшиеся остатки от деления, поставленные в порядке, обратном порядку их получения, и представляют $Z_q$.

Блок-схема алгоритма представлена на рис. 4.1. Обычно его представляют в виде «*лестницы*».

**Пример 4.2.**

Выполнить преобразование $123_{10} \to Z_5$.

$$\begin{array}{r|l} 123 & 5 \\ \hline 120 & 24 \quad 5 \\ \hline 3 & 20 \quad 4 \\ & 4 \end{array}$$

Остатки от деления (3, 4) и результат последнего целочисленного деления (4) образуют обратный порядок цифр нового числа. Следовательно, $123_{10} = 443_5$.

Необходимо заметить, что полученное число нельзя читать «*четыреста сорок три*», поскольку десятки, сотни, тысячи и прочие подобные обозначения чисел относятся только к десятичной системе счисления. Прочитывать число следует простым перечислением его цифр с указанием системы счисления («*число четыре, четыре, три в пятиричной системе счисления*»).

**Способ 2** вытекает из соотношения (4.4); действия производятся в соответствии со следующим алгоритмом:

1) определить $m-1$ — максимальный показатель степени в представления числа по форме (4.1) для основания $q$;

2) целочисленно разделить исходное число $Z_{10}$ на основание новой системы счисления в степени $m-1$ (т. е. $q^{m-1}$ и найти остаток от деления; результат деления определит первую цифру числа $Z_q$;

3) остаток от деления целочисленно разделить на $q^{m-2}$, результат деления принять за вторую цифру нового числа; найти остаток; продолжать эту последовательность действий, пока показатель степени $q$ не достигнет значения 0.

Продемонстрируем действие алгоритма на той же задаче, что была рассмотрена выше. Определить $m-1$ можно либо подбором ($5^0 = 1 < 123$; $5^1 = 5 < 123$; $5^2 = 25 < 123$; $5^3 = 125 > 123$, следовательно, $m-1 = 2$), либо логарифмированием с оставлением целой части логарифма ($\log_5 123 = 2{,}99$, т. е. $m-1 = 2$).

Далее:

$$b_2 = 123 \operatorname{div} 5^2 = \mathbf{4}; \quad \delta_1 = 123 \bmod 5^2 = 23 \ (i = 2 - 1 = 1);$$
$$b_1 = 23 \operatorname{div} 5^1 = \mathbf{4}; \quad \delta_0 = 23 \bmod 5^1 = \mathbf{3} \ (i = 0).$$

Алгоритмы перевода $Z_p \to Z_{10}$ явно вытекает из выражений (4.1) или (4.2): необходимо $Z_p$ представить в форме многочлена и выполнить все операции по правилам десятичной арифметики.

**Пример 4.3.**

Выполнить преобразование $443_5 \to Z_{10}$:

$$443_5 = 4 \cdot 5^2 + 4 \cdot 5^1 + 3 \cdot 5^0 = 4 \cdot 25 + 4 \cdot 5 + 3 \cdot 1 = 123_{10}.$$

Необходимо еще раз подчеркнуть, что приведенными алгоритмами удобно пользоваться при переводе числа из десятичной системы в какую-то иную или наоборот. Они работают и для перевода между любыми иными системами счисления, однако преобразование будет затруднено тем, что все арифметические операции придется осуществлять по правилам исходной (в первых алгоритмах) или конечной (в последнем алгоритме) системы счисления. По этой причине переход, например, $Z_3 \to Z_8$ проще осуществить через промежуточное преобразование к 10-й системе $Z_3 \to Z_{10} \to Z_8$. Ситуация, однако, значительно упрощается, если основания исходной и конечной систем счисления оказываются связанными соотношением $p = q^r$, где $r$ — целое число (естественно, большее 1) или $r = 1/n$ ($n > 1$, целое) — эти случаи будут рассмотрены ниже.

### 4.2.2. Перевод дробных чисел из одной системы счисления в другую

Вещественное число, в общем случае содержащее целую и дробную часть, всегда можно представить в виде суммы целого числа и правильной дроби. Поскольку в предыдущем разделе проблема записи натуральных чисел в различных системах счисления уже была решена, можно ограничить рассмотрение только алгоритмами перевода правильных дробей. Введем следующие обозначения: правильную дробь в исходной системе счисления $p$ будем записывать в виде $0{,}Y_p$, дробь в системе $q$ — $0{,}Y_q$, а преобразование — в виде $0{,}Y_p \to 0{,}Y_q$. Последовательность рассуждений весьма напоминает проведенную ранее для натуральных чисел. В частности, это касается рекомендации осуществлять преобразование через промежуточный переход к 10-й системе, чтобы избежать необходимости проводить вычисления в «непривычных» системах счисления, т. е. $0{,}Y_p \to 0{,}Y_{10} \to 0{,}Y_q$. Это, в

свою очередь, разбивает задачу на две составляющие: преобразование $0{,}Y_p \to 0{,}Y_{10}$ и $0{,}Y_{10} \to 0{,}Y_q$, каждое из которых может рассматриваться независимо.

Алгоритмы перевода $0{,}Y_{10} \to 0{,}Y_q$ выводится путем следующих рассуждений. Если основание системы счисления $q$, простая дробь содержит $n$ цифр и $b_k$ — цифры дроби ($1 \leqslant k \leqslant n$, $0 \leqslant b_k \leqslant q-1$), то она может быть представлена в виде суммы:

$$0{,}Y_q = \sum_{k=i}^{n} b_k q^{-k} = \frac{1}{q}\left(b_1 + \frac{1}{q}\left(b_2 + \ldots + \frac{1}{q}\left(b_{n-1} + \frac{1}{q}b_n\right)\right)\ldots\right). \tag{4.5}$$

Часть дроби от разряда $i$ до ее конца обозначим $\varepsilon_i$:

$$0{,}Y_q = (0, b_1 b_2 \ldots \underbrace{b_i b_{i+1} \ldots b_n}_{\varepsilon})_q$$

и примем $\varepsilon_n = b_n/q$ (очевидно, $\varepsilon_1 = 0{,}Y_q$), тогда

$$\varepsilon_i = \frac{1}{q}\left(b_i + \underbrace{\frac{1}{q}\left(b_{i+1} + \frac{1}{q}\left(b_{i+2} + \ldots + \frac{1}{q}b_n\right)\ldots\right)}_{\varepsilon_{i+1}}\right),$$

откуда легко усматривается рекуррентное соотношение

$$\varepsilon_i = \frac{1}{q}(b_i + \varepsilon_{i+1}). \tag{4.6}$$

Если вновь позаимствовать в PASCAL'е обозначение функции — на этот раз trunc, которая производит округление целого вещественного числа отбрасыванием его дробной части, то следствием (4.6) будут соотношения, позволяющие находить цифры новой дроби:

$$b_i = \mathrm{trunc}(q\varepsilon_i); \quad \varepsilon_{i+1} = q\varepsilon_i - \mathrm{trunc}(q\varepsilon_i). \tag{4.7}$$

Соотношения (4.7) задают алгоритм преобразования $0{,}Y_{10} \to 0{,}Y_q$:

1) умножить исходную дробь в 10-й системе счисления на $q$, выделить целую часть — она будет первой цифрой новой дроби; отбросить целую часть;

2) для оставшейся дробной части операцию умножения с выделением целой и дробных частей повторять, пока в дробной части не окажется 0 или не будет достигнута желаемая точность конечного числа (*exact*); появляющиеся при этом целые будут цифрами новой дроби;

3) записать дробь в виде последовательности цифр после ноля с разделителем в порядке их появления в п. 1 и 2.

Блок-схема алгоритма представлена на рис. 4.2. Цикл перевода заканчивается либо в том случае, когда окажется $\varepsilon_{i+1} = 0$, либо последовательность действий повторится наперед заданное число раз (значение константы ***ex***), которое совпадает с количеством значащих цифр в дробной части.

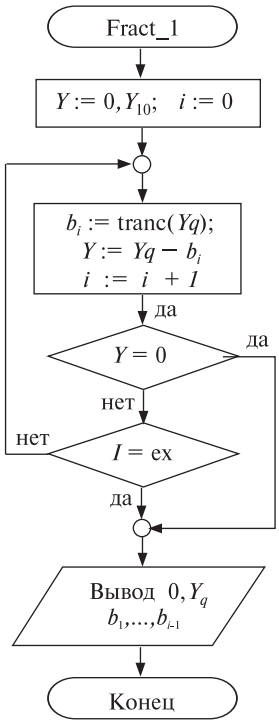



**Рис. 4.2.** Блок-схема алгоритма

**Пример 4.4.**

Выполнить преобразование $0{,}375_{10} \to 0{,}Y_2$:

$$0{,}375 \cdot 2 = \mathbf{0}{,}750$$
$$0{,}75 \cdot 2 = \mathbf{1}{,}50$$
$$0{,}5 \cdot 2 = \mathbf{1}{,}0$$

Таким образом, $0{,}375_{10} = 0{,}011_2$.

Перевод $0{,}Y_p \to 0{,}Y_{10}$, как и в случае натуральных чисел, сводится к вычислению значения формы (4.5) в десятичной системе счисления. Например,

$$0{,}011_2 = 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} = \\ = 0{,}25 + 0{,}125 = 0{,}375_{10}.$$

Следует сознавать, что после перевода дроби, которая была конечной в исходной системе счисления, она может оказаться бесконечной в новой системе. Соответственно, рациональное число в исходной системе может после перехода превратиться в иррациональное. Справедливо и обратное утверждение: число иррациональное в исходной системе счисления в иной системе может оказаться рациональным.

**Пример 4.5.**

Выполнить преобразование $5{,}3(3)_{10} \to X_3$.

Перевод целой части, очевидно, дает $5_{10} = 12_3$. Перевод дробной части: $0{,}3(3)_{10} = 0{,}1_3$. Окончательно: $5{,}3(3)_{10} = 12{,}1_3$.

Как уже было сказано, значение целого числа не зависит от формы его представления и выражает количество входящих в него единиц. Простая дробь имеет смысл доли единицы, и это «*дольное*» содержание также не зависит от выбора способа пред-

ставления. Другими словами, треть пирога остается третью в любой системе счисления.

### 4.2.3. Понятие экономичности системы счисления

Число в системе счисления $p$ с $k$ разрядами, очевидно, будет иметь наибольшее значение в том случае, если все цифры числа окажутся максимальными, т. е. равными $p-1$. Тогда

$$(Z_p)^{\max} = \underbrace{\langle p-1\rangle \ldots \langle p-1\rangle}_{k \text{ цифр}} = p^k - 1. \tag{4.8}$$

Количество разрядов числа при переходе от одной системы счисления к другой в общем случае меняется. Очевидно, если $p = q^\sigma$ ($\sigma$ не обязательно целое), то $(Z_p)^{\max} = p^k - 1 = q^{\sigma k} - 1$. То есть количество разрядов числа в системах счисления $p$ и $q$ будут различаться в $\sigma$ раз. Очевидно соотношение

$$\sigma = \frac{\log p}{\log q}. \tag{4.9}$$

При этом основание логарифма никакого значения не имеет, поскольку $\sigma$ определяется отношением логарифмов. Сравним количество цифр в числе $99_{10}$ и его представлении в двоичной системе счисления $99_{10} = 1100011_2$; т. е. двоичная запись требует 7 цифр вместо 2 в десятичной. $\sigma = \ln 10/ln2 = 3{,}322$; следовательно, количество цифр в десятичном представлении нужно умножить на 3,322 и округлить в большую сторону: $2 \cdot 3{,}322 = 6{,}644 \approx 7$.

Введем понятие *экономичности* представления числа в данной системе счисления.

> *Под **экономичностью системы счисления** будем понимать то количество чисел, которое можно записать в данной системе с помощью определенного количества цифр.*

Речь в данном случае идет не о количестве разрядов, а об общем количестве сочетаний цифр, которые интерпретируются как различные числа. Поясним на примере: пусть в нашем распоряжении имеется 12 цифр. Мы можем разбить их на 6 групп по 2 цифры (0 и 1) и получить шестиразрядное двоичное число; общее количество таких чисел, как уже неоднократно обсуждалось, равно $2^6$. Можно разбить заданное количество цифр на 4 группы по три цифры и воспользоваться троичной системой счисления — в этом случае общее количество различных их сочетаний составит

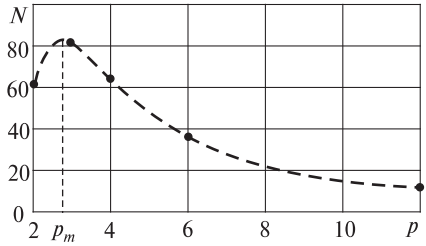



**Рис. 4.3.** Зависимость количества чисел от основания системы счисления при использовании 12 возможных цифр для записи чисел

$3^4$. Аналогично можно произвести другие разбиения; при этом число групп определит разрядность числа, а количество цифр в группе — основание системы счисления. Результаты различных разбиений можно проиллюстрировать таблицей:

| Основание системы счисления $p$ | 2 | 3 | 4 | 6 | 12 |
|---|---|---|---|---|---|
| Разрядность числа $k$ | 6 | 4 | 3 | 2 | 1 |
| Общее количество различных чисел $N$ | $2^6 = 64$ | $3^4 = 81$ | $4^3 = 64$ | $6^2 = 36$ | $12^1 = 12$ |

Из приведенных оценок видно, что *наиболее экономичной оказывается троичная система счисления*, причем результат будет тем же, если исследовать случаи с другим исходным количеством сочетаний цифр.

Точное расположение максимума экономичности может быть установлено следующими рассуждениями. Пусть имеется $n$ знаков для записи чисел, а основание системы счисления $p$. Тогда количество разрядов числа $k = n/p$, а общее количество чисел ($N$), которые могут быть составлены,

$$N = p^{n/p}. \tag{4.10}$$

Если считать $N(p)$ непрерывной функцией, то можно найти то значение $p_m$, при котором $N$ принимает максимальное значение. Функция имеет вид, представленный на рис. 4.3.

Для нахождения положения максимума нужно найти производную функции $N(p)$, приравнять ее к нулю и решить полученное уравнение относительно $p$:

$$\frac{dN}{dp} = -\frac{n}{p^2}p^{n/p}\ln p + \frac{n}{p}p^{n/p} - 1 = np^{n/p} - 2(1 - \ln p). \tag{4.11}$$

Приравнивая полученное выражение к нулю, получаем $lnp = 1$, или $p_m = e$, где $e = 2{,}71828\ldots$ — основание натурального логарифма. Ближайшее к e целое число, очевидно, 3 — по этой причине троичная система счисления оказывается самой экономичной для представления чисел. В 60-х годах в нашей стране была построена вычислительная машина «Сетунь», которая работала в троичной системе счисления. Предпочтение все же отдается двоичной системе, поскольку по экономичности она оказывается второй за троичной, а технически она реализуется гораздо проще остальных. Таким образом, простота технических решений оказывается не единственным аргументом в пользу применения двоичной системы в компьютерах.

### 4.2.4. Перевод чисел между системами счисления 2 ↔ 8 ↔ 16

Интерес к двоичной системе счисления вызван тем, что именно эта система используется для представления чисел в компьютере. Однако двоичная запись оказывается громоздкой, поскольку содержит много цифр и, кроме того, плохо воспринимается и запоминается человеком из-за зрительной однородности (все число состоит из нулей и единиц). Поэтому в нумерации ячеек памяти компьютера, записи кодов команд, нумерации регистров и устройств и пр. используются системы счисления с основаниями 8 и 16; выбор именно этих систем счисления обусловлен тем, что переход от них к двоичной системе и обратно осуществляется, как будет показано ниже, весьма простым образом.

*Двоичная система счисления* имеет основанием 2 и, соответственно, 2 цифры: 0 и 1.

*Восьмеричная система счисления* имеет основание 8 и цифры 0, 1,..., 7.

*Шестнадцатеричная система счисления* имеет основание 16 и цифры 0, 1, ..., 9, A, B, C, D, E, F. При этом знак «A» является 16-ричной цифрой, соответствующей числу 10 в десятичной системе; $B_{16} = 11_{10}$; $C_{16} = 12_{10}$; $D_{16} = 13_{10}$; $E_{16} = 14_{10}$; $F_{16} = 15_{10}$. Другими словами, в данном случае A–F — это не буквы латинского алфавита, а цифры 16-ричной системы счисления и поэтому они имеют только такое начертание (не могут быть представлены в виде, например, соответствующих строчных букв, как в текстах).

Пользуясь алгоритмами, сформулированными в разд. 4.2.1, можно заполнить табл. 4.1.

Таблица 4.1

Представление чисел в системах счисления

| 10-я | 2-я | 8-ричная | 16-ричная | 10-я | 2-я | 8-ричная | 16-ричная |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 8 | 1000 | 10 | 8 |
| 1 | 1 | 1 | 1 | 9 | 1001 | 11 | 9 |
| 2 | 10 | 2 | 2 | 10 | 1010 | 12 | A |
| 3 | 11 | 3 | 3 | 11 | 1011 | 13 | B |
| 4 | 100 | 4 | 4 | 12 | 1100 | 14 | C |
| 5 | 101 | 5 | 5 | 13 | 1101 | 15 | D |
| 6 | 110 | 6 | 6 | 14 | 1110 | 16 | E |
| 7 | 111 | 7 | 7 | 15 | 1111 | 17 | F |

Докажем две теоремы.

**Теорема 1.** *Для преобразования целого числа $Z_p \to Z_q$ в том случае, если системы счисления связаны соотношением $q = p^r$, где $r$ — целое число большее 1, достаточно $Z_p$ разбить* ***справа налево*** *на группы по $r$ цифр и каждую из них независимо перевести в систему $q$.*

**Доказательство:**

Пусть максимальный показатель степени в записи числа $p$ по форме (4.1) равен $k-1$, причем $2r > k-1 > r$. Тогда

$$Z_p = (a_{k-1} \dots a_1 a_0)_p =$$
$$= a_{k-1}p^{k-1} + a_{k-2}p^{k-2} + \dots + a_j p^j + \dots + a_1 p^1 + a_0 p^0.$$

Вынесем множитель $p^r$ из всех слагаемых, у которых $j \geqslant r$. Получим

$$Z_p = (a_{k-1}p^{k-1-r} + a_{k-2}p^{k-2-r} + \dots + a_r p^0)p^r +$$
$$+ (a_{r-1}p^{r-1} + \dots + a_0 p^0)p^0 = b_1 q^1 + b_0 q^0,$$

где

$$b_1 = a_{k-1}p^{k-1-r} + \dots + a_r p^0 = (a_{k-1} \dots a_r)_p;$$
$$b_0 = a_{r-1}p^{r-1} + \dots + a_0 p^0 = (a_{r-1} \dots a_0)_p.$$

Таким образом, $r$-разрядные числа системы с основанием $p$ оказываются записанными как цифры системы с основанием $q$. Этот результат можно обобщить на ситуацию произвольного $k-1 > r$ — в этом случае выделится не две, а больше $(m)$ цифр числа с основанием $q$. Очевидно, $Z_q = (b_m \dots b_0)_q$.

**Пример 4.6.**

Выполнить преобразование $Z_2 = 110001_2 \to Z_8$. Исходное число разбивается на группы по три разряда справа налево ($8 = 2^3$, следовательно,

$r = 3$) и каждая тройка в соответствии с табл. 4.1 переводится в 8-ричную систему счисления независимо от остальных троек:

$$10110001.$$

Следовательно, $10110001_2 = 261_8$. Аналогично, разбивая $Z_2$ на группы по 4 двоичные цифры и дополняя старшую группу незначащими нулями слева, получим $10110001_2 = B1_{16}$.

**Теорема 2.** *Для преобразования целого числа $Z_p \to Z_q$ в том случае, если системы счисления связаны соотношением $p = q^r$, где $r$ — целое число большее 1, достаточно каждую цифру $Z_p$ заменить соответствующим $r$-разрядным числом в системе счисления $q$, дополняя его при необходимости незначащими нулями слева до группы в $r$ цифр.*

**Доказательство:**

Пусть исходное число содержит две цифры, т. е.

$$Z_p = (a_1 a_0)_p = a_1 p^1 + a_0 p^0.$$

Для каждой цифры справедливо: $0 \leqslant a_i \leqslant p-1$, и поскольку $p = q^r$, то $0 \leqslant a_i \leqslant q^r - 1$; в представлении этих цифр в системе счисления $q$ максимальная степень многочленов (4.1) будет не более $r-1$, и эти многочлены будут содержать по $r$ цифр:

$$a_1 = b_{r-1}^{(1)} q^{r-1} + b_{r-2}^{(1)} q^{r-2} + \ldots + b_0^{(1)} q^0;$$
$$a_0 = b_{r-1}^{(0)} q^{r-1} + b_{r-2}^{(0)} q^{r-2} + \ldots + b_0^{(0)} q^0.$$

Тогда

$$Z_p = (a_1 a_0)_p = (b_{r-1}^{(1)} q^{r-1} + \ldots + b_0^{(1)} q^0) q^r + (b_{r-1}^{(0)} q^{r-1} + \ldots + b_0^{(0)} q^0) q^0 =$$
$$= b_{r-1}^{(1)} q^{2r-1} + \ldots + b_0^{(1)} q^r + b_{r-1}^{(0)} q^{r-1} + \ldots + b_0^{(0)} q^0 = (b_{r-1}^{(1)} \ldots b_0^{(0)})_q = Z_q,$$

причем число $Z_q$ содержит $2r$ цифр. Доказательство легко обобщается на случай произвольного количества цифр в числе $Z_p$.

**Пример 4.7.**

Выполнить преобразование $53_{16} \to Z_2$:

$$53_{16} = \underbrace{101}_{5}\,\underbrace{0011}_{3}{}_2.$$

Переводы $Z_8 \to Z_{16}$ и $Z_{16} \to Z_8$, очевидно, удобнее осуществлять через промежуточный переход к двоичной системе. Например, $172_8 = 001111010_2 = 7A_{16}$.

Подобным же образом происходит перевод дробных числе с той лишь разницей, что он осуществляется от десятичного разделителя вправо.

**Пример 4.8.**

Выполнить преобразование $0{,}D83_{16} \to 0{,}Y_4$.

Имеем $16 = 4^2$, следовательно, $r = 2$ — каждую 16-ричную цифру заменяем двумя четверичными: $\mathrm{D}_{16} \to 31_4$; $8_{16} \to 20_4$; $3_{16} \to 03_4$:

$$0{,}D83_{16} = 0{,}312003_4.$$

Выполнить преобразование $0{,}21012_3 \to 0{,}Y_9$.

Имеем $9 = 3^2$ и $r = 2$ — каждые 2 троичные цифцы заменяем одной девятиричной, дополняя последнюю незначащим нулем: $21_3 = 7_9$; $01_3 = 1_9$; $(2 + 0)_3 = 6_9$:

$$0{,}21012_3 = 0{,}716_9.$$

### 4.2.5. Преобразование нормализованных чисел

Вещественное число $X$ может быть представлено в двух формах — естественной и нормализованной. В *естественной* форме у $X$ имеется целая и дробная части, между которыми помещается *разделитель* (запятая или точка), например 123,4567. Однако такая запись неудобна для слишком больших или, наоборот, слишком малых чисел. Кроме того, использование такой формы (она называется также «*представлением числа с фиксированной запятой*») в компьютере вызвало бы снижение точности вычислений из-за необходимости приведения в соответствие разрядов обрабатываемых чисел и связанных с этим округлений или могло бы породить ситуацию, называемую *переполнением*, когда старший разряд числа не умещается в отведенной разрядной сетке. По указанным причинам вещественные числа в компьютере представляются в *нормализованном* виде (другое название — «*представление числа с плавающей запятой*»), главным достоинством которой является автоматическое масштабирование числа на каждом этапе обработки, что, с одной стороны, обеспечивает максимально возможную точность вычислений, а с другой — избавляет от необходимости принимать меры по предотвращению переполнения (за исключением достаточно экзотических ситуаций с выходом числа за отведенную разрядную сетку). По сути это универсальная форма записи всех чисел, кроме определенных как «тип: целые» (например, Integer, Word или Byte в PASCAL'е).

Сначала ознакомимся с необходимыми понятиями применительно к 10-й системе счисления.

*Число $X_{10}$ называется **нормализованным**, если оно представлено в виде $X_{10} = \pm M_{10} \cdot 10^{\pm k_{10}}$.*

В этой записи $M_{10}$ называется *мантиссой* нормализованного числа; значения мантиссы лежат в интервале $0{,}1 \leqslant M_{10} < 1$. $k_{10}$ называется *порядком* нормализованного числа — это целое положительное десятичное число. Примеры: $-1234_{10} = -0{,}1234 \times \times 10^4$; $0{,}03456_{10} = 0{,}3456 \cdot 10^{-1}$.

Понятие нормализованного числа следует отличать от понятия числа *в нормальной форме*; данная форма достаточно часто используется при записи чисел в математике, физике, технических дисциплинах и отличается от нормализованного представления тем, что мантисса лежит в интервале $1 \leqslant M_{10} < 10$, например постоянная Больцмана $k_{\text{Б}} = 1{,}38 \cdot 10^{-23}$.

При нормализации происходит расчленение «составляющих» числа с выделением знака числа, мантиссы, знака порядка и порядка; как будет показано ниже, это создает определенные удобства при хранении и обработке чисел в компьютере.

Аналогично нормализации десятичного числа можно в нормализованной форме представить и число в произвольной системе счисления $p$:

$$X_p = \pm M_p \cdot p^{\pm k_p}. \tag{4.12}$$

При этом значения мантиссы лежат в интервале $p^{-1} \leqslant M_p < 1$ (т. е. первая значащая цифра мантиссы всегда ненулевая), а показатель степени представляется в системе $p$ ($k_p$). Например, для $p = 2$:

$$X_2 = -101{,}01_2 = -0{,}10101_2 \cdot 2^{11_2}.$$

Мантисса располагается в промежутке $0{,}1_2 \leqslant M_2 < 1$, что соответствует десятичному интервалу $0{,}5_{10} \leqslant M_{10} < 1$.

Подобно задаче о преобразовании целых и дробных чисел, можно поставить задачу о переводе представления числа в нормализованной форме в системе счисления $p$ к нормализованному представлению в системе $q$. Практическое значение такого преобразование состоит в том, что, как было сказано, в компьютере все вещественные числа хранятся и обрабатываются в нормализованном двоичном представлении, и, следовательно, при их вводе осуществляется перевод $X_{10} \to X_2$, а при выводе — обратный перевод $X_2 \to X_{10}$. Однако прежде необходимо рассмотреть, как преобразуется вещественное число из естественной формы в нормализованный вид.

При нормализации различаются ситуации $X_p > 1$ и $X_p < p^{-1}$. В первом случае для нормализации необходимо перемещать разделитель разрядов *влево* по числу до тех пор, пока не исчезнет целая часть числа, но первая цифра после разделителя будет

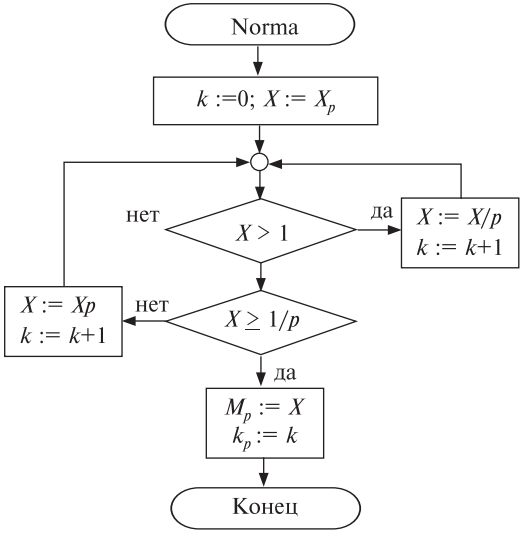



**Рис. 4.4.** Алгоритм нормализации

ненулевой; каждое перемещение разделителя на 1 разряд влево эквивалентно делению числа на $p$, и, чтобы число не менялось, показатель должен возрастать на 1 при каждом сдвиге. Если обозначить эту операцию $N_{\leftarrow}$ (будем называть ее «*нормализация влево*»), то $N_{\leftarrow}[(123{,}45)_{10}] = 0{,}12345_{10}\cdot 10^3$; $N_{\leftarrow}[(23{,}4\cdot 10^5)_{10}] = 0{,}234_{10}\cdot 10^7$; $N_{\leftarrow}[(1212{,}2)_3] = 0{,}12122_3\cdot 3^{11}$. Аналогично можно ввести операцию «*нормализация вправо*» ($N_{\rightarrow}$), обеспечивающая нормализацию чисел меньших $p^{-1}$; очевидно, такие числа необходимо *умножать* на $p$ с одновременным уменьшением показателя на 1 до тех пор, пока первая цифра после разделителя станет ненулевой. Например, $N_{\rightarrow}[(0{,}000101\cdot 2^{-101})_2] = 0{,}101\cdot 2^{-1000}$; $N_{\rightarrow}[(0{,}000987)_{10}] = 0{,}987_{10}\cdot 10^{-3}$. Общий алгоритм нормализации можно изобразить в виде блок-схемы на рис. 4.4.

При практической реализации данного алгоритма не следует забывать, что значение $k_p$ должно изменяться на 1 по правилам арифметики системы счисления $p$. В дальнейшем при необходимости проведения нормализации в ходе каких-либо преобразований будем просто ссылаться на приведенный алгоритм как готовый модуль.

Вернемся к задаче перевода нормализованного числа из одной системы счисления в другую. Пусть имеется число

$$X_p = \pm M_p \cdot p^{\pm k_p},$$

для которого необходимо найти соответствующее ему

$$X_q = \pm M_q \cdot q^{\pm k_q}.$$

Представляется достаточно очевидным, что преобразование не затронет знаков мантиссы и показателя степени. Таким образом, для осуществления преобразования необходимо установить соответствие между $(M_p, k_p)$ и $(M_q, k_q)$. Оно получается достаточно просто, исходя из того, что $X_p = X_q$, откуда следует, что

$$M_q = M_p \frac{p^{k_p}}{q^{k_q}}. \tag{4.13}$$

Из (4.13) вытекает, что для преобразования можно $M_p$ умножить на $p^{k_p}$, т. е. перейти к естественной форме числа в системе $p$, перевести его в систему $q$, а затем нормализовать. Однако в таком варианте действий теряется точность числа и возможно переполнение на промежуточных этапах преобразования. Во избежание этого необходимо чередовать умножение (или деление) на $p$ и нормализацию по основанию $q$. При этом, поскольку все операции выполняются по правилам арифметики в системы $p$, будут получены не $(M_q, k_q)$ в окончательном варианте, а их представления в системе $p$ — обозначим их $(M_q)_p$ и $(k_q)_p$, которые затем нужно будет перевести в систему $q$. Различаются также ситуации $k_p \geqslant 0$ и $k_p < 0$ — в первом случае необходимо умножать начальное и промежуточные значения мантиссы на $p$ и для нормализации делить на $q$, во втором — наоборот. Каждый раз при умножении или делении на $p$ показатель $k_p$ будет менять свой значение на 1; продолжать действия следует до тех пор, пока не выполнится условие $k_p = 0$. Алгоритм действий для ситуации $k_p \geqslant 0$ представлен на рис. 4.5.

**Пример 4.9.**

Выполнить преобразование $X_{10} = 16,5_{10} \to X_2$.

Очевидно, $p = 10$, $q = 2$. Перевод можно осуществить отдельно для целой и дробной части, а затем их объединить, — для нас этот результат послужит эталоном для проверки нового алгоритма. Легко получить, что $16_{10} = 10000_2$, а $0,5_{10} = 0,1_2$; следовательно, $16,5_{10} = 10000,1_2 = (0,100001 \times \times 2^{101})_2$.

Алгоритм Real_1 начинает функционировать после нормализации исходного числа; для этой цели можно воспользоваться алгоритмом Norma; в результате начальными значениями будут $M_{10} = 0,165$; $k_{10} = 2$. Результаты операций будем заносить в таблицу:

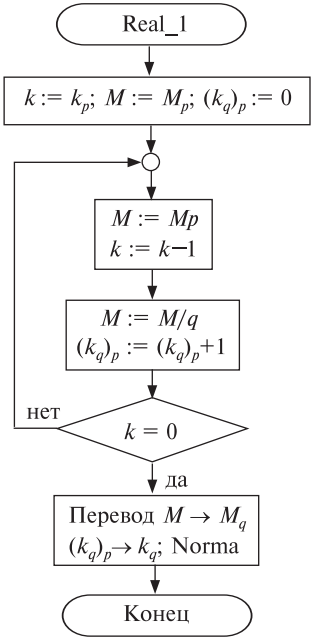



**Рис. 4.5.** Алгоритм 1 нормализации при $k_p \geqslant 0$

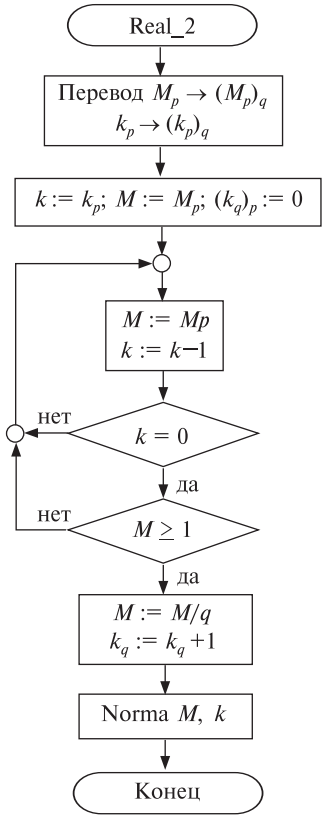



**Рис. 4.6.** Алгоритм 2 нормализации при $k_p \geqslant 0$

| Шаг | Действие | $M = (M_2)_{10}$ | $k_{10}$ | $(k_2)_{10}$ |
|---|---|---|---|---|
| 0 | | 0,165 | 2 | 0 |
| 1 | $M := M \cdot 10$; $k_{10} = k_{10} - 1$ | 1,65 | 1 | 0 |
| 2 | $M := M/2$; $k_2 = k_2 + 1$ | 0,825 | 1 | 1 |
| 3 | $M := M \cdot 10$; $k_{10} = k_{10} - 1$ | 8,25 | 0 | 1 |
| 4 | Так как $k_{10} = 0$, $M \to M_2$ | $1000{,}01_2$ | | 1 |
| 5 | Нормализация по $q = 2$ | $0{,}100001_2$ | | 5 |
| 6 | $(k_2)_{10} \to k_2$ | $0{,}100001_2$ | | 101 |

Окончательно имеем: $X_2 = (0{,}1000012^{101})_2$.

Подобным же будет алгоритм преобразования $X_{10} \to X_2$ и при $k_p < 0$.

Последовательность действий при обратном переводе $X_2 \to$ $\to X_{10}$ отчасти противоположна только что рассмотренной; для $k_p \geqslant 0$ она представлена в виде блок-схемы на рис. 4.6. Нормализация в конце (после достижения $k = 0$) выполняется при необходимости.

**Пример 4.10.**

Выполнить преобразование: $X_2 = (0{,}11 \cdot 2^{110})_2 \to X_{10}$.

Имеем $p = 2$, $q = 10$. Контроль результата: $0{,}11_2 = 0{,}75_{10}$; $(2^{110})_2 = (2^6)_{10} = 64$; следовательно, $(0{,}11 \cdot 2^{110})_2 = 0{,}75 \cdot 64 = 48_{10}$. Для преобразования воспользуемся построенным алгоритмом Real_2 (рис. 4.6). Промежуточные результаты снова будем заносить в таблицу.

| Шаг | Действие | $M = (M_2)_{10}$ | $k_{10}$ | $k = (k_2)_{10}$ |
|---|---|---|---|---|
| 1 | Перевод | 0,75 | 6 | 0 |
| 2 | $M := M \cdot 2$; $k = k - 1$ | 1,5 | 5 | 0 |
| 3 | $M := M/10$; $k_{10} = k_{10} + 1$ | 0,15 | 5 | 1 |
| 4 | $M := M \cdot 2$; $k = k - 1$ | 0,3 | 4 | 1 |
| 5 | $M := M \cdot 2$; $k = k - 1$ | 0,6 | 3 | 1 |
| 6 | $M := M \cdot 2$; $k = k - 1$ | 1,2 | 2 | 1 |
| 7 | $M := M/10$; $k_{10} = k_{10} + 1$ | 0,12 | 2 | 2 |
| 8 | $M := M \cdot 2$; $k = k - 1$ | 0,24 | 1 | 1 |
| 9 | $M := M \cdot 2$; $k = k - 1$ | 0,48 | 0 | 2 |
| 10 | Так как $k = 0$, то $M = M_{10}$ | 0,48 | 0 | 2 |

Таким образом, получаем $(0{,}11 \cdot 2^{110})_2 = 0{,}48 \cdot 10^2 = 48$.

Как и в предыдущем преобразовании, алгоритм в случае $k_p < 0$ будет отличаться только тем, что число в процессе перевода необходимо делить на $q$ и умножать на $p$.

## 4.3. Кодирование чисел в компьютере и действия над ними

В предыдущем разделе мы обсудили возможность представления чисел в двоичной системе счисления. Результатом этого обсуждения могло бы стать следующее резюме:

*двоичное представление возможно; имеется однозначное соответствие между двоичным и любым другим (в частности, десятичным) позиционным представлением; представление возможно как в форме с фиксированной, так и в форме с плавающей запятой; имеются алгоритмы преобразования чисел между системами счисления при различных формах их представления.*

Как указывалось в начале данной главы, второй важной особенностью представления чисел в регистрах и в памяти компьютера является то, что, в отличие от записи числа на бумаге, компьютерные ячейки имеют ограниченный размер и, следовательно, вынуждают использовать при записи чисел и действиях с ними конечное количество разрядов. Это приводит к тому, что бесконечное множество вещественных чисел заменяется конечным множеством их представлений, которые называются *кодами чисел*, а обычные арифметические операции с числами заменяются операциями с кодами. Способы кодирования и допустимые над ними действия оказываются различными для следующих числовых множеств:

- целые положительные числа (целые числа без знака);
- целые числа со знаком;
- вещественные нормализованные числа.

Рассмотрим подробнее перечисленные группы.

### 4.3.1. Кодирование и обработка в компьютере целых чисел без знака

Будем исходить из того, что для записи числа в устройствах компьютера выделяется фиксированное количество двоичных разрядов. Память компьютера имеет байтовую структуру, однако размер одной адресуемой ячейки обычно составляет несколько байт. Например, в компьютерах IBM ячейка памяти объединяет 2 байта (16 двоичных разрядов) — такая комбинация связанных соседних ячеек, обрабатываемая совместно, называется *машинным словом*. Для представления числа в регистре арифметико-логического устройства процессора, где формируется результат операции, имеется еще один дополнительный одноразрядный регистр, который называется *регистром переполнения* и который можно рассматривать в качестве продолжения (т. е. 17-го бита) регистра результата. Назначение этого бита выяснится чуть позже.

Конечный размер разрядной сетки порождает понятие «*наибольшее целое число*», которого в обычном (немашинном) представлении чисел просто не существует. Если количество разрядов $k$ и $p = 2$, то согласно (4.8) $(Z_2)^{\max} = 2^k - 1$. В частности, при $k = 16$ $(Z_2)^{\max} = 2^{16} - 1 = 111111111111111_2 = 65535_{10}$. Другими словами, целого числа, скажем, 65636 и более в компьютере просто не может существовать и, следовательно, появление в ходе вычислений чисел, превышающих $(Z_2)^{\max}$, должно интерпретироваться как ошибка. Минимальным целым числом в беззнаковом

представлении, очевидно, является $(Z_2)^{\min} = 0000000000000000_2 = 0_{10}$. В языке программирования PASCAL целые числа без знака, для записи которых отводится 2 байта, определены как тип Word. Тип устанавливает способ кодирования числа, количество отводимых для записи ячеек памяти (т. е. разрядность числа), а также перечень допустимых операций при обработке. Выход за границу 65535 возможен только за счет увеличения количества разрядов для записи числа, но это порождает новый тип со своим $Z^{\max}$; например, тип Longint* с максимальным значением $2147483647_{10}$, числа которого занимают 4 байта.

Рассмотрим, как с беззнаковыми числами выполняются арифметические операции, не меняющие типа числа; очевидно, к ним относятся сложение и умножение.

**Сложение** выполняется согласно таблице сложения, которая для двоичных чисел имеет вид:

$$0+0=0 \quad 0+1=1$$
$$1+0=1 \quad 1+1=10$$

В последнем случае в том разряде, где находились слагаемые, оказывается 0, а 1 переносится в старший разряд. Место, где сохраняется переносимая в старший разряд 1 до того, как она будет использована в операции, называется *битом переноса* (см. пример 9.1).

**Пример 4.11.**

Найти сумму $9876_{10} + 12345_{10}$ при беззнаковой двоичной кодировке и 16-битном машинном слове. После перевода слагаемых в двоичную систему счисления и выполнения сложения получим (для удобства восприятия 16-разрядное число разобьем на группы по четыре разряда):

| | | |
|---|---|---|
| Переносы: | 1 11 | 111 |
| | 0010 0110 1001 0100 | 9876 |
| + | 0011 0000 0011 1001 | + 12345 |
| Регистр переполнения → | 0 0101 0110 1100 1101 | 22221 |

**Пример 4.12.**

Найти сумму $65534_{10} + 3_{10}$.

| | | |
|---|---|---|
| Переносы: | 111 1111 1111 11 | |
| | 1111 1111 1111 1110 | 65534 |
| + | 0000 0000 0000 0011 | + 3 |
| Регистр переполнения → | 1 0000 0000 0000 0001 | 1 |

* Longint является типом целого числа со знаком.

В последнем примере в результате сложения получилось число, превышающее максимально возможное; результат ошибочен, о чем свидетельствует появление 1 в регистре переполнения. Возникновение такой ситуации в ходе выполнения программы, написанной на языке, где предусмотрено строгое описание типа переменных, приводит к прекращению работы и выводу сообщения об ошибке. В программах, предназначенных для обработки числовой информации (например, Excel, MathCAD или Calc), при переполнении разрядной сетки целое число автоматически преобразуется в вещественный тип. Таким образом, регистр переноса в данном случае служит индикатором корректности процесса вычислений.

**Умножение** выполняется согласно таблице умножения, которая для двоичных чисел имеет предельно простой вид:

$$0 \cdot 0 = 0 \quad 0 \cdot 1 = 0$$
$$1 \cdot 0 = 0 \quad 1 \cdot 1 = 1$$

**Пример 4.13.**

Найти произведение $13_{10} \cdot 5_{10}$. Операции выполнить в двоичной системе счисления.

$$\begin{array}{rr} & 1101 \\ \times & 101 \\ \hline & 1101 \\ + & 1101 \\ \hline & 1000001_2 = 65_{10} \end{array}$$

Таким образом, умножение двоичных чисел сводится к операциям сдвига на один двоичный разряд влево и повторения первого сомножителя в тех разрядах, где второй сомножитель содержит 1, и сдвига без повторения в разрядах с 0. Сдвиг всегда чередуется со сложением из-за ограниченности числа регистров, которые имеются в процессоре для размещения операндов. Другими словами, реализации отдельной операции умножения в процессоре не требуется.

Как и в операции сложения, при умножении чисел с ограниченной разрядной сеткой может возникнуть переполнение. Решается проблема рассмотренными выше способами.

### 4.3.2. Кодирование и обработка в компьютере целых чисел со знаком

Кодирование целых чисел, имеющих знак, можно осуществить двумя способами. В первом варианте один (старший) разряд

машинном слове отводится для записи знака числа; при этом условились кодировать знак «+» *нулем*, знак «−» — *единицей*. Под запись самого числа, очевидно, остается 15 двоичных разрядов, что обеспечивает наибольшее значение числа $Z^{\max} = 2^{15}-1 = 32767_{10}$. Такое представление чисел называется *прямым кодом*. Однако его применение усложняет порядок обработки чисел; например, операция сложения двух чисел с разными знаками должна быть заменена операцией вычитания меньшего из большего с последующим присвоением результату знака большего по модулю числа. Другими словами, операция сопровождается большим количеством проверок условий и выработкой признаков, в соответствии с которыми выбирается то или иное действие.

Альтернативным вариантом является представление чисел со знаком в *дополнительном коде*. Идея построения дополнительного кода достаточно проста: на оси целых положительных чисел, помещающихся в машинное слово (0–65535), сместим положение «0» на середину интервала; числа, попадающие в первую половину (0–32767) будем считать положительными, а числа из второй половины (32768–65535) — отрицательными. В этом случае судить о знаке числа можно будет по его величине и в явном виде выделение знака не потребуется. Например, $1000000000000001_2 = 32769_{10}$ является кодом отрицательного числа, а $0000000000000001_2 = 1_{10}$ — кодом положительного. Принадлежность к интервалу кодов положительных или отрицательных чисел видна по состоянию старшего бита — у кодов положительных чисел его значение 0, отрицательных — 1. Это напоминает представление со знаком, но не является таковым, поскольку используется другой принцип кодирования. Его применение позволяет заменить вычитание чисел их суммированием в дополнительном коде. Мы убедимся в этом чуть позднее после того, как обсудим способ построения дополнительного кода целых чисел.

***Дополнением** $(D)$ $k$-разрядного целого числа $Z$ в системе счисления $p$ называется величина $D(Z_p, k) = p^k - Z$.*

Данную формулу можно представить в ином виде:

$$D(Z_p, k) = ((p^k - 1) - Z) + 1.$$

Число $p^k - 1$ согласно (4.8) состоит из $k$ наибольших в данной системе счисления цифр $(p-1)$, например $9999_{10}$, $FFF_{16}$ или $1111111_2$. Поэтому $(p^k - 1) - Z$ можно получить *дополнением* до $p-1$ каждой цифры числа $Z$ и последующим прибавлением к последнему разряду 1.

**Пример 4.14.**

Построить дополнение числа $278_{10}$. В данном случае $p = 10$, $k = 3$.

$$D(278_{10}, 3) = \{\langle 9-2\rangle\langle 9-7\rangle\langle 9-8\rangle\} + 1,$$

т. е. $721 + 1 = 722$.

Важным свойством дополнения является то, что его сумма с исходным числом *в заданной разрядной сетке* будет равна 0. В рассмотренном примере

$$722 + 278 = 1\ 000 = 0.$$

В разряде тысяч 1 должна быть отброшена, поскольку она выходит за отведенную разрядную сетку.

Так как в двоичной системе счисления дополнением 1 является 0, а дополнением 0 является 1, построение $D(Z_2, k)$ сводится к *инверсии* данного числа, т. е. замена нулей единицами и единиц нулями, и прибавлением 1 к последнему разряду. Другими словами, *дополнение* двоичного числа формируется в два этапа:

1) строится инвертированное представление исходного числа;

2) к инвертированному представлению прибавляется 1 по правилам двоичной арифметики.

*Дополнительный код* ($DK$) двоичных целых чисел строится по следующим правилам:

1) для $Z_2 \geqslant 0$ дополнительный код совпадает с самим числом ($DK = Z_2$);

2) для $Z_2 < 0$ дополнительный код совпадает с дополнением модуля числа, т. е. $DK = D(|Z_2|, k)$.

**Пример 4.15.**

Построить дополнительные двоичные коды чисел (a) $3_{10}$ и (b) $-3_{10}$.

a) так как $Z > 0$, DK = 0000 0000 0000 0011;

b) так как $Z < 0$, 1) модуль числа 0000 0000 0000 0011;

2) инверсия числа 1111 1111 1111 1100;

3) $DK$ 1111 1111 1111 1101.

Проверка:

$$\begin{array}{r} 11111\ 1111\ 1111\ 111 \text{ — переносы} \\ 0000\ 0000\ 0000\ 0011 \\ +\ 1111\ 1111\ 1111\ 1101 \\ \hline \boxed{1}\ 0000\ 0000\ 0000\ 0000 \end{array}$$

Вновь убеждаемся, что

$$DK(Z) + DK(-Z) = 0. \tag{4.14}$$

Таблица 4.2

| Прямой десятичный код | Прямой двоичный код с 16-ю разрядами | Дополнительный двоичный код с 16-ю разрядами |
|---|---|---|
| −32 769 | — | — |
| −32 768 | — | 1 000 000 000 000 000 |
| −32 767 | 1 111 111 111 111 110 | 1 000 000 000 000 000 |
| −32 766 | 1 111 111 111 111 110 | 1 000 000 000 000 010 |
| ... | ... | ... |
| −3 | 1 000 000 000 000 010 | 1 111 111 111 111 100 |
| −2 | 1 000 000 000 000 010 | 1 111 111 111 111 110 |
| −1 | 1 000 000 000 000 000 | 1 111 111 111 111 110 |
| 0 | 0000 0000 0000 0000 | 0000 0000 0000 0000 |
| 1 | 0000 0000 0000 0001 | 0000 0000 0000 0001 |
| 2 | 0000 0000 0000 0010 | 0000 0000 0000 0010 |
| ... | ... | ... |
| 32766 | 111 111 111 111 110 | 111 111 111 111 110 |
| 32767 | 111 111 111 111 111 | 111 111 111 111 111 |
| 32768 | — | — |

Сопоставление прямых и дополнительных кодов представлено в виде табл. 4.2.

Видно, что общее количество кодов совпадает и, следовательно, одинаковым будет количество кодируемых чисел в обоих способах. Точнее, дополнительных кодов оказывается на один больше, чем прямых, и интервал целых чисел со знаком при их размещении в 2-байтном машинном слове составляет $[-32768; 32767]$ — именно такими являются граничные значения целых чисел типа Integer в языке PASCAL, что свидетельствует об использовании дополнительного кодирования в представлении чисел. Перевод в дополнительный код происходит автоматически при вводе чисел; в таком виде числа хранятся в ОЗУ и затем участвуют в арифметических операциях. При этом, как уже было сказано, операция вычитания двух чисел как самостоятельная отсутствует — она заменяется сложением первого числа с дополнительным кодом второго, т. е. просто сложением содержимого двух ячеек памяти. Убедимся в правомочности этого.

**Пример 4.16.**

Найти значение $(27 - 3)_{10}$ в двоичной кодировке.

$$\begin{array}{ll} \text{Переносы:} & 11111\ 1111\ 1111\ 111 \\ \text{DK}(27) = & 0000\ 0000\ 0001\ 1011 \\ \text{DK}(-3) = & + \underline{1111\ 1111\ 1111\ 1101} \\ & \boxed{1}\ 0000\ 0000\ 0001\ 1000_2 = 24_{10} \end{array}$$

Таблица 4.3

| Старший бит $Z^{(1)}$ | Старший бит $Z^{(2)}$ | Старший бит $Z$ | Регистр переполнения | Комментарий |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | Сложение двух положительных чисел без переполнения. Результат корректен |
| 0 | 0 | 1 | 0 | Переполнение при сложении двух положительных чисел. **Результат некорректен** |
| 1 | 1 | 1 | 1 | Сложение двух отрицательных чисел без переполнения. Результат корректен |
| 1 | 1 | 0 | 1 | Переполнение при сложении двух отрицательных чисел. **Результат некорректен** |
| 0 | 1 | 0 | 1 | Сложение чисел с разными знаками; $Z^{(1)} > \|Z^{(2)}\|$. Результат корректен |
| 0 | 1 | 1 | 0 | Сложение чисел с разными знаками; $Z^{(1)} < \|Z^{(2)}\|$. Результат корректен |

В данном случае появление 1 в регистре переполнения не интерпретируется как ошибка вычислений, поскольку на ее отсутствие указывают знаки чисел и результата. Порядок проверок и анализа корректности операций сложения-вычитания ($Z = Z^{(1)} + Z^{(2)}$) можно представить в виде табл. 4.3.

Необходимо уточнить, что при выполнении вычитания отрицательного числа строится дополнительный код его дополнительного кода и вновь вместо вычитания выполняется сложение.

Подобным же образом число из дополнительного кода переводится в прямой при выполнении операции умножения; перемножаются всегда положительные числа по рассмотренным выше правилам; знаковый бит результата, очевидно, будет содержать 0, если знаки чисел одинаковы, и 1 при противоположных знаках.

Над множеством целых чисел со знаком операция деления не определена, поскольку в общем случае ее результатом будет вещественное число. Однако допустимыми являются операции целочисленного деления и нахождения остатка от целочисленного деления (те, что мы немного ранее обозначили div и mod). Точнее, значения обеих величин находятся одновременно в одной процедуре, которая, в конечном счете, сводится к последователь-

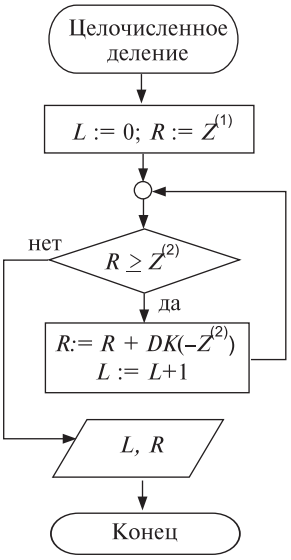



**Рис. 4.7.** Алгоритм нахождения значений $L$ и $R$

ности вычитаний или, еще точнее, сложений с дополнительным кодом делителя.

Примем обозначения: $Z^{(1)}$ — делимое; $Z^{(2)}$ — делитель; $L$ — результат целочисленного деления $Z^{(1)}$ на $Z^{(2)}$; $R$ — остаток от целочисленного деления $Z^{(1)}$ на $Z^{(2)}$. Эти величины связаны между собой очевидным соотношением

$$Z^{(1)} = LZ^{(2)} + R,$$

из которого следует алгоритм нахождения значений $L$ и $R$ для заданных $Z^{(1)}$ и $Z^{(2)}$; его блок-схема для положительных $Z^{(1)}$ на $Z^{(2)}$ представлена на рис. 4.7.

Таким образом, операции div и mod, как, впрочем, и операция умножения, реализуются программно, т. е. сводятся к последовательности небольшого числа более простых действий. При этом уровень программной реализации может быть различным. Если реализация выполнена на уровне команд центрального процессора, то эти операции оказываются доступны из любого приложения (любой прикладной программы). Если же в системе команд процессора эти микропрограммы отсутствуют, их приходится описывать в виде процедур в самих приложениях и, следовательно, они будут доступны только в этих приложениях.

### 4.3.3. Кодирование и обработка в компьютере вещественных чисел

Вернемся к обсуждению того обстоятельства, что в компьютере для записи числа в любой форме представления отводится конечное число разрядов. Для целых чисел это обстоятельство привело к появлению понятий наибольшего целого числа. Однако для каждого целого числа, не превышающего по модулю наибольшего, имеется ровно одно представление в машинном коде и, если не происходит переполнения, результат выполнения операции над целыми числами будет *точным*, поскольку дискретные множества исходных чисел однозначно отображаются на дискретное множество результатов.

Ситуация радикальным образом меняется при представлении и обработке вещественных чисел. На математической числовой

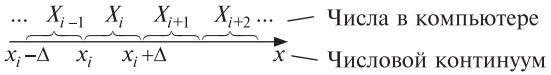



**Рис. 4.8.** Представление чисел

оси вещественные числа образуют непрерывное множество (*континуум*), т. е. два числа могут находиться сколь угодно близко друг к другу, и на любом отрезке содержится бесконечно много значений чисел. В машинном представлении количество возможных значений чисел *конечно*; для двоичной системы счисления оно определяется как $2^k$, где $k$ — количество двоичных разрядов в представлении мантиссы. То есть вещественные числа в компьютере заменяются их *кодами*, которые образуют *конечное дискретное множество*; каждый код оказывается *представителем* целого интервала значений континуума (рис. 4.8).

Из данного обстоятельства вытекают ряд следствий.

**Следствие 1** состоит в том, что строгие отношения между числами континуума превращаются в нестрогие для их компьютерных представителей, т. е.

если $x^{(1)} > x^{(2)}$, то $X^{(1)} \geqslant X^{(2)}$;
если $x^{(1)} = x^{(2)}$, то $X^{(1)} = X^{(2)}$;
если $x^{(1)} < x^{(2)}$, то $X^{(1)} \leqslant X^{(2)}$.

**Следствие 2.** Поскольку код вещественного числа в компьютере является *приблизительным* представителем многих чисел из интервала, то и результаты вычислений также будут заведомо неточными, содержащими неизбежную погрешность. В этом состоит *главная особенность обработки вещественных чисел в компьютере — она всегда ведется с погрешностью* (кстати, оценка этой погрешности — самостоятельная и непростая задача). В случае цепочечных вычислений эта погрешность будет подниматься по десятичным разрядам числа, и, естественно, понижать точность вычислений.

**Следствие 3.** Наряду с понятием наибольшего вещественного числа (из-за ограниченности разрядной сетки) появляется понятие наименьшего числа или *машинного нуля*. Например, в типе Real языка PASCAL любое десятичное число, по модулю меньшее $2{,}3 \cdot 10^{-39}$, оказывается машинным нулем, т. е. считается равным 0 при сохранении и в операциях с ним. Таким образом, математическое понятие «0» как точное число в компьютерном представлении заменяется понятием «машинный нуль» как *число, меньшее некоторой определенной величины*.


**Рис. 4.9.** Размещение числа в ячейках

Как уже было сказано, основной формой представления кодов вещественных чисел в компьютере является двоичная нормализованная. При этом записываться и храниться в памяти компьютера должны все составляющие нормализованной формы (знак числа, мантисса, знак порядка и порядок), что требует нескольких ячеек памяти. Например, числа типа Real («вещественный») из языка PASCAL размещаются в 6 байтах, т. е. 48 двоичных разрядах. Непосредственное распределение компонентов нормализованного числа по разрядам определяется конструктивными особенностями компьютера и программным обеспечением. На рис. 4.9 приведен пример размещения числа в двух ячейках памяти (32 разряда).

Поскольку значение мантиссы лежит в интервале $0{,}1_2 \leqslant \leqslant M_2 < 1$, ноль в разряде целых и разделитель десятичных разрядов в представление не включаются, т. е. мантисса содержит только цифры дробной части. Более того, можно не сохранять и первую значащую цифру мантиссы, поскольку она всегда 1 (но, естественно, восстанавливать ее при вычислениях) — это дает возможность хранить дополнительный «скрытый» разряд, т. е. несколько повысить точность обработки. В ходе выполнения арифметических операций, как указывалось ранее, промежуточные и конечный значения нормализуются, состоящая в сдвиге мантиссы вправо или влево с одновременным изменением порядка, что эквивалентно смещению разделителя десятичных разрядов — именно по этой причине такая форма представления числа получила название «*с плавающей запятой*». Как и в случае целых чисел, для кодов вещественных чисел существует понятие *переполнение*, однако возникает оно не после заполнения разрядной сетки мантиссы — это приводит лишь к нормализации числа, а при заполнении всех разрядов порядка. Для представленного выше примера размещения числа в 32-х битах, очевидно,

$$|X_2|^{\max} = 0{,}11111111111111111111111_2 \cdot 2^{11111} \approx 2{,}147_{10} \cdot 10^9.$$

При этом точность обработки составит 7 десятичных разрядов.

При $|X_2| > |X_2|^{\max}$ возникнет переполнение, т. е. операция станет некорректной.

**Пример 4.17.**

Установить распределение разрядов двоичного представления числа типа Real, если для его записи отводится 48 бит, а максимальное значение десятичного порядка 38. Какова точность обработки таких чисел?

1) 2 бита расходуется на запись знака числа и порядка;

2) согласно (4.9) $k_2 = 3{,}322k_{10}$; поскольку $k_{10} = 38$, очевидно, максимальный показатель порядка двоичного числа $k_2 = 3{,}322 \cdot 38 \approx 126_{10}$, что требует в двоичном представлении согласно формуле Хартли 7 бит;

3) под запись мантиссы отводится $48 - 2 - 7 = 39$ бит;

4) с учетом скрытого разряда точность обработки составит $\dfrac{39+1}{3{,}322} \approx$ $\approx$ 12 десятичных разрядов.

Изначальной причиной возникновения погрешности обработки кодов вещественных чисел является ограниченность разрядной сетки при их представлении и, следовательно, наличие погрешности неизбежно. Однако ее величина зависит от количества имеющихся разрядов и, в частности, уменьшить погрешность можно за счет расширения разрядной сетки, т. е. выделения большего количества ячеек памяти для записи числа. Например, в языке PASCAL определен вещественный тип Extended, числа которого занимают 10 байт, что обеспечивает точность мантиссы до 20 десятичных знаков и значение модуля порядка до 4932. Несколько вариантов представления вещественных чисел в языках программирования высокого уровня используется как одно из средств оптимизации программы. Повышение точности вычислений требует больших ресурсов памяти компьютера; одновременно с этим возрастает и время вычислений. Таким образом, при составлении программы для практической задачи решается проблема нахождения компромисса между точностью результата и временем обработки.

В процессе выполнения арифметических действий с нормализованными числами отдельно обрабатываются мантиссы и порядки. Поскольку операции над кодами вещественных чисел в компьютере обладают некоторой спецификой по сравнению с обычными арифметическими, будем обозначать их следующим образом: $\oplus$ — сложение (вычитание), $\otimes$ — умножение, $\varnothing$ — деление.

**Сложение нормализованных чисел**

Пусть имеются два числа $X_2 = M_2 p^{k_2}$ и $X_1 = M_1 p^{k_1}$ (здесь индексы у мантиссы и порядка означают не систему счисления,

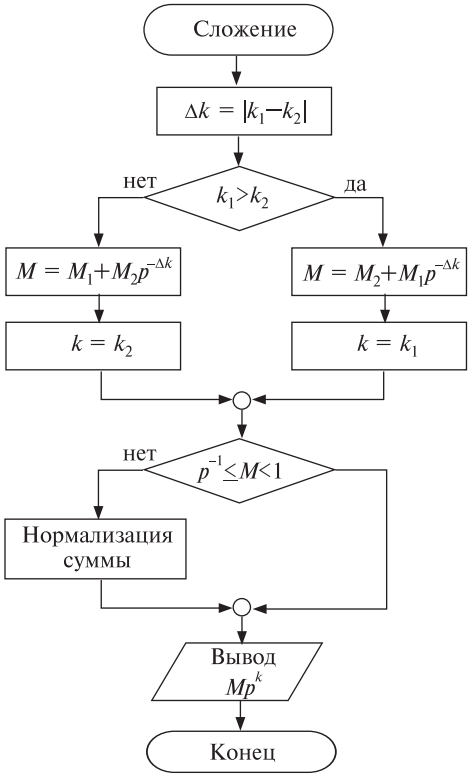



**Рис. 4.10.** Алгоритм сложения нормализованных чисел

а служат номерами чисел). Сложение должно начинаться с выявления большего из $k_1$ и $k_2$, нахождения модуля их разности $\Delta k = |k_1 - k_2|$ и сдвига *вправо* на $\Delta k$ разрядов мантиссы того числа, у которого $k$ оказался меньше. После этого выполняется сложение мантисс, порядку результата присваивается значение большего из имеющихся и при необходимости результат нормализуются. Алгоритм сложения нормализованных чисел представлен в виде блок-схемы на рис. 4.10. При сдвиге вправо мантиссы меньшего числа происходит потеря $\Delta k$ младших значащих цифр, что приводит к появлению погрешности сложения.

Рассмотрим действие алгоритма на примере сложения десятичных чисел в ограниченной разрядной сетке.

**Пример 4.18.**

Найти сумму $X_1 = 0{,}87654 \cdot 10^1$ и $X_2 = 0{,}94567 \cdot 10^2$, если для записи мантиссы отводится 5 разрядов.

Согласно алгоритму $\Delta k = 1$ и $k_1 < k_2$. Следовательно, $k = k_2 = 2$, а мантисса числа $X_1$ должна быть сдвинута на 1 разряд вправо (при этом из-за ограниченности разрядно сетки пропадет цифра 4). Новая мантисса получается суммированием $M = 0{,}94567 + 0{,}08765 = 1{,}03332$; поскольку она выходит за допустимый интервал представления мантисс, необходимо его нормализовать: $M' = 0{,}10333$ (при этом теряется цифра 2 в младшем разряде); $k' = k + 1 = 3$. Окончательно получаем: $X = 0{,}10333 \cdot 10^3$. Точный результат суммирования оказался бы 103,3324, т. е. последние два десятичных разряда оказываются потерянными.

Следствием существования погрешности сложения (и, в равной мере, вычитания) кодов вещественных чисел оказывается то, что такое суммирование не обладает ассоциативностью, т. е. в общем случае

$$(X_1 \oplus X_2) \oplus X_3 \neq X_1 \oplus (X_2 \oplus X_3).$$

**Вычитание нормализованных чисел**, как и чисел целых, не является самостоятельной операцией и сводится к сложению с дополнительным кодом числа.

**Умножение нормализованных чисел** $X_1 \otimes X_2$ выполняется в соответствии с правилами: если по-прежнему $X_2 = M_2 p^{k_2}$ и $X_1 = M_1 p^{k_1}$, то, очевидно, мантисса произведения $M = M_1 M_2$, а порядок $k = k_1 + k_2$; при необходимости полученное число нормализуется.

**Операция деления**, проводимая как над целыми, так и вещественными числами, приводит в общем случае к появлению вещественного числа, поэтому целые числа предварительно преобразуются в вещественный тип, т. е. переводятся в нормализованную форму. Очевидно, при делении $X_1 \oslash X_2$ мантисса частного $M = M_1/M_2$, а порядок $k = k_1 - k_2$. При этом непосредственно операция деления сводится к сдвигу делителя вправо и последовательному вычитанию его из делителя (т. е. сложения с дополнительным кодом вычитаемого). Как и в предыдущих операциях, результат деления при необходимости нормализуется.

В операциях умножения нормализованных чисел в компьютере возможны ситуации, когда не будут в точности выполняться сочетательный и распределительный законы, т. е.

$$(X_1 \otimes X_2) \otimes X_3 \neq X_1 \otimes (X_2 \otimes X_3);$$
$$(X_1 \oplus X_2) \otimes X_3 \neq X_1 \otimes X_3 \oplus X_2 \otimes X_3.$$

Заканчивая рассмотрение порядка обработки чисел в компьютере, хотелось бы сделать ряд общих замечаний:

1. В компьютерах арифметические устройства выполняют действия не с самими двоичными числами по правилам двоичной арифметики, а с их двоичными кодами (представлениями) по правилам арифметики двоичных кодов.

2. Причиной отличий правил арифметики двоичных кодов от правил обычной арифметики является ограниченность разрядной сетки, применяемой для записи чисел в компьютере. По этой же причине отличаются понятия «*ноль*» и «*машинный ноль*», «*бесконечность*» — «*максимальное число*», а также становится возможной ситуация переполнения, что требует ее постоянного отслеживания.

3. Применение при вычислениях формы представления чисел с плавающей запятой обеспечивает единообразие при их записи и обработке, и, что важно, в результате автоматического масштабирования числа на каждом этапе его обработки сокращается погрешность вычислений.

4. Различие правил обработки целых и нормализованных чисел приводит к необходимости точного описания типов переменных перед их использованием в программах. Вторая причина описания типов состоит в оптимизации расходования памяти компьютера, поскольку числа разных типов требуют для хранения различных ресурсов памяти.

## Контрольные вопросы и задания к гл. 4

1. Почему унарную систему счисления нельзя считать позиционной с основанием 1?

2. Составьте блок-схему алгоритма преобразования $Z_p \to Z_1 \to Z_q$ (п. 4.2.1) и реализуйте его для произвольных $p$ и $q$ на каком-либо языке программирования.

3. Составьте блок-схему алгоритма преобразования $Z_{10} \to Z_q$ (п. 4.2.1, способ 2) и реализуйте его для произвольного $q$ на каком-либо языке программирования.

4. В MS Escel разработайте и реализуйте алгоритм перевода $Z_{10} \to Z_p$.

5. В MS Escel разработайте и реализуйте алгоритм перевода $Z_p \to Z_{10}$.

6. В MS Escel разработайте и реализуйте алгоритм перевода $0{,}Y_{10} \to$ $\to 0{,}Y_p$ с заданной точностью.

7. В MS Escel разработайте и реализуйте алгоритм перевода $0{,}Y_p \to$ $\to 0{,}Y_{10}$ с заданной точностью.

8. На языке программирования разработайте программу, осуществляющую перевод вещественных чисел в естественной форме (с фиксированной запятой) между любыми системами счисления.

9. Исследуйте экономичность различных систем счисления для общего количества цифр 24, 60, 120. Сделайте вывод.

10. Постройте график функции (4.10) в MS Excel или каком-либо математическом пакете и определите положение максимума ($p_m$).

11. Получите выражение (4.11) аналитическим путем.

12. Произведите преобразования $Z_p \to Z_q$ без перехода к промежуточным системам счисления: $120112_3 \to (\ldots)_9$; $\mathrm{BAC}_{16} \to (\ldots)_4$; $12345678_9 \to (\ldots)_3$.

13. Исследуйте, как осуществляется перевод $0{,}Y_p \to 0{,}Y_q$, если одно основание системы счисления является целой степенью другого основания.

14. Приведите примеры перевода дробного числа из одной системы счисления в другую, при котором конечная дробь становится бесконечной и наоборот.

15. Для вещественного числа в естественной форме представления постройте алгоритм перевода $X_{10} \to X_2$ при $k_p < 0$. Реализуйте его программным путем одновременно с алгоритмом для $k_p \geqslant 0$.

16. Для вещественного числа в естественной форме представления постройте алгоритм перевода $X_2 \to X_{10}$ при $k_p < 0$. Реализуйте его программным путем одновременно с алгоритмом для $k_p \geqslant 0$.

17. Реализуйте программным путем изображенный на рис. 4.2 алгоритм нормализации числа в произвольной системе счисления.

18. Осуществите перевод $123{,}456_{10} \cdot 10^3$ в восьмиричную систему в нормализованной форме с шестью значащими цифрами.

19. Осуществите перевод $123{,}456_8 \cdot 8^3$ в десятичную систему в нормализованной форме с шестью значащими цифрами.

20. Исследуйте, как осуществляется преобразование чисел в нормализованной форме представления из одной системы счисления в другую, если одно основание системы счисления является целой степенью другого основания.

21. Предложите алгоритм умножения двух целых двоичных чисел, предусматривающий чередование операций сдвига второго сомножителя и промежуточного накапливающего суммирования. Реализуйте алгоритм на языке программирования.

22. Подтвердите справедливость таблицы 4.2 конкретными примерами.

23. Постройте обобщенный вариант блок-схемы на рис. 4.9, учитывающий, что любое из чисел $Z_1$ и $Z_2$ или они оба могут быть отрицательными. Реализуйте алгоритм на языке программирования.

24. Реализуйте программным путем алгоритм, представленный на рис. 4.10. Будет ли он справедлив для операции вычитания?

25. С двумя десятичными числами: 37,5 и 3 произведите следующие операции при условиях: нормализованная форма представления, двоичная система счисления, 16-разрядная сетка для представления двоичных чисел:

a) 37,5 + 3; b) 37,5 × 3; c) 37,5 − 3.

26. Приведите пример того, что сложение кодов вещественных чисел в нормализованной форме с ограниченной разрядной сеткой не обладает ассоциативностью.

27. Покажите на примерах, что при операциях с кодами вещественных чисел в нормализованной форме при ограниченной разрядной сетке в общем случае не выполняются сочетательный и распределительный законы.

# 5 Передача информации

Использование информации для решения каких-либо задач, безусловно, сопряжено с необходимостью ее распространения, т. е. осуществления процессов передачи и приема. При этом как указывалось выше, приходится решать проблему согласования метода кодирования с характеристиками канала связи, а также обеспечить защиту передаваемой информации от возможных искажений. Рассмотрению этих вопросов и будет посвящена ближайшие главы.

## 5.1. Общая схема передачи информации по линии связи

Ранее источник информации был определен как объект или субъект, порождающий информацию и представляющий ее в виде сообщения, т. е. последовательности сигналов. При этом человек в информационном взаимодействии с окружающей средой ограничен возможностями собственных органов чувств. Однако спектр процессов, на основе которых передается информация, может быть расширен за счет использования *средств связи*:

> ***Средства связи*** *— совокупность устройств, обеспечивающих преобразование первичного сообщения от источника информации в сигналы заданной физической природы, их передачу, прием и представление в форме удобной потребителю.*

Средств связи существует множество: почта, телефон, радио, телевидение, компьютерные сети и пр. Однако при всем разнообразии их конкретной реализации можно выделить общие элементы, представленные на рис. 5.1.

Источник информации (ИИ) выдает ее в виде первичного сообщения, представленного последовательностью первичных сигналов. Для дальнейшей передачи эти сигналы преобразуются в

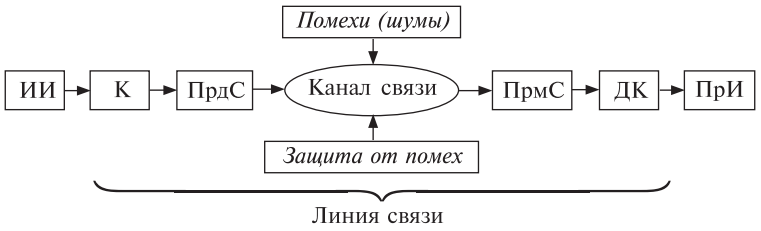



**Рис. 5.1.** Общая схема передачи информации: ИИ — источник информации; К — кодер; ПрдС — передатчик сообщения; ПрмС — приемник сообщения; ДК — декодер; ПрИ — приемник информации

сигналы такой физической природы, которые могут распространяться в заданном материальном носителе — формируется вторичное сообщение. Примерами преобразователей являются: мегафон или телефонный аппарат, преобразующие голосовые сигналы в электрические; радиопередатчик, преобразующие голосовые сигналы в радиоволны; телекамера, преобразующая изображение в последовательность электрических импульсов; модем, переводящий высокочастотные компьютерные сигналы в аналоговые низкочастотные и обратно, и пр. В общем случае при преобразовании выходные сигналы не полностью воспроизводят все особенности первичного сообщения, а лишь его существенные стороны, т. е. при преобразовании часть информации теряется. Например, полоса пропускания частот при телефонной связи от 300 до 3400 Гц, в то время как частоты, воспринимаемые человеческим ухом, лежат в интервале 16...20000 Гц (т. е. телефонные линии «обрезают» высокие частоты, что приводит к искажениям звука); в черно-белом телевидении при преобразовании теряется цвет изображения. Именно в связи с возможными потерями встает задача выработки таких способов представления и последующего преобразования первичного сообщения, которые обеспечивали бы возможно более полную сохранность исходной информации и, одновременно, согласование со скоростью передачи информации по данной линии связи.

При необходимости перед преобразованием или в процессе его может осуществляться кодирование первичного сообщения кодером (К). Кодирование (точнее, первичное кодирование) может осуществляться непосредственно источником информации, например человеком при работе на передатчике с использованием азбуки Морзе. Возможно совмещение кодера с преобразователем, например при работе человека за клавиатурой телеграфного

Таблица 5.1

| Канал связи | Среда | Носитель сообщения | Процесс, используемый для передачи сообщения |
|---|---|---|---|
| Почта, курьеры | Среда обитания человека | Бумага | Механическое перемещение носителя |
| Телефон, компьютерные сети | Проводник | Электрический ток | Перемещение электрических зарядов |
| Радио, телевидение | Электромагнитное поле | Электромагнитные волны | Распространение электромагнитных волн |
| Зрение | Электромагнитное поле | Световые волны | Распространение световых волн |
| Слух | Воздух | Звуковые волны | Распространение звуковых волн |
| Обоняние, вкус | Воздух, пища | Химические вещества | Химические реакции |
| Осязание | Поверхность кожи | Объект, воздействующий на органы осязания | Теплопередача, давление |

аппарата или компьютера он вводит знаки естественного языка, а уже устройством они переводятся в коды, которые затем передаются.

Непосредственная передача осуществляется передатчиком вторичного сообщения (ПрдС). Он инициирует некоторый нестационарный процесс, обеспечивающий распространение сигналов в *канале связи*.

> ***Канал связи*** *— материальная среда, а также физический или иной процесс, посредством которого осуществляется передача сообщения, т. е. распространение сигналов в пространстве с течением времени.*

Каналы связи в зависимости от характера сигналов, передаваемых по ним подразделяются на *дискретные* и *аналоговые*. Примером дискретного канала является компьютерная сеть; аналогового — телефонная линия и радиоканал.

В табл. 5.1 приведены примеры некоторых каналов связи.

Любой *реальный* канал связи подвержен внешним воздействиям, а также в нем могут происходить внутренние процессы, в

результате которых искажаются передаваемые сигналы и, следовательно, теряется связанная с ними информация. Такие воздействия называются *шумами* (помехами). Источники помех могут быть *внешними*, например так называемые *наводки* от мощных потребителей электричества или атмосферных явлений, приводящие к появлению нарушений в радиосвязи; одновременное действие нескольких близкорасположенных однотипных источников (одновременный разговор нескольких человек). К помехам могут приводить и *внутренние* особенности данного канала, например физические неоднородности носителя, паразитные явления в шинах компьютера, процессы затухания сигнала в линии связи из-за большой удаленности. Если уровень помех оказывается соизмерим с интенсивностью несущего сигнала, то передача информации по данному каналу оказывается невозможной. Однако и при относительно низких уровнях шумов они могут вызывать искажения передаваемых сигналов и, следовательно, частичную потерю связанной с ними информации. Существуют и применяются методы защиты от помех, например экранирование электрических линий связей, улучшение избирательности приемного устройства и т. д. Другим способом защиты от помех является использование специальных методов кодирования информации, о чем речь пойдет ниже.

После прохождения сообщения по каналу связи оно попадает в приемное устройство (ПрмС), где преобразуется в форму, необходимую для дальнейшей интерпретации. Если перед передачей применялось кодирование, после приема сообщение направляется в декодер (ДК) и лишь затем — к получателю (потребителю) информации (ПрИ). При этом декодер может быть совмещен с преобразователем (например, телеграфный аппарат или компьютер) или с приемником информации (радист, принимающий сигналы азбуки Морзе и интерпретирующий их).

Понятие *линия связи* охватывает все элементы представленной на рис. 5.1 схемы от источника до приемника информации, т. е.

> ***Линия связи** — это совокупность средств связи и канала связи, посредством которых осуществляется передача инфорамации от источника к приемнику.*

Характеристиками любой линии связи являются скорость, с которой возможна передача сообщения в ней, а также степень искажения сообщения в процессе передачи. Из этих параметров вычленим те, что относятся непосредственно к каналу связи, т. е.

характеризуют среду и процесс передачи. При этом мы затронем только вопросы передачи по дискретному каналу связи.

## 5.2. Характеристики дискретного канала связи

Введем довольно очевидное определение:

***Дискретный канал** — канал связи, используемый для передачи дискретных сообщений.*

Следует еще раз обратить внимание, что дискретность — это не свойство канала связи, а свойство передаваемых по нему сигналов.


**Рис. 5.2.** Канал как «черный ящик»

Каналы для передачи информации могут иметь самую разнообразную физическую природу. Однако нас не будут интересовать механизмы, приводящие к возможным изменениям сигналов при их передаче по каналу — это позволяет рассматривать канал как некий «черный ящик», на вход которого подается последовательность сигналов и какая-то последовательность получается на выходе (рис. 5.2).

Источник дискретных сообщений (ИДС) использует для представления информации первичный алфавит $\{A\}$. Первичный кодер (ПК) кодирует знаки первичного алфавита $n$ элементарными сигналами с алфавитом $\{a\}$. Действие помех в процессе передачи может состоять в том, что алфавит принимаемых сигналов будет отличаться от алфавита входных сигналов как их числом так и характеристиками — пусть это будет алфавит $\{b\}$, содержащий $m$ элементарных сигналов. Несовпадение алфавитов сигналов приводит к тому, что на выходе канала появляются такие комбинации элементарных сигналов, которые не могут быть интерпретированы как коды знаков первичного алфавита.

Вводя количественные характеристики процесса передачи информации, постараемся выделить из них те, которые зависят только от свойств канала, и те, которые определяются особенностями источника дискретного сообщения.

Дискретный канал считается заданным, если известны:

- время передачи одного элементарного сигнала $\tau$;
- исходный алфавит элементарных сигналов $\{a\}$, т. е. все его знаки $a_i$, $i = 1 \ldots n$, где $n$ — число знаков алфавита $\{a\}$;

- $n$ значений вероятностей появления элементарных сигналов на входе $p(a_i)$; эти вероятности называются *априорными* (поскольку они определяются не свойствами канала, а источником сообщения, т. е. являются внешними по отношению к каналу и самому факту передачи сообщения);
- алфавит сигналов на выходе канала $\{b\}$, т. е. все знаки $b_j$, $j = 1, \ldots, m$, где $m$ — число знаков алфавита $\{b\}$; в общем случае $n \neq m$;
- значения условных вероятностей $p_{a_i}(b_j)$, каждая из которых характеризует вероятность появления на выходе канала сигнала $b_j$ при условии, что на вход был послан сигнал $a_i$; поскольку эти вероятности определяются свойствами самого канала передачи, они называются *апостериорными*; очевидно, количество таких вероятностей равно $nm$:

$$p_{a_1}(b_1), p_{a_1}(b_2), ..., p_{a_1}(b_m);$$
$$p_{a_2}(b_1), p_{a_2}(b_2), ..., p_{a_2}(b_m);$$
$$...$$
$$p_{a_n}(b_1), p_{a_n}(b_2), ..., p_{a_n}(b_m).$$

Очевидно также, что для каждой строки выполняется условие нормировки

$$\sum_{j=1}^{m} p_{a_i}(b_j) = 1, \quad i = 1, \ldots, n.$$

Как мы увидим в дальнейшем, все остальные характеристики дискретного канала могут быть определены через перечисленные параметры.

> *Дискретный канал называется* ***однородным,*** *если для любой пары $i$ и $j$ условная вероятность $p_{a_i}(b_j)$ с течением времени не изменяется (т. е. влияние помех все время одинаково).*

> *Дискретный канал называется* ***каналом без памяти,*** *если $p(a_i)$ и $p_{a_i}(b_j)$ не зависят от места знака в первичном сообщении (т. е. отсутствуют корреляции знаков).*

> *Если все переданные сигналы прошли канал без искажений, то такой канал называется* ***идеальным*** *(без шума).*

*Если все переданные сигналы на приемном конце были заменены противоположными, то канал называется идеальным* ***инвертирующим.***

Будем считать, что для передачи используются колебательные или волновые процессы — с практической точки зрения такие каналы представляют наибольший интерес (в частности, к ним относятся компьютерные линии связи).

Введем ряд величин, характеризующих передачу информации по каналу.

**Длительность элементарного импульса.** Мы будем считать, что передача информации в канал осуществляется элементарными импульсами, совокупность которых и образует алфавит сигналов на входе $\{a\}$. При двоичном кодировании, очевидно, таких сигналов два — 0 (пауза) и 1 (импульс). Будем считать также, что их длительность одинакова и составляет $\tau$*. Из этого, в свою очередь, следует, что передача ведется на фиксированной частоте (она называется также несущей)

$$\nu_m = \frac{1}{\tau},$$

и означает в свою очередь, что за 1 с в канал могут быть посланы $\nu_m$ сигналов.

Если код знака первичного алфавита состоит из $k_i$ элементарных сигналов, время его передачи по каналу составит $t_i = k_i\tau$, а среднее время передачи кодовой комбинации одного знака первичного алфавита $t = K(A, a)\tau$.

**Пропускная способность канала связи.** С передачей одного элементарного сигнала связано некоторое количество информации $I_s$. Если общее число различных элементарных сигналов $n$, а вероятности их появления $p(a_i)$, $i = 1, \ldots, n$, то согласно формуле Шеннона

$$I_s = -\sum_{i=1}^{n} p(a_i) \log_2 p(a_i).$$

Как мы неоднократно убеждались ранее, данная величина зависит от распределения вероятностей знаков; своего максимального

---

* На самом деле при передаче чаще используется частотная модуляция, при которой длительность сигналов различна, но эти технические детали не нарушают хода наших построений.

значения она достигает при условии, что появление всех элементарных сигналов (знаков вторичного алфавита) будет равновероятным — в таком случае

$$I_s = I_s^{\max} = \log_2 n.$$

Это значение является *предельным* (наибольшим) для информационного содержания элементарного сигнала выбранного вторичного алфавита. Поскольку такое количество информации передается за время $\tau$, можно ввести величину, характеризующую предельную интенсивность информационного потока через канал, — *пропускную способность канала*

$$C = \frac{I_s^{\max}}{\tau} = \nu_m I_s^{\max}. \tag{5.1}$$

Данная величина является характеристикой канала связи, поскольку зависит только от его особенностей. Это выражение служит определением пропускной способности как идеального канала (без помех), так и реального канала с помехами — просто, как мы увидим далее, информационное содержание элементарного сигнала в реальном канале оказывается меньше $\log_2 n$.

Если $I_s^{\max}$ выражено в битах, а $\tau$ — в секундах, то единицей измерения $C$ будет бит/с. Раньше эта единица называлась *бод*, однако название не прижилось, и по этой причине пропускная способность канала связи измеряется в бит/с. Производными единицами являются:

1 Кбит/с $= 10^3$ бит/с;
1 Мбит/с $= 10^6$ бит/с;
1 Гбит/с $= 10^9$ бит/с.

При отсутствии в канале связи помех $I_s^{\max} = \log_2 n$; тогда

$$C_0 = \frac{\log_2 n}{\tau} = \nu_m \log_2 n \tag{5.2}$$

— максимально возможное значение пропускной способности (это обстоятельство отражено индексом «0»); в реальном канале $I_s^{\max} \leqslant \log_2 n$ и, следовательно, $C \leqslant C_0$.

**Скорость передачи информации.** Если источник выдает $\nu_m$ элементарных сигналов в единицу времени, а средняя длина кода одного знака составляет $K(A, a)$, то, очевидно, отношение $\nu_m / K(A, a)$ будет выражать число знаков первичного алфавита, выдаваемых источником за единицу времени. Если с каждым из них связано среднее количество информации $I^{(A)}$, то можно найти общее количество информации, передаваемой источником за

единицу времени — эта величина называется *скоростью передачи* или *энтропией источника* (будем обозначать ее $J$):

$$J = \frac{\nu_m}{K(A,a)} I^{(A)} = \frac{I^{(A)}}{\tau K(A,a)}. \tag{5.3}$$

Энтропия источника, в отличие от пропускной способности, является характеристикой источника, а не канала связи.

Размерностью $J$, как и $C$, является бит/с. Каково соотношение этих характеристик? Рассмотрим канал без помех. Тогда выразив $\nu_m$ из (5.2) и подставив в (5.2), получим

$$J = \frac{I^{(A)} C_0}{K(A,a)\log_2 n}.$$

Согласно первой теореме Шеннона при любом способе кодирования

$$K(A,a) \geqslant \frac{I^{(A)}}{\log_2 n},$$

хотя может быть сколь угодно близкой к этому значению. Следовательно, всегда $J \leqslant C_0$, т. е.

**Скорость передачи информации по каналу связи не может превысить его пропускной способности*.**

Как показано в теории Шеннона, данное утверждение справедливо как при отсутствии в канале помех (шумов) (идеальный канал связи), так и при их наличии (реальный канал связи).

**Пример 5.1.**

Первичный алфавит состоит из трех знаков с вероятностями $p_1 = 0{,}2$; $p_2 = 0{,}7$; $p_3 = 0{,}1$. Для передачи по каналу без помех используются равномерный двоичный код. Передача ведется на частоте 500 Гц. Каковы пропускная способность канала и скорость передачи?

Поскольку код двоичный, $n = 2$, а из (5.2) $C_0 = 500$ бит/с.

Число знаков первичного алфавита $N = 3$. Из (2.14)

$$I^{(A)} = I_1 = -0{,}2\log_2 0{,}2 - 0{,}7\log_2 0{,}7 - 0{,}1\log_2 0{,}1 = 1{,}16 \text{ бит},$$

$K(A,2) \geqslant \log_2 N = 2$. Следовательно, из (5.3) получаем

$$J = \frac{I_1^{(A)}}{K(A,2)\tau} = \frac{\nu I_1^{(A)}}{K(A,2)} = \frac{500 \cdot 1{,}16}{2} = 290 \text{ бит/с}.$$

---

* На самом деле в этом результате нет ничего неожиданного: представьте трубу, через которую за единицу времени может проходить объем воды $C$, и источник, выдающий $J$ воды за это же время; ясно, что если трубу присоединить к источнику, то вода не будет теряться только в том случае, если $C \geqslant J$.

## 5.3. Влияние шумов на пропускную способность дискретного канала связи

### 5.3.1. Математическая постановка задачи

Используем для анализа процесса передачи информации по дискретному каналу с помехами энтропийный подход.

Выражение (2.13) для взаимной информации в приложении к решаемой задачи можно интерпретировать следующим образом.

Если считать опытом $\alpha$ отправку сигнала, а $\beta$ — получение его из канала, то величина $I(\alpha, \beta)$ выражает собой информацию, которая содержится в сигнале $\alpha$ относительно полученного сигнала $\beta$:

$$I(\alpha, \beta) = H(\beta) - H_\alpha(\beta).$$

Следует сделать несколько замечаний.

1. Ситуация $H_\alpha(\beta) = 0$ (выходные сигналы совпадают с входными) соответствует идеальному каналу без помех.

2. Если $\alpha$ и $\beta$ независимы, т. е. отсутствует связь между сигналами на входе и выходе канала, то $H_\alpha(\beta) = H(\beta)$ и информация не передается.

3. Потери информации при передаче характеризуются апостериорными вероятностями, входящими в выражение для $H_\alpha(\beta)$.

4. Ранее было доказано (см. задание 11 к гл. 2), что $I(\alpha, \beta) = I(\beta, \alpha)$, т. е. $\alpha$ содержит относительно $\beta$ то же количество информации, что и $\beta$ относительно $\alpha$. Следовательно,

$$I(\beta, \alpha) = H(\alpha) - H_\beta(\alpha). \tag{5.4}$$

Сущность задачи о передаче информации по реальному каналу состоит в том, чтобы построить заключение о том, какой сигнал $\alpha$ был отослан в случае приема $\beta$.

Пусть опыт $\beta$ состоит в выяснении того, какой сигнал был получен на приемном конце канала; исходами этого опыта являются сигналы $b_j$, общее число которых равно $m$. Опыт $\alpha$ состоит в выяснении того, какой сигнал был послан на вход канала; исходами этого опыта являются сигналы, образующие алфавит $a_i$; их общее количество равно $n$.

Смысл выражения (5.4) в применении к рассматриваемой ситуации состоит в том, что распознанный на приемном конце сигнал содержит информацию о сигнале, который был отправлен, но в общем случае информацию не полную. Влияние помех в

канале таково, что в процессе передачи часть начальной информации теряется и исход опыта $\beta$ не несет полной информации относительно предшествующего исхода опыта $\alpha$.

Энтропия, связанная с определением того, какой сигнал передан, согласно формуле (2.4)

$$H(\alpha) = -\sum_{i=1}^{n} p(a_i) \log_2 p(a_i).$$

$H_\beta(\alpha)$ — условная энтропия (энтропия опыта $\alpha$ при условии, что реализовался опыт $\beta$). В нашем случае согласно (2.8) и (2.9):

$$H_{b_j}(\alpha) = -\sum_{i=1}^{n} p_{b_j}(a_i) \log_2 p_{b_j}(a_i);$$

$$H_\alpha(\beta) = \sum_{j=1}^{m} p(b_j) H_{b_j}(\alpha) = -\sum_{j=1}^{m} p(b_j) \sum_{i=1}^{n} p_{b_j}(a_i) \log_2 p_{b_j}(a_i).$$

Окончательно для средней информации на один элементарный сигнал имеем

$$I(\beta, \alpha) = -\sum_{i=1}^{n} p(a_i) \log_2 p(a_i) + \sum_{j=1}^{m} p(b_j) \sum_{i=1}^{n} p_{b_j}(a_i) \log_2 p_{b_j}(a_i). \tag{5.5}$$

Часто бывает удобнее воспользоваться подобным же соотношением, которое получается на основе равенства

$$I(\beta, \alpha) = I(\alpha, \beta) = H(\beta) - H_\alpha(\beta)$$

Тогда

$$I(\alpha, \beta) = -\sum_{j=1}^{m} p(b_i) \log_2 p(b_i) + \sum_{i=1}^{n} p(a_i) \sum_{j=1}^{m} p_{a_i}(b_j) \log_2 p_{a_i}(b_j) \tag{5.6}$$

Таким образом, для определения информации сигнала на приемном конце канала необходимо знание априорных и апостериорных вероятностей. Справедливо и обратное утверждение: знание априорных и апостериорных вероятностей позволяет установить (вычислить) информацию, связанную с переданным сигналом.

Как уже указывалось, апостериорные вероятности определяются свойствами канала связи, а априорные — особенностями источника (точнее, кодера). Следовательно, воспользовавшись (5.5) или (5.6) и варьируя значения $p(a_i)$ в допустимых

по условию задачи пределах, можно найти наибольшее значение $\max\{I(\beta,\alpha)\}$. Тогда

$$C = \frac{I_s^{\max}}{\tau} = \nu_m \max\{I(\beta,\alpha)\}. \tag{5.7}$$

Полученное выражение определяет порядок решения задачи о нахождении пропускной способности конкретного канала:

1) исходя из особенностей канала, определить априорные и апостериорные вероятности;

2) варьируя $p(a_i)$ и пользуясь (5.5), найти максимальное информационное содержание элементарного сигнала;

3) по (5.7) вычислить пропускную способность.

**Пример 5.2.**

Пусть в канале отсутствуют помехи или они не препятствуют передаче. Тогда $m = n$, сигналы на приемном конце совпадают с отправленными. Это означает, что апостериорные вероятности $p_{b_i}(a_j) = 1$ при $i = j$ и $p_{b_i}(a_j) = 0$ при $i \neq j$. В этом случае $H_\beta(\alpha) = 0$ и, следовательно, $\max\{I(\beta,\alpha)\} = \max\{H(\alpha)\} = \log_2 n$ (это значение достигается, если появление всех входных сигналов равновероятно). Из (5.6) получаем $C = \nu_m \log_2 n$, что совпадает с (5.3). Тем самым показано, что определение (5.3) пропускной способности канала без помех является частным случаем более общего определения (5.6).

Рассмотрим некоторые примеры каналов передачи информации с помехами.

### 5.3.2. Однородный двоичный симметричный канал

Пусть на вход канала подаются сигналы двух типов (0 и 1 — например, импульс и пауза) и они же принимаются на выходе, т. е. $\{a\} = \{b\}$, $m = n$. Пусть, далее, вероятность ошибки передачи для обоих сигналов одинакова и равна $p$; такой канал называется *двоичным симметричным* (относительно инверсии $0 \leftrightarrow 1$). Если вероятность ошибки $p$, то согласно (А.8) для дополнительных событий вероятность безошибочной передачи равна $1 - p$.

Применяя принятые выше обозначения для апостериорных вероятностей, можно записать

$$p_0(1) = p_1(0) = p; \quad p_0(0) = p_1(1) = 1 - p,$$

Этим соотношениям может быть дана графическая интерпретация (рис. 5.3) (он называется графом канала). Линии со стрелками указывают, в какие принимаемые сигналы могут перейти

те, что отправлены на входе; рядом со стрелками указаны вероятности соответствующих переходов.

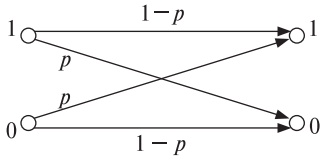



**Рис. 5.3.** Граф канала

Найдем пропускную способность канала, воспользовавшись определением (5.7). Оно, в свою очередь, требует вычисления $I(\beta,\alpha)$ (или $I(\alpha,\beta)$) и установления ее максимума как функции $p$. Если по-прежнему опыт $\alpha$ состоит в распознавании сигнала на входе канала, а опыт $\beta$ — на выходе, с учетом того, что $b_1 = a_1$, а $b_2 = a_2$, и воспользовавшись выражением (5.6), получаем

$$H_0(\beta) = H_1(\beta) = -(1-p)\log_2(1-p) - p\log_2 p.$$

и

$$\begin{aligned} H_\alpha(\beta) &= p(0)H_0(\beta) + p(1)H_1(\beta) = \\ &= \{p(0)+p(1)\}\{-(1-p)\log_2(1-p) - p\log_2 p\} = \\ &= -(1-p)\log_2(1-p) - p\log_2 p, \end{aligned}$$

поскольку $p(0)+p(1)=1$ как достоверное событие. Другими словами, в рассматриваемом канале энтропия $H_\alpha(\beta)$ не зависит от значений $p(0)$ и $p(1)$, а определяется только вероятностью искажений сигнала при передаче. Следовательно,

$$C = \nu_m \max\{H(\beta) + (1-p)\log_2(1-p) + p\log_2 p\}.$$

Поскольку $m = 2$, $\max\{H(\beta)\} = 1$. Таким образом, окончательно для пропускной способности симметричного двоичного канала имеем

$$C(p) = \nu_m\{1 + (1-p)\log_2(1-p) + p\log_2 p\}.$$

График функции $C(p)/C_0$ изображен на рис. 5.4. Максимального значения равного $\nu_m$ функция $C(p)$ достигает при $p = 0$ (очевидно, это означает отсутствие помех) и при $p = 1$ — это соответствует ситуации, когда канал полностью инвертирует входные сигналы (т. е. заменяет 0 на 1, а 1 на 0) — это не служит препятствием для однозначной идентификации посланного сигнала по принятому и, следова-

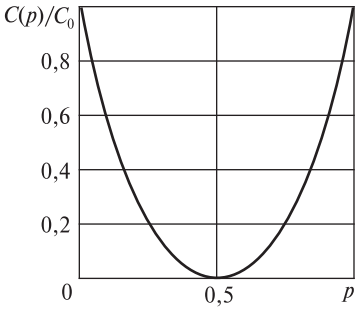



**Рис. 5.4.** График функции $C(p)/C_0$

тельно, не снижает пропускной способности канала. Во всех остальных ситуациях $0 < p < 1$ и $C(p) < \nu_m$. Наконец, при $p = 0{,}5$ пропускная способность становится равной 0 — это вполне естественно, поскольку вероятность искажения 0,5 соответствует ситуации, при которой независимо от того, какой сигнал был послан, на приемном конце с равной вероятностью может появиться любой из двух допустимых сигналов. Ясно, что передача в таких условиях оказывается невозможной.

Поскольку канал двоичный, согласно (5.3) $\nu_m = C_0$. Проведя соответствующую замену, получим

$$C = C_0\{1 + (1-p)\log_2(1-p) + p\log_2 p\}. \tag{5.8}$$

Выражение в фигурных скобках не превышает 1, следовательно, справедливо соотношение $C \leqslant C_0$, т. е. можно считать доказанным, что наличие помех снижает пропускную способность (и даже может сделать ее равной 0).

**Пример 5.3.**

Каково отношение $C/C_0$, если средняя частота появления ошибки при передаче сообщения в двоичном симметричном канале составляет 1 ошибочный сигнал на 100 переданных?

Очевидно, вероятность появления ошибки передачи $p = 0{,}01$. Следовательно, из (5.8) получаем

$$\frac{C}{C_0} = 1 + p\log_2 p + (1-p)\log_2(1-p) \approx 0{,}9192,$$

т. е. пропускная способность канала снизилась приблизительно на 8 %.

Следует заметить, что отношение $C/C_0$ убывает при увеличении вероятности искажения довольно быстро, что иллюстрируется таблицей:

| Число ошибок передачи (на 100) | 1 | 10 | 25 | 50 |
|---|---|---|---|---|
| Вероятность ошибки | 0,01 | 0,10 | 0,25 | 0,50 |
| $C/C_0$ | 0,92 | 0,53 | 0,19 | 0 |

Отдельно следует обсудить вопрос, каким образом может быть установлен факт, что канал является симметричным и как определить вероятность искажения. Эти данные не вытекают ни из какой теории и могут быть получены только опытным путем.

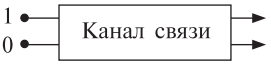



**Рис. 5.5.** Модель канала

На вход в канал подаются большое количество (для обеспечения заметной статистики) чередующихся сигналов 0 и 1 (рис. 5.5), а на приемных концах подсчитывается, сколько пришло тех или иных

сигналов. Если число искаженных сигналов на обоих приемных концах одинаково (близко), то можно делать вывод о симметричности канала. Отношение количества искаженных (ошибочных) импульсов к числу отправленных, очевидно, даст вероятность искажения.

### 5.3.3. Однородный симметричный канал со стиранием

Снова рассмотрим двоичный канал (на входе сигналы 0 и 1 с вероятностями появления $p(0)$ и $p(1)$ соответственно). Допустим, что на приемном конце любой из них с вероятностью $p$ может быть интерпретирован как противоположный (см. п. 5.3.2), но, помимо этого, с вероятность $q$ искажения в канале оказываются такими, что из-за конечной точности приемника принятый знак не идентифицируется ни с одним из поступающих на вход.

Другим словами, сигнал искажается настолько, что становится «неузнаваемым» (рис. 5.6). В таком случае можно считать, что принят новый сигнал $s$, появление которого следует интерпретировать как пропажу (стирание) входного сигнала (как если бы при переписывании мало разборчивого текста все непонятные буквы мы заменяли бы каким-то знаком) — по этой причине канал назван *двоичным симметричным со стиранием*. По-видимому, вероятность стирания $q > p$, поскольку это событие требует меньшего искажения исходного сигнала, чем его инверсия.

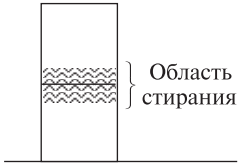



**Рис. 5.6.** Искажение сигнала

Апостериорные вероятности в этом случае составят:

$$p_0(0) = p_1(1) = 1 - p - q; \quad p_0(1) = p_1(0) = p; \quad p_0(s) = p_1(s) = q,$$

Граф канала будет выглядеть так, как на рис. 5.7.

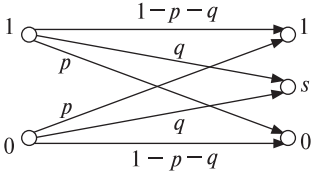



**Рис. 5.7.** Граф канала

Для определения пропускной способности вновь, как и при решении предыдущей задачи, воспользуемся соотношением (5.6). Расчет условной энтропии дает:

$$H_0(\beta) = H_1(\beta) = -(1-p-q)\log_2(1-p-q) - p\log_2 p - q\log_2 q;$$

$$H_\alpha(\beta) = p(0)H_0(\beta) + p(1)H_1(\beta) =$$

$$= \{p(0) + p(1)\}\{-(1-p-q)\log_2(1-p-q) - p\log_2 p - q\log_2 q\} =$$

$$= \{-(1-p-q)\log_2(1-p-q) - p\log_2 p - q\log_2 q\};$$

вновь учтено, что $p(0)+p(1)=1$ как достоверное событие. Таким образом,

$$I(\alpha,\beta)=H(\alpha)+(1-p-q)\log_2(1-p-q)-p\log_2 p-q\log_2 q.$$

Поскольку $H_\alpha(\beta)$ не зависит от значений априорных вероятностей, $I(\alpha,\beta)$ достигает максимума при таких $p(0)$ и $p(1)$, когда наибольшее значение приобретает энтропия $H(\beta)$. Для нахождения $H(\beta)$ необходимо знать вероятности всех сигналов, появляющихся на выходе из канала (обозначим эти вероятности $q_j$, $j=0,1,2$).

Вероятность появления $s$ (стирания) уже установлена: $q_2=q$. Для 0 вероятность $q_0=p(0)(1-p-q)+p(1)p$; аналогично для 1 находим $q_1=p(1)p+p(1)(1-p-q)$. Тогда

$$H(\beta)=-q_1\log_2(q_1)-q_2\log_2 q_2-q\log_2 q.$$

Поскольку $q$ определяется особенностями канала и не зависит от априорных вероятностей сигналов на входе, наибольшим $H(\beta)$ будет при максимальном значении выражения $-q_1\log_2 q_1-q_2\log_2 q_2$, причем при любых $p(0)$ и $p(1)$ справедливо $q_1+q_2=1-q$. Можно показать, что указанное выражение достигает максимума при условии $q_1=q_2=0{,}5(1-q)$. Тогда

$$\max\{H(\beta)\}=-(1-q)\log_2\frac{1-q}{2}-q\log_2 q;$$

$$\max\{I(\alpha,\beta)\}=$$

$$=-(1-q)\log_2\frac{1-q}{2}+(1-p-q)\log_2(1-p-q)+p\log_2 p=$$

$$=(1-q)[1-\log_2(1-q)]+(1-p-q)\log_2(1-p-q)+p\log_2 p.$$

Окончательно для пропускной способности двоичного симметричного канала со стиранием имеем

$$C=\nu_m\{(1-q)[1-\log_2(1-q)]+(1-p-q)\log_2(1-p-q)+p\log_2 p\}.$$

Проанализируем полученный результат $C=C(p,q)$: $C$ будет уменьшаться при увеличении как $p$, так и $q$. Если вероятности $p$ и $q$ отличны от 0, то, как видно из полученного выражения, $C<C_0$.

В реальных двоичных каналах со стиранием $p<q$, т. е. вероятность такого искажения входного сигнала, при котором его невозможно распознать, выше вероятности такого искажения, при котором сигнал становится похожим на второй из используемых сигналов.

В тех ситуациях, когда $p$ пренебрежимо мала и единственным искажением оказывается стирание сигнала, пропускная способность оказывается равной $C=\nu_m(1-q)$. График этой функции

представлен на рис. 5.8. Полученный результат представляется вполне закономерным: при $p = 0$ из $\nu_m$ двоичных сигналов, передаваемых по каналу за единицу времени, в среднем $\nu_m q$ будет «стираться», но при этом остальные $\nu_m(1-q)$ сигналов будут на приемном конце расшифровываться без потерь, и с каждым из них связан ровно 1 бит информации.

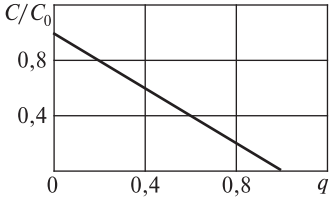



**Рис. 5.8.** График функции $C(q)$

Заканчивая рассмотрение характеристик реального дискретного канала передачи информации, следует сделать следующие заключения.

1. Помехи, существующие в реальном канале связи, приводят к снижению его пропускной способности (по сравнению с аналогичным каналом без помех).

2. Пропускная способность реального канала может быть рассчитана по известным априорным и апостериорным вероятностям. Для их определения требуются статистические исследования передачи информации в канале.

## 5.4. Передача информации по непрерывному каналу

Выше мы обсуждали передачу информации в канале связи посредством дискретных сигналов. Однако при этом непосредственно сам канал связи — проводники, электромагнитное поле, звук, оптоволоконные линии и пр. — свойствами дискретности не обладает. Другими словами, по тем же каналам может передаваться и аналоговая информация — характер передаваемых сигналов определяется передатчиком. Линии связи, основанные на использовании аналоговых сигналов, имеют весьма широкую область практического применения — это радио- и телевизионная связь, телефон и модем, различные телеметрические каналы и пр.

***Непрерывным** называется канал связи, который обеспечивает передачу непрерывных (аналоговых) сигналов.*

Схема непрерывного канала представлена на рис. 5.9.

Непрерывные сигналы, поступающие в канал связи из передатчика (ПрдС), описываются некоторой непрерывной функцией времени $Z(t)$. Ограничения на значения этой функции задаются

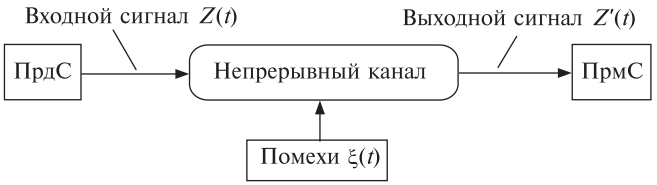



**Рис. 5.9.** Схема дискретного канала передачи информации

средней мощностью передаваемых сигналов $\langle P_Z \rangle$. Другой характеристикой непрерывного канала является ширина полосы пропускания — интервал частот сигналов $\nu_{\min} - \nu_{\max}$, которые могут распространяться в данном канале. Если по своему физическому смыслу $Z$ является напряжением или силой электрического тока, то при неизменном электросопротивлении канала связи $\langle P_Z \rangle \approx \langle Z^2 \rangle$, т. е. мощность сигнала определяет его амплитуду и средний квадрат значения параметра сигнала.

Сигналы на выходе канала $Z'(t)$, поступающие в приемник (ПрмС), также являются аналоговыми и формируются в результате сложения сигналов на входе канала и помех — их можно описать некоторой непрерывной функцией времени $\xi(t)$; в результате

$$Z'(t) = Z(t) + \xi(t).$$

Явный вид функции помех заранее неизвестен. Поэтому для количественного описания прохождения сигналов по непрерывному каналу приходится принимать ту или иную модель помех и модель канала. Наиболее распространенной является модель гауссовского канала: принимается, что помехи, будучи непрерывными случайными величинами, подчиняются нормальному (гауссовскому) статистическому распределению с математическим ожиданием (средним значением) равным нулю ($m_\xi = 0$):

$$w(\xi) = \frac{1}{\sqrt{2\pi}\sigma_\xi} \exp\left(-\frac{\xi^2}{2\sigma_\xi^2}\right); \quad \sigma_\xi^2 = D_\xi.$$

Эта функция имеет единственный параметр $\sigma_\xi$, квадрат которого называется дисперсией и имеет смысл *средней мощности помех* в канале с единичным электросопротивлением, т. е.

$$\sigma_\xi^2 \approx P_\xi.$$

Если при этом выполняется условие, что в пределах полосы пропускания средняя мощность помех оказывается одинаковой на всех частотах, а вне этой полосы она равна нулю, то такие помехи называются *белым шумом*.

Не вдаваясь в математическую сторону вывода, укажем, что основываясь на аппарате, описывающем непрерывные случайные величины, можно получить выражение для информации, связанной с отдельным аналоговым сигналом, а на его основе вывести формулу для пропускной способности непрерывного канала. В частности, для принятой модели гауссовского канала с белым шумом получается выражение, которое также называется *формулой Шеннона*:

$$C = \nu_m \log_2 \left(1 + \frac{P_Z}{P_\xi}\right), \tag{5.9}$$

где $P_Z$ — средняя мощность сигнала; $P_\xi$ — средняя мощность помех; $\nu_m \equiv \nu_{\max}$ — наибольшая частота в полосе пропускания.

Замечания к (5.9):

1) из (5.9) следует, что при фиксированной $\nu_m$ пропускная способность определяется только отношением мощностей сигнала и помех. Ограничение пропускной способности непрерывного канала связано с тем, что любые используемые для связи сигналы имеют конечную мощность;

2) $C = 0$ только при $P_Z = 0$. То есть непрерывный канал обеспечивает передачу информации даже в том случае, если уровень шумов превышают уровень сигнала — это используется для скрытой (неперехватываемой) передачи;

3) повысить пропускную способность непрерывного канала можно за счет расширения полосы пропускания.

В табл. 5.1 приведены характеристики некоторых каналов связи*.

Таблица 5.1

| Вид связи | $\nu_m$, Гц | $P_s/P_n$ | $C$, бит/с |
|---|---|---|---|
| Телеграф | 120 | $\approx 2^6$ | 640 |
| Телефон | $3 \cdot 10^3$ | $\approx 2^{17}$ | $5 \cdot 10^4$ |
| Телевидение | $7 \cdot 10^6$ | $\approx 2^{17}$ | $130 \cdot 10^6$ |
| Компьютерная сеть | | | до $10^9$ |
| Слух человека | $20 \cdot 10^3$ | | $5 \cdot 10^4$ |
| Глаза человека | | | $5 \cdot 10^6$ |

* Данные взяты из книги Ф. Бауэра и Г. Гооза [5, с. 65–66]. — *Прим. автора.* В 2010 году принят стандарт передачи по сети Ethernet на скорости 100 Гбит/с, а максимальная скорость передачи (пока в экспериментах) по оптоволоконной линии связи достигла 1,6 Тбит/с (Тера = $10^{12}$). — *Прим. ред.*

Из сопоставления данных видно, что пропускная способность телефонного канала связи совпадает с пропускной способностью органов слуха человека. Однако она существенно выше скорости обработки информации человеком, которая составляет не более 50 бит/с. Другими словами, человеческие каналы связи допускают значительную избыточность информации, поступающей в мозг.

Мы коснулись лишь одной (хотя и наиболее часто применяемой) модели непрерывного канала. В реальных каналах действие помех на входные сигналы может быть гораздо сложнее и, соответственно, гораздо хуже поддаваться математическому описанию.

## 5.5. Способы передачи информации в компьютерных линиях связи

В компьютерных линиях связи используются два способа передачи:

- *параллельный*, когда передаются одновременно все биты машинного слова;
- *последовательный*, когда биты передаются поочередно, начиная с младшего.

### 5.5.1. Канал параллельной передачи

Для одновременной передачи нескольких сигналов, очевидно, требуется линия связи, количество проводников в которой совпадает с числом передаваемых сигналов. Такая линия связи называется *шиной*. Количество проводников определяет *ширину*, или *разрядность* шины. Например, во внутренних линиях компьютера могут использоваться 16-, 32- и 64-разрядные шины. На рис. 5.10 показана линия параллельной передачи, связывающая регистр АЛУ и ячейку памяти компьютера.

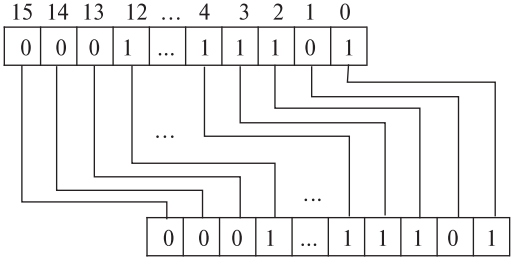



**Рис. 5.10.** Параллельный способ передачи данных в компьютере

Шина обеспечивает наиболее быстрый способ передачи информации, поскольку за два такта синхрогенератора компьютера передается все разряды шины. В общем случае, если $\nu_m$ — частота генератора, $h$ — разрядность шины, а $\eta$ — число тактов, за которые осуществляется передача, то согласно (5.2) и (5.4) пропускная способность канала параллельной передачи

$$C = \frac{\nu_m h}{\eta}. \tag{5.10}$$

Например, при тактовой частоте генератора 4 ГГц, ширине шины 64 бит и передаче за 2 такта пропускная способность канала передачи согласно (5.10) составит

$$C = 64 \cdot 4 \cdot 10^9/2 = 128 \cdot 10^9 \text{ бит/с} = 128 \text{ Гбит/с}.$$

Параллельный способ передачи используется во внутренних линиях связи компьютера (на материнской плате).

К недостаткам параллельного способа передачи следует отнести, во-первых, то, что невозможна передача на большие расстояния (более нескольких метров), поскольку между параллельными проводниками имеется электроемкость, увеличивающаяся с их длиной, существование которой приводит к тому, что при протекании импульса по какому-либо одному проводнику возникают наводки в других. Во-вторых, данный способ требует многожильных специальных проводов для связи, что существенно повышает стоимость линии*.

### 5.5.2. Последовательная передача данных

Для передачи информации на большие расстояния, например при объединении компьютеров в сети, используется последовательный способ передачи. Возможны два режима последовательной передачи: *синхронный* и *асинхронный*.

При *синхронной* передаче каждый передаваемый бит сопровождается импульсом синхронизации, информирующим приемник о наличии в линии информационного бита. Синхронизирующие импульсы управляют приемом информации. Следовательно, между передатчиком и приемником должны быть протянуты минимум три провода: один — для передачи данных, второй — для передачи синхроимпульсов, третий — общий заземленный.

---

* Стандарт Ethernet 100 Гбит основан на параллельной передаче по 8 оптоволоконным кабелям, основная проблема при этом — компенсация различных задержек передачи по отдельным кабелям. — *Прим. ред.*

Если же расстояние между источником и приемником составляет несколько метров, то каждый из сигналов (информационный и синхронизирующий) приходится посылать по экранированному кабелю, что значительно увеличивает стоимость линии связи. Кроме того, такая передача оказывается целесообразной, если передается некоторый массив символов (не отдельные символы). Оба перечисленных обстоятельства приводят к тому, что синхронный способ связи не получил широкого распространения.

*Асинхронный* способ передачи не требует синхронизации действий приемника и передатчика; по этой причине для связи достаточно линии из двух проводников, причем, оказывается возможным использование даже телефонных линий, т. е. неспециализированных компьютерных. При этом источник и приемник информации должны быть согласованы по формату и скорости передачи.

Передача выполняется машинными словами (информационными битами), дополненными несколькими служебными. Рассмотрим пример передачи 8-битного слова с одним контрольным битом. В отсутствии передачи в линии поддерживается уровень сигнала, соответствующий логической 1 (например, +5 В).

Передатчик может начать пересылку в любой момент посредством генерации *стартового бита*, который переводит линию в состояние логического 0 на время продолжительности элементарного сигнала $\tau_0$ — по нему приемник узнает, что передача началась. Затем происходит передача информационных битов, начиная с младшего (0-го). За ними передается контрольный бит четности. Наконец, за ним следует *стоповый бит* (их может быть два), который вновь переводит линию в состояние ожидания, т. е. логической единицы. Вся передаваемая цепочка сигналов от стартового до стопового бита называется *кадром*.

Передача следующего кадра может начаться сразу после стопового бита, причем новый стартовый бит может быть послан в любой момент времени (не обязательно кратный $\tau_0$), например $0{,}67\tau_0$ или $3{,}45\tau_0$ — поэтому передача и называется асинхронной. Безусловно, данная схема передачи требует предварительного согласования передатчика и приемника по продолжительности элементарного сигнала. Кроме того, для обеспечения максимальной защищенности сигнала от искажений приемник настраивается на считывание бита в середине его длительности.

Помимо информационных и контрольных битов в последовательном способе передачи кадр, как было сказано, дополняется

еще двумя–тремя граничными битами. Это, естественно, приводит к увеличению избыточности кода и суммарной времени передачи. Поскольку биты передаются по очереди, скорость передачи ниже, чем в параллельном способе (при одинаковых частотах генераторов). Тем не менее, и в последовательных линиях скорость может достигать единиц Гбит/с — такой скорости более чем достаточно для передачи, например, телевизионного сигнала. При этом неоспоримым преимуществом данного способа является то, что в нем нет ограничений на дальность передачи.

Последовательный способ передачи используется в настоящее время и для обмена компьютера с внешними устройствами — жесткими дисками, принтерами, сканерами и пр., подключаемыми через порты PCA, SATA, USB с пропускной способностью 3–6 Гбит/с.

## Контрольные вопросы и задания к гл. 5

1. Какими факторами определяется близость реального канала связи к идеальному?

2. Приведите примеры процессов, используемых для передачи информации, и связанных с ними сигналов помимо указанных в тексте.

3. Что произойдет при попытке передачи информации со скоростью, превышающей пропускную способность канала связи? Почему?

4. Оцените длину звукового файла, если требуется обеспечить телефонное качество звучания в течение 10 с, а на запись одного отсчета отводится 6 бит?

5. Человек может осмысленно читать со скоростью 15 знаков в секунду. Оцените пропускную способность зрительного канала в данном виде деятельности.

6. Оцените пропускную способность слухового канала радиста, принимающего сигналы азбуки Морзе, если известно, что для распознавания одного элементарного сигнала ему требуется 0,2 с.

7. При дискретизации аналогового сообщения число градаций при квантовании равно 64, а частота развертки по времени — 200 Гц. Какой пропускной способности требуется канал связи без шумов для передачи данной информации, если используется равномерное двоичное кодирование?

8. Для передачи сообщений, представленных с помощью телеграфного кода, используется канал без помех с пропускной способностью 1000 бит/с. Сколько знаков первичного алфивита можно передать за 1 с по данному каналу?

9. Почему происходит потеря информации при ее передаче по каналу с шумом?

10. Определите, на какую долю снижается пропускная способность симметричного двоичного канала с шумом по сравнению с идеальным каналом, если вероятность появления ошибки передачи составляет: (a) 0,001; (b) 0,02; (c) 0,1; (d) 0,5; (e) 0,98. Поясните полученные результаты.

11. С помощью пакета Excel или MathCAD постройте график зависимости отношения $C/C_0$ от вероятности появления ошибки передачи $p$ в симметричном канале с шумом.

12. Выведите формулу для пропускной способности несимметричного однородного двоичного канала.

13. С помощью пакета Excel или MathCAD для симметричного двоичного канала со стиранием постройте график зависимости $C(p)$ при фиксированном $q$, а также $C(q)$ при фиксированном $p$.

14. Покажите, что в симметричном двоичном канале со стиранием энтропия $H(\beta)$ достигает максимума при условии $q_1 = q_2 = 0{,}5(1 - q)$.

15. Почему параллельный способ не применяется для передачи информации на большие расстояния? Каким образом, в принципе, можно увеличить дальность параллельной передачи?

16. Сколько времени будет выводиться на экран дисплея картинка размером 1280×768 пиксель при цветовом режиме 32 бит на цвет, если для обмена используется 32-разрядная шина, частота тактового генератора составляет 4 ГГц и передача осуществляется за 2 такта?

17. Почему при асинхронной последовательной передаче не требуется синхронизации работы источника и приемника?

18. Алфавит обитателей планеты Тау-Кита содержит следующий набор жестов:

Предложите вариант равномерного двоичного кодирования этого алфавита, а также определите избыточность кода при последовательной передаче с одним битом четности.

# 6 Обеспечение надежности передачи и хранения информации

В предыдущем разделе на основе энтропийного подхода было показано, что передача сигналов по реальным каналам связи сопровождается частичной потерей информации, что, естественно, порождает усилия по предотвращению этого. Поиск методов повышения надежности передачи является одной из важнейших задач теории связи. Все эти методы могут быть отнесены к одной из трех групп (хотя применяться они могут и совместно):

- *радиотехнические* — использование каналов (линий) связи с защитой от помех, выбор рациональных значений уровней сигналов, частот передачи, повышение добротности линии связи и пр.;
- *организационные* — создание структурной избыточности линий связи: использование нескольких параллельных каналов передачи одной и той же информации или каналов с обратной связью;
- *математические* — использование методов построения кодов, устойчивых к воздействию помех.

Именно методы помехоустойчивого кодирования и будут интересовать нас в данном разделе.

## 6.1. Общие подходы

Если бы вопрос «Возможна ли передача информации без потерь по каналу с шумом?» задать инженерам-связистам и радиотехникам 30–40-х гг. XX в., они, безусловно, ответили бы отрицательно, посоветовав для сокращения потерь увеличить ширину полосы пропускания канала и поднять мощность сигнала. Громадная заслуга К. Шеннона в том, что он доказал *теоретическую возможность* передачи сообщения без потерь информации по реальным каналам, если при этом выполнен ряд условий. Задача была сформулирована в виде теоремы, которая затем получила строгое математическое доказательство. Выше (см. п. 3.1)

мы ознакомились с первой теоремой Шеннона, касающейся кодирования информации при передаче по идеальному каналу связи. Критерием оптимальности кодирования служила избыточность кода, которую, как было показано, можно сделать сколь угодно близкой к нулю, например применяя блочное кодирование по методу Хаффмана.

Вторая теорема Шеннона относится к реальным каналам связи и гласит следующее:

**При передаче информации по каналу с шумом всегда имеется способ кодирования, при котором сообщение будет передаваться со сколь угодно высокой достоверностью, если скорость передачи не превышает пропускной способности канала.**

Последняя часть определения, как уже говорилось, относится и к идеальному каналу — в любых случаях, если скорости передачи будет превышать пропускную способность канала, возможна потеря информации. Смысл данной теоремы в том, что при передаче по реальным каналам можно закодировать сообщение таким образом, что действие шумов не приведет к потере информации. Это достигается за счет создания *избыточности кода* (т. е. увеличения длины кодовой комбинации); безусловно, возрастает время передачи, что следует считать платой за надежность. При этом в теореме и при ее доказательстве не указывается, каким образом следует осуществлять такое кодирование, — она лишь утверждает принципиальную его возможность; поиск же кода — самостоятельная задача.

Прежде чем обсуждать варианты построения подобных кодов, попробуем оценить, насколько велика вероятность возникновения ошибки, но не при передаче, а при хранении информации, что, впрочем, не меняет сути дела. Поучительным представляется пример, рассмотренный в книге С.А. Майорова и др.* Пластмассовые корпуса микросхем памяти компьютеров содержат, хоть и в ничтожных количествах, тяжелые элементы, которые вследствие радиоактивного распада испускают $\alpha$-частицы. Попадая в полупроводниковые элементы памяти, они могут вызвать изменение их состояния, т. е. искажение хранимой информации. Как часто можно ожидать такого сбоя? Лабораторные эксперименты показали, что «время наработки на отказ» отдельно взятого

* Майоров С.А., Кириллов В.В., Приблуда А.А. Введение в микроЭВМ. Л.: Машиностроение, 1988. — 304 с. (с. 132-133).

элемента превышает миллион лет. Казалось бы, это обстоятельство не дает оснований для беспокойства. Однако следует учесть, что общее количество подобных элементов в памяти современного компьютера весьма велико. Например, модуль емкостью 1 Мбайт содержит 8388608 двоичных элементов. Следовательно, среднее время появления ошибки составит

$$t \approx \frac{10^6}{810^6} \approx \frac{1}{8} \text{ года} \approx 45 \text{ дней}.$$

Это уже совсем небольшая величина, которая, с учетом того, что память современных ПК составляет 4–8 Гбайт, а жестких дисков доходит до Тбайт, «время на отказ» составляет доли секунды. В то же время нет экономных технических способов исключения влияния этих $\alpha$-частиц. Данная оценка показывает, что проблема защиты информации от искажения даже при ее хранении весьма актуальна.

Предположим, что помехи привели к искажению сигнала и утрате связанной с ним информации. В чем может состоять устранение последствия? Очевидно, возможны две ситуации.

В первом случае принятая информация может содержать признак, по которому можно было бы судить: состоялось искажение или нет. Простейший вариант — это контроль четности (он, в частности, широко использовался в модемах). Кодирование заключается в добавлении к каждому байту девятого бита таким образом, чтобы дополнить количество единиц в комбинации до четного. Поскольку искажение сигнала состоит в его инверсии, т. е. замене 0 на 1 или наоборот, нечетная сумма единиц сразу зафиксирует ошибку передачи. Однако локализация ошибки (т. е. указание, в каком бите она допущена) невозможна — это требует повторной передачи байта.

Во втором случае ошибку можно попытаться исправить. Например, за счет трехкратной передачи каждого бита. Если окажется искаженным один бит из трех, то ошибка будет исправлена (по большинству). При тройном повторении для повышения надежности три бита располагают не подряд, а на фиксированном расстоянии друг от друга. Вместе с тем использование тройного повторения втрое снижает скорость передачи данных, что нельзя считать приемлемым с экономической точки зрения.

Таким образом, задача состоит в поиске таких методов кодирования информации, которые позволили бы контролировать правильность передачи (хранения) и при обнаружении ошибки исправлять ее.

***Помехоустойчивыми** (корректирующими) называются коды, позволяющие обнаружить и при необходимости исправить ошибки в принятом сообщении.*

Далее построение подобных кодов будем рассматривать с учетом следующих исходных положений:

- помехоустойчивые коды строятся на *основе равномерных первичных*;
- помехоустойчиные коды также являются *равномерными;*
- применяется только *двоичное кодирование.*

При построении помехоустойчивых кодов, как уже было сказано, можно выделить две ситуации:

- кодирование обеспечивает *только установление факта* искажения информации (*коды, обнаруживающие ошибку*) — в этом случае исправление осуществляется ее повторной передачей;
- кодирование позволяет декодеру *локализовать* и *автоматически исправить* ошибку передачи (хранения) (*коды, исправляющие ошибку*).

За счет чего коды могут обрести качество помехоустойчивости? Рассмотрим простой пример.

Пусть в нашем распоряжении для записи кодов имеются 3-битные ячейки. Поскольку $k = 3$, общее число кодовых комбинаций $n = 2^k = 8$:

$$000, 001, 010, 011, 100, 101, 110, 111.$$

Представим, что любой из этих кодов мы отправили в канал связи, где исказился (инвертировался) один его бит (например, послали 101, получили 100) — новый (искаженный) код принадлежит к той же таблице кодов и, следовательно, мы не можем узнать ошибочный он или верный.

Способность кода к обнаружению и исправлению ошибки основана на создании *избыточности* кодируемого сообщения. Избыточные коды формируются по определенным правилам.

Пусть $N$ — число знаков первичного алфавита. Длина равномерного двоичного кода $K(A, 2) \geqslant \log_2 N$ — при этом каждый знак получает свою уникальную последовательность знаков вторичного алфавита. Общее число кодовых комбинаций $S_\text{р} = 2^{K(A,2)}$; очевидно, $S_\text{р} \geqslant N$; все эти кодовые комбинации возможны — они называются *разрешенными* (о чем свидетельствует индекс «р» у $S_\text{р}$. Поскольку используем код равномерный, в дальнейшем примем обозначение $K(A, 2) = k$ и будем называть

часть помехоустойчивого кода, составленную из указанных $k$ бит, *информационной* (поскольку именно они содержат информацию о передаваемом знаке первичного алфавита). Как уже показано, если пересылать только эти информационные биты, то любое искажение, состоящее в инверсии хотя бы одного бита, приведет к появлению новой разрешенной кодовой комбинации и, следовательно, установлено быть не может.

Возможность обнаружения и исправления ошибок в помехоустойчивых кодах достигается тем, что после первичного кодирования (установления соответствия каждому знаку первичного алфавита его кода) осуществляется *вторичное кодирование*, в ходе которого к $k$ информационным битам по определенным правилам добавляются $r$ *проверочных* (*корректирующих*) бит. В результате общая длина кодовой комбинации становится равной $n = k + r$ — в дальнейшем такие коды будем называть $(n, k)$-*кодами*, а число возможных кодовых комбинаций возрастает до $S = 2^n$. Из них не все оказываются разрешенными — их только $S_{\text{р}}$, остальные же $S_f = S - S_{\text{р}}$ комбинаций являются *запрещенными*. Если при передаче возникает ошибка, она проявится в том, что разрешенная кодовая комбинация перейдет в запрещенную — это можно отследить и даже исправить. Такое обнаружение, очевидно, окажется невозможным, если в результате ошибки передачи одна разрешенная кодовая комбинация перейдет в другую. В связи с этим возникает проблема поиска таких способов избыточного кодирования, при которых вероятность перехода одной разрешенной кодовой комбинации в другую была бы минимальной. Задача помехоустойчивого кодирования может быть сформулирована следующим образом:

**Из $S$ возможных кодовых комбинаций выбрать $S_p$ разрешенных таким образом, чтобы получить возможность обнаружить (или исправить) все ошибки заранее оговоренной кратности при заданной модели ошибок и минимальной избыточности кода.**

Данная постановка нуждается в ряде комментариев.

1. При построении кода заранее оговариваются желаемые его свойства: «*должен обнаруживать двукратную ошибку*», «*должен исправлять однократную ошибку*» и т. д. Не может быть построено универсальных помехоустойчивых кодов, исправляющих ошибки любой кратности. Именно это подразумевается под «заранее оговоренными ошибками».

2. Под *кратностью ошибки* (далее будем обозначать $\theta$) понимается количество бит в передаваемом коде, которые могут быть искажены одновременно.

3. Шумы, приводящие к появлению ошибок передачи, могут подчиняться различным закономерностям; соответственно, построению помехоустойчивого кода должно предшествовать принятие некоторого исходного соглашения — *математической модели ошибок*; только для принятой модели код будет применим.

4. Поскольку вариантов построения помехоустойчивых кодов существует много, вводится количественная величина, характеризующая оптимальность кода — его избыточность.

Наши дальнейшие построения мы будем вести для простейшей *модели случайных независимых ошибок*, т. е. будем считать, что вероятность появления ошибки при передаче любого отдельного бита кодовой комбинации составляет $p$, а ее появление в каждом отдельном знаке (бите) кода является случайным событием, которое не зависит от того, с ошибками или без были переданы предыдущие биты.

Пусть $p$ — вероятность появления ошибки при передаче 1 бита. Тогда

$1-p$ — вероятность безошибочной передачи отдельного бита;

$(1-p)^n$ — вероятность безошибочной передачи цепочки $n$ бит;

$P_n = 1-(1-p)^n$ — вероятность появления ошибки в комбинации $n$ бит.

При малых $p$ можно воспользоваться известным соотношением $(1-\delta)^n \approx 1-n\delta$ для $|\delta| \ll 1$; тогда $P_n \approx np$. Так можно оценить суммарную вероятность появления всех ошибок. Если же требуется найти вероятность ошибки с заданной кратностью $\theta$ (например, однократной, двукратной и т. д.), то следует воспользоваться формулой Бернулли (*биноминальным распределением*)

$$P_n(p;\theta) = C_n^\theta p^\theta (1-p)^{n-\theta} = \frac{n!}{\theta!(n-\theta)!} p^\theta (1-p)^{n-\theta}. \qquad (6.1)$$

**Пример 6.1.**

Вероятность искажения отдельного бита $p = 0{,}02$, длина кодовой комбинации $n = 8$. Найти вероятность безошибочной передачи всей комбинации, вероятность ошибки передачи, а также вероятности передачи с одной, двумя и тремя ошибками.

Вероятность $P_n \approx np \approx 0{,}16$ и, следовательно, вероятность передачи без ошибки $P = 1 - P_n \approx 0{,}84$; более точное значение можно получить из (6.1), если принять $\theta = 0$, тогда

$$P_8(0{,}02;0) = C_8^0 \cdot 0{,}02^0 \cdot 0{,}98^8 = 0{,}851.$$

Аналогично

$$P_8(0,02;1) = C_8^1 \cdot 0{,}02^1 \cdot 0{,}98^7 = \frac{8!}{1! \cdot 7!} \cdot 0{,}02 \cdot 0{,}868 = 0,139;$$
$$P_8(0,02;2) = C_8^2 \cdot 0{,}02^2 \cdot 0{,}98^6 = 0{,}0099;$$
$$P_8(0,02;3) = C_8^3 \cdot 0{,}02^3 \cdot 0{,}98^5 = 0,000405.$$

Из рассмотренного примера видно, что в принятой модели наиболее вероятны ошибки *малой кратности* ($\theta = 1$ и $\theta = 2$) — на их долю приходится 99,93 % всех возможных ошибок, следовательно, помехоустойчивые коды должны обеспечить защиту именно от них и, в первую очередь, от однократных ошибок.

Следует заметить, что приведенная модель независимых ошибок является не единственной и применима далеко не всегда. В частности, в некоторых каналах возможно образование *пакетов ошибок*, т. е. нескольких следующих подряд и связанных друг с другом ошибок — для защиты от них требуются иные методы кодирования.

В помехоустойчивых кодах $k$ информационных бит дополняются $r$ проверочными, в результате чего коды становятся *избыточными;* в качестве количественной меры принимается *относительная избыточность помехоустойчивого кода*

$$F(n,k) = \frac{n-k}{k} = \frac{r}{k}, \tag{6.2}$$

которая показывает, какая часть переданной кодовой комбинации не содержит первичной информации.

Прежде чем обсуждать примеры построения помехоустойчивых кодов, рассмотрим их классификацию; она представлена на рис. 6.1.

Первый классификационный признак — коды бывают *блочными* или *непрерывными*. При *блочном кодировании* передаваемые двоичные сообщения представляют собой последовательности отдельных блоков — *кодовых комбинаций*, которыми кодируются знаки (или группы знаков) первичного алфавита. Из главы 3 известно, что если все кодовые комбинации имеют одинаковую длину, код называется равномерным; если нет — неравномерным. При декодировании удобнее (проще) иметь дело с равномерным кодом, поэтому именно он используется в помехоустойчивом кодировании. *Непрерывные* (цепные) *коды* представляют собой непрерывную последовательность бит, не разделяемую на блоки (информационные и проверочные биты в них чередуются по определенному правилу).

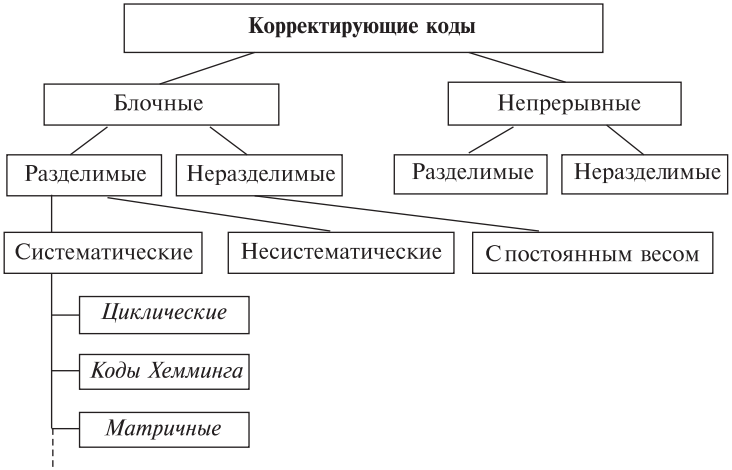



**Рис. 6.1.** Классификация корректирующих кодов

Второй классификационные признак, относящийся как к блочным, так и к непрерывным кодам, подразделяет коды на *разделимые* и *неразделимые*. *Разделимыми* называются коды, в которых информационные и проверочные биты располагаются в строго определенных позициях. В *неразделимых* кодах такой определнности нет, что затрудняет их кодирование и декодирование. Поэтому практический интерес представляют в основном разделимые коды, а из неразделимых — только *коды с постоянным весом*.

Третий классификационный признак относится только к блочным разделимым кодам — они подразделяются на *систематические* (линейные) и *несистематические*.

> ***Систематическими*** *называются такие* $(n, k)$*-коды, в которых информационные и проверочные биты связаны между собой зависимостями, описываемыми линейными уравнениями.*

В *несистематических* (нелинейных) кодах информационные и проверочные биты либо вообще не имеют связи, либо эта связь нелинейна, такие коды применяются крайне редко.

Наиболее часто в линиях связи используются систематические коды, к которым относятся *канонический*, *циклические*, *коды Хемминга*, *матричные* и ряд других — к рассмотрению именно таких кодов мы и приступим.

## 6.2. Принципы построения ($n,k$)-кодов

### 6.2.1. ($n,k$)-коды, обнаруживающие ошибки

Для того чтобы код обнаруживал и исправлял ошибки, множество всех возможных комбинаций кода $S = 2^n$ разделяется на два непересекающихся подмножества: разрешенных $S_{\text{р}} = 2^k$ и запрещенных $S_f = 2^n - 2^k$ комбинаций. Кодер использует только разрешенные комбинации. Запрещенные комбинации не используются при передаче, а могут образовываться лишь под воздействием помех.

Как уже указывалось, помехоустойчивые коды подразделяются на коды, *обнаруживающие* ошибки, и коды, *исправляющие* ошибки. Они различаются правилами размещения разрешенных кодовых комбинаций относительно запрещенных. При этом для кода с фиксированными параметрами:

**Вероятность обнаружения ошибок ($n,k$)-кода определяется его характером (способом построения) и не зависит от свойств канала связи.**

Принцип обнаружения ошибок заключается в том, что декодер проверяет принадлежность принятой комбинации разрешенному подмножеству: если код принадлежит ему, то принимается решение, что ошибок нет; если же принятая комбинация оказывается запрещенной, то принимается решение о наличии ошибки. Код может обнаруживать ошибки, если число запрещенных комбинаций $S_f > 0$. Ошибки нельзя обнаружить только в тех случаях, когда одна разрешенная кодовая комбинация перешла в другую.

Принцип отбора кодовых комбинаций, наиболее устойчивых к искажениям, предложил Р. Хемминг. Его идея состоит в том, что любую $n$-битную кодовую комбинацию можно представить в виде вектора в $n$-мерном пространстве — такое пространство получило название *кодового*, а векторы, соответствующие кодовым комбинациям, — *кодовых векторов*. Проиллюстрируем этот подход для кода (3,2): $k = 2$, следовательно, число разрешенных кодовых комбинаций $S_{\text{р}} = 2^2 = 4$; общее число кодов $S = 2^3 = 8$. Число проверочных бит $r = n - k = 1$; условимся устанавливать их значение таким образом, чтобы сумма (количество) «1» во всех кодовых комбинациях было бы четным (по этой причине такой проверочный бит называется *битом четности*).

Будем обозначать разрешенные кодовые комбинации $U_i$. Их информационная часть может принимать значения 00, 01, 10 и

Таблица 6.1

| $U_i$ | $b_1$ | $b_2$ | $b_3$ |
|---|---|---|---|
| $U_0$ | 0 | 0 | 0 |
| $U_1$ | 0 | 1 | 1 |
| $U_2$ | 1 | 0 | 1 |
| $U_3$ | 1 | 1 | 0 |

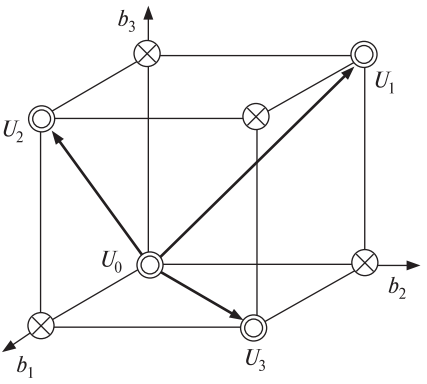



**Рис. 6.2.** Разрешенные и запрещенные кодовые векторы при $n = 3$

11 (это биты $b_1$ и $b_2$); проверочный бит ($b_3$) принимает значения, приведенные в табл. 6.1.

Любую из 8 существующих для $n = 3$ кодовых комбинаций можно считать вектором в пространстве, построенном на единичных векторах $b_1, b_2, b_3$, это иллюстрируется рис. 6.2. Отметим все четыре разрешенные комбинации кружками; остальные четыре, очевидно, будут запрещенными, их отметим крестиками. Легко видеть, что пространственно любые разрешенные комбинациями разделены сильнее, чем разрешенная и запрещенная.

Для количественного описания относительного расположения кодовых векторов Хемминг ввел понятие «*кодовое расстояние*» и определил его следующим образом:

*для любых двух кодовых комбинаций одинаковой длины*

$$U^{(1)} = (b_1^{(1)}, b_2^{(1)}, ..., b_n^{(1)}); \quad U^{(2)} = (b_1^{(2)}, b_2^{(2)}, ..., b_n^{(2)})$$

*кодовым расстоянием называется величина, определяемая соотношением*

$$d(U^{(1)}, U^{(2)}) = \sum_{i=1}^{n} (b_i^{(1)} \oplus b_i^{(2)}),$$

*где* $\oplus$ *означает «суммирование по модулю 2»**.

Например, для рассмотренного выше кода

$$d(U^{(1)}, U^{(3)}) = 0 \oplus 1 + 1 \oplus 1 + 1 \oplus 0 = 1 + 0 + 1 = 2.$$

* Суммирование по модулю 2 выполняется по следующим правилам: $0 \oplus 0 = 0$; $0 \oplus 1 = 1$; $1 \oplus 0 = 1$; $1 \oplus 1 = 0$.

Таким образом, кодовое расстояние равно *количеству несовпадающих единиц* в комбинациях.

Кодовое расстояние между данным вектором и «нулевым» $U_0 = (0, 0, 0)$ называется *весом кодовой комбинации*; очевидно, вес равен количеству единиц в кодовой комбинации. С введением этого понятия можно считать, что кодовое расстояние равно сумме по модулю 2 весов кодов.

Легко убедиться, что все расстояния между разрешенными на рис. 6.2 комбинациями $d(U^{(i)}, U^{(j)})$ также равны 2. Напротив, расстояние между разрешенной и ближайшей запрещенной, например (011) и (010), равно 1. Для того чтобы одна разрешенная комбинаяция перешла в другую разрешенную, требуется, чтобы ошибка образовалась одновременно в 2-х битах; для перехода из разрешенной в запрещенную — только в одном. Ранее, формулируя математическую модель ошибок, мы показали, что вероятность однократной ошибки гораздо выше, чем двукратной. Таким образом, действие помех, скорее всего, приведет к появлению запрещенной комбинации, что будет сигналом о некорректности передачи.

В обычных (не помехоустойчивых) кодах минимальное расстояние между соседними кодовыми комбинациями равно 1 (поскольу равномерные коды могут отличаться значением лишь одного какого-то разряда) — в результате изменение любого бита кодовой комбинации переводит ее в другую разрешенную кодовую комбинацию, что отследить и исправить невозможно.

В помехоустойчивых кодах разрешенные комбинации различаются сильнее, чем разрешенная и ближайшая запрещенная. Это и используется для обнаружения ошибки. Например, при получении на приемном конце канала связи комбинаций (001), (100) или (111) можно сразу сделать заключение об ошибке, поскольку такие коды не проходят проверку на четность. Правда, в рассмотренном примере можно зафиксировать лишь появление ошибок нечетной кратности ($\theta = 1, 3$); ошибки с $\theta = 2$ обнаружены не будут, поскольку они не изменят четность веса кодовой комбинации.

Следует понимать, что в предложенном варианте построения помехоустойчивого кода факт наличия ошибки может быть зафиксирован, однако ее местоположение (конкретный искаженный бит) — нет, и, следовательно, она не может быть исправлена. В этом и состоит смысл использования кодов, обнаруживающих

ошибку передачи: если ошибка установлена, кодовая комбинация не корректируется, а передается повтороно.

Возможны коды, в которых кодовые расстояния между разными разрешенными кодовыми комбинациями оказываются неодинаковыми, в этих случаях определяющее значение для корректирующих свойств кода имеет *минимальное* из всех возможных *кодовое расстояние* — его будем обозначать $d_{\min}$.

Теперь задумаемся на вопросом связи $d_{\min}$ и кратности ошибки $\theta$, которую код должен обнаруживать. По своему смыслу $\theta$ — количество бит в кодовой комбинации, которые могут быть искажены одновременно. Для того чтобы такое искажение не переводило одну разрешенную комбинацию в другую разрешенную, очевидно, кодовое расстояние между ними должно превышать $\theta$ минимум на 1, т. е.

$$d_{\min} \geqslant \theta + 1. \tag{6.3}$$

Выражение (6.3) называется условием обнаружения ошибки кратности $\theta$. Например, коды с $d_{\min} = 2$ могут обнаруживать лишь ошибки с $\theta = 1$, что показано выше.

### 6.2.2. ($n$,$k$)-коды, исправляющие ошибки

Принцип исправления ошибок состоит в том, что все множество комбинаций кода $2^n$ разделяется на $2^k$ *непересекающихся* подмножеств, которые называются *областями решений.* Каждая область решения примыкает к одной из разрешенных кодовых комбинаций. Если принятая комбинация попадает в одну из $2^k$ областей, то принимается решение о передаче кодовой комбинации, соответствующей этой области. Исправление ошибок будет происходить в тех случаях, когда переданный код и принятая запрещенная комбинация относятся к одной области решений. В каждую область решений включают такие запрещенные комбинации, при приеме которых наиболее вероятной является одна из разрешенных кодовых комбинаций, что соответствует *принципу максимальноro правдоподобия.*

Для *исправления* ошибки необходимо, чтобы области решения, принадлежащие каждой разрешенной комбинации, не перекрывались и любая из запрещенных комбинаций входила только в одну область, — это иллюстрируется рис. 6.3.

Для того чтобы соседние области решения не перекрывались и не соприкасались (поскольку запрещенная комбинация может входить лишь в одну область), расстояние между ближайшими

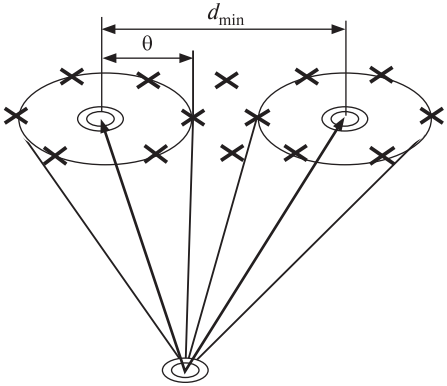



**Рис. 6.3.** Иллюстрация существования областей решений

комбинациями областей должно составлять минимум 1. Из этого вытекает условие исправления ошибки кратности $\theta$:

$$d_{\min} \geqslant 2\theta + 1. \tag{6.4}$$

Например, если ставится задача построения кода, способного исправить однократную ошибку, то минимальное кодовое расстояние между его векторами должно быть равно 3.

Как следует из условий (6.3) и (6.4), для повышения кратности обнаруживаемых и исправляемых ошибок передачи требуется повышать $d_{\min}$, а значит увеличивать число проверочных бит кода $r$, т. е. его избыточность. Задача теории помехоустойчивого кодирования состоит в построении кода с заданным $d_{\min}$ при наименьшем $r$. Общего решения этой задачи не существует, однако можно привести некоторые оценки.

Нижнюю границу $r$ для $(n, k)$ кодов, *исправляющих* ошибки, можно найти, если приравнять количество запрещенных комбинаций, которые могут быть исправлены, к числу всех ошибок кратностей от 1 до $\theta$ во всех разрешенных кодовых комбинациях — в этом случае любой вариант ошибки переводит разрешенную кодовую комбинацию в запрещенную. Число запрещенных комбинаций составляет $S_f = 2^n - 2^k$; количество ошибочных комбинаций с кратностью ошибки не выше $\theta$

$$N_f(n, \theta) = 2^k \sum_{i=1}^{\theta} C_n^i = 2^k \sum_{i=1}^{\theta} \frac{n!}{i!(n-i)!}.$$

Для возможности исправления ошибок должно выполняться соотношение $S_f \geqslant N_f$ (при этом любое искажение приводит к

появлению запрещенной комбинации), т. е.

$$2^n - 2^k \geqslant 2^k \sum_{i=1}^{\theta} C_n^i,$$

или

$$2^{n-k} - 1 \geqslant \sum_{i=1}^{\theta} C_n^i; \quad 2^r \geqslant \sum_{i=1}^{\theta} C_n^i + 1 = \sum_{i=0}^{\theta} C_n^i,$$

так как $C_n^0 = \dfrac{n!}{0!n!} = 1$. Окончательно

$$r \geqslant \log_2 \sum_{i=0}^{\theta} C_n^i. \tag{6.5}$$

Выражение (6.5) называется *границей Хэмминга* для числа контрольных бит в кодах, исправляющих ошибок.

**Пример 6.2.**

Пусть $\theta = 1$. Тогда из (6.5) находим

$$\sum_{i=0}^{1} C_n^i = C_n^0 + C_n^1 = 1 + n,$$

следовательно, $r \geqslant \log_2(1 + k + r)$, или $2^r \geqslant 1 + k + r$; окончательно,

$$2^r - r \geqslant 1 + k. \tag{6.6}$$

Поскольку в этом выражении $r$ и $k$ являются целыми, проще всего решать уравнение подбором значений. Из (6.2) и (6.6) легко получить следующую таблицу:

| $k$ | $r$ | $(n,k)$ | $F(n,k)$ |
|---|---|---|---|
| 1 | 2 | (3,1) | 2 |
| 2 | 3 | (5,2) | 1,5 |
| 3 | 3 | (6,3) | 1 |
| 4 | 3 | (7,4) | 0,75 |
| 5 | 4 | (9,5) | 0,8 |
| ... | ... | ... | ... |
| 8 | 4 | (12,8) | 0,5 |
| ... | ... | ... | ... |
| 11 | 4 | (15,11) | 0,36 |
| 12 | 5 | (17,12) | 0,42 |
| ... | ... | ... | ... |
| 26 | 5 | (31,26) | 0,19 |

Таким образом, если, например, происходит передача 4-х информационных бит ($k = 4$) и мы хотим построить код, позволяющий исправить однократную ошибку, то к информационным битам необходимо будет добавить $r = 3$ проверочных и избыточность кода составит 0,75. Из приведенной таблицы также видно, что выгоднее передавать более длинные цепочки информационных бит, поскольку $r$ изменяется не пропорционально $k$. Однако и здесь существует некий разумный предел, поскольку с ростом $k$ повышается вероятность появления ошибок высокой кратности.

## 6.3. Систематический помехоустойчивый код

### 6.3.1. Общие принципы построения систематических кодов

Рассмотрим $(n, k)$-код. Кодовую комбинацию представим в виде кодового вектора, у которого информационные биты обозначим $u_1, \ldots, u_k$, а проверочные — $p_1, \ldots, p_r$: $U = (u_1, u_2, \ldots, u_k, p_1, p_2, \ldots, p_r)$. Поскольку кодирование двоичное, ясно, что каждый бит кода может принимать значение 0 или 1.

*$(n, k)$ код называется* ***систематическим*** *(или* ***линейным****), если значения всех проверочных бит $p_j$ определяются линейными комбинациями информационных бит $u_i$, т. е.*

$$p_j = \sum_{i=1}^{k} c_{j,i} u_i = c_{j,1} u_1 \oplus c_{j,2} u_2 \oplus \ldots \oplus c_{j,k} u_k, \quad j = 1, \ldots, r. \quad (6.7)$$

Суммирование в выражении (6.7) осуществляется по модулю 2; коэффициенты в линейных комбинациях $c_{j,i}$ могут принимать значения 1 или 0 — именно ими определяются свойства кода. В систематических кодах информационные и проверочные биты занимают строго определенные позиции в кодовых комбинациях (но не обязательно, как в приведенном выше случае, сначала одни, затем другие — возможны иные правила относительного расположения частей кодовых комбинаций).

Из $2^k$ разрешенных кодовых векторов не все оказываются независимыми, т. е. они могут быть построены как линейные комбинации других векторов. Можно доказать, что *линейно независимых* (не выражающихся через другие) всегда можно выделить $k$ векторов:

*Кодовые векторы* $U^{(1)},\ldots,U^{(k)}$ *называются* ***линейно независимыми,*** *если выполняется условие*

$$\alpha_1 U^{(1)} \oplus \alpha_2 U^{(2)} \oplus \ldots \oplus \alpha_k U^{(k)} \neq 0$$

*при любых наборах значений* $\alpha_i$ ($\alpha_i = 0$ *или 1, но не все одновременно 0).*

Например, кодовые комбинации $U^{(1)} = (1111)$; $U^{(2)} = (1100)$; $U^{(3)} = (1010)$; $U^{(4)} = (1001)$ не являются линейно независимыми, поскольку при $\alpha_i = 1$ сумма $1U^{(1)} \oplus 1U^{(2)} \oplus 1U^{(3)} \oplus 1U^{(4)} = 0$. Напротив, векторы $U^{(2)}, U^{(3)}$ и $U^{(4)}$ оказываются линейно независимыми, в чем можно убедиться, перебрав различные комбинации коэффициентов $\alpha_i$.

Совокупность $k$ линейно независимых кодовых комбинаций образуют *порождающую* (*производящую*) матрицу систематического $(n,k)$-кода

$$C_{n,k} = \begin{pmatrix} U^{(1)} \\ U^{(2)} \\ \ldots \\ U^{(k)} \end{pmatrix} = \begin{pmatrix} u_{11} & u_{12} & \ldots & u_{1k} & p_{11} & \ldots & p_{1r} \\ u_{21} & u_{22} & \ldots & u_{2k} & p_{21} & \ldots & p_{2r} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ u_{k1} & u_{k2} & \ldots & u_{kk} & p_{k1} & \ldots & p_{kr} \end{pmatrix},$$

при этом сами независимые комбинации, каждая из которых формирует одну строку матрицы, называются *базисными.* Все остальные разрешенные комбинации (небазисные) могут быть получены как линейные комбинации базисных (включая «нулевую» комбинацию — у нее все $\alpha_j = 0$):

$$U_j = \sum_{i=1}^{k} \alpha_{j,i} u_i, \quad j = k+1, \ldots, 2^k,$$

(суммирование, как всегда, выполняется по mod 2).

В производящей матрице можно выделить информационную подматрицу размером $k \times k$ (будем обозначать ее $A_{k,k}$) и проверочную подматрицу размером $r \times k$ (будем обозначать ее $P_{r,k}$); т. е. производящую матрицу можно представлять как *объединение* информационной и проверочной подматриц:

$$G_{n,k} = A_{k,k} + P_{r,k}. \tag{6.8}$$

Соотношение (6.8) по сути задает алгоритм построения систематического кода:

- в соответствии с $k$ и желательными корректирующими свойствами кода ($\theta$) определить $r$;
- для выбранного $\theta$ и типа кода (обнаруживающий, исправляющий ошибку) определить $d_{\min}$;

- из каких-то соображений подобрать $k$ линейно независимых кодовых векторов, сформировать информационную подматрицу;
- подобрать уравнения проверок и строки проверочной подматрицы таким образом, чтобы вес (число единиц) каждой базисной кодовой комбинации оказался не меньше $d_{\min}$;
- после построения порождающей матрицы, остальные $2^k - k - 1$ разрешенных кодов следует найти как линейные комбинации (по mod 2) базисных (безусловно, среди них имеется нулевой вектор, что учтено в приведенном выше выражении).

Мы реализуем описаный алгоритм для построения конкретного систематического кода.

### 6.3.2. Канонический систематический код

Как уже отмечалось, процедура построения начинается с выбора информационной подматрицы $A_{k,k}$ с линейно независимыми строками. Этот выбор является неоднозначным, и, следовательно, для одних и тех же начальных условий могут быть построены различные коды.

Очевидно, наиболее простым вариантом $A_{k,k}$ с независимыми строками является единичная диагональная матрица — по этой причине код получил название *канонического*.

Далее мы формулируем перед собой задачу следующим образом: *построить канонический систематический код, исправляющий при передаче 4-х информационных бит однократные ошибки.*

1) $k = 4$, $\theta_{\text{исп}} = 1$; из (6.6) $r = 3$, т. е. строится код (7,4);

2) поскольку $\theta_{\text{исп}} = 1$, из (6.4) $d_{\min} = 3$;

3) так как в каждой строке информационной подматрицы будет находиться одна 1, следовательно, условия проверок должны быть такими, чтобы в строках проверочной подматрицы содержались бы не менее $d_{\min} = 3 - 1 = 2$ единиц;

4) информационная подматрица

$$A_{4,4} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

5) уравнения проверок — их общее количество равно числу контрольных бит, т. е. 3; однозначность выбора также отсутствует; обязательным условием является то, что каждый информаци-

онный бит должен входить как минимум в два уравнения проверки, иначе может оказаться невыполненным требование $d_{\min} = 3$:

$$\begin{cases} p_1 = u_1 \oplus u_2 \oplus u_3; \\ p_2 = u_1 \oplus u_3 \oplus u_4; \\ p_3 = u_2 \oplus u_3 \oplus u_4; \end{cases} \tag{6.9}$$

- для информационной подматрицы, пользуясь (6.9), можно построить проверочную подматрицу

$$P_{3,4} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}.$$

  Возможно, здесь стоит обратить внимание на то, что эту матрицу (с точностью до перестановки строк) можно было записать сразу, поскольку в ней присутствуют все возможные сочетания, когда строки содержат не менее 2-х единиц. Однако уравнения проверок (6.9) все равно пришлось бы составлять при разработке схемы кодера и декодера;
- объединяем информационную и проверочную части, получаем производящую матрицу

$$G_{7,4} = \begin{pmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix};$$

- линейными комбинациями базисных векторов строим все остальные; в рассматриваемом примере 4 базисный и один нулевой, следовательно, должны быть построены $2^4 - 4 - 1 = 11$ векторов:

$$\begin{aligned} &U_5 = U_1 \oplus U_2 = (1100011); \\ &U_6 = U_1 \oplus U_3 = (1010001); \\ &\cdots\cdots\cdots\cdots\cdots\cdots \\ &U_{15} = U_1 \oplus U_2 \oplus U_3 \oplus U_4 = (1111111). \end{aligned}$$

Легко проверить корректирующие свойства построенных кодов: пусть вместо $U_5 = (1100011)$ получена комбинация $(U_5)' = (0100011)$ т. е. в первом бите возникла ошибка. На основании уравнений проверок (6.9) получаем:

$$(p_1)' = 0 \oplus 1 \oplus 0 = 1 \neq p_1,$$

т. е. ошибка в $u_1$ или $u_2$, или $u_3$, но не в $u_4$;

$$(p_2)' = 0 \oplus 0 \oplus 0 = 0 \neq p_2,$$

т. е. ошибка в $u_1$ или $u_3$, но не в $u_2$;

$$(p_3)' = 1 \oplus 0 \oplus 0 = 1 = p_3,$$

т. е. ошибка в $u_3$.

После локализации ошибки достаточно инвертировать ошибочный бит (т. е. заменить 1 на 0 или 0 на 1).

Таким образом, мы убеждаемся в том, что описанная выше процедура действительно позволяет построить помехоустойчивый код с заданными корректирующими свойствами. Безусловно, построение кода и при необходимости локализация и исправление ошибки передачи производятся автоматически.

Значительным достоинством канонического систематического кода оказывается то, что локализацию и исправление ошибок в нем можно выполнять на уровне простых матричных операций.

Дадим ряд вспомогательных определений.

*Два кодовых вектора называются* ***ортогональными,*** *если их скалярное произведение равно 0, т. е. для двух векторов одинаковой длины* $U = (u_1, \ldots, u_n)$ *и* $V = (v_1, \ldots, v_n)$ *выполняется соотношение (при суммировании по mod 2):*

$$(U \cdot V) = \sum_{i=1}^{n} u_i v_i = 0.$$

Можно показать, что в любом $(n, k)$-коде для $k$ базисных векторов всегда найдется $r$ ортогональных каждому из них.

Для иллюстрации дальнейших рассуждений вновь обратимся к коду (7,4). Найдем вектор $V$, ортогональный всем четырем базисным:

$$(U_1 \cdot V) = u_1 v_1 \oplus u_5 v_5 \oplus u_6 v_6 = v_1 \oplus v_5 \oplus v_6 = 0 \Rightarrow v_1 = v_5 \oplus v_6;$$
$$(U_2 \cdot V) = u_2 v_2 \oplus u_5 v_5 \oplus u_7 v_7 = v_2 \oplus v_5 \oplus v_7 = 0 \Rightarrow v_2 = v_5 \oplus v_7;$$
$$(U_3 \cdot V) = u_3 v_3 \oplus u_5 v_5 \oplus u_6 v_6 \oplus u_7 v_7 = v_3 \oplus v_5 \oplus v_6 \oplus v_7 = 0 \Rightarrow$$
$$\Rightarrow v_3 = v_5 \oplus v_6 \oplus v_7;$$
$$(U_4 \cdot V) = u_4 v_4 \oplus u_6 v_6 \oplus u_7 v_7 = v_4 \oplus v_6 \oplus v_7 = 0 \Rightarrow v_4 = v_6 \oplus v_7.$$

В результате мы получаем систему условий, позволяющих найти $v_1$–$v_4$ по заданным $v_5$–$v_7$. Всего ортогональный векторов

должно быть 3 (поскольку $r = 3$); для их построения можно выбрать $v_5$–$v_7$ удобным образом, а $v_1$–$v_4$ рассчитать по полученным ранее соотношениям; наиболее простым будет такой выбор $v_5$–$v_7$, чтобы в каждом из трех вариантов лишь одно из значений было бы равно 1, а остальные — 0 (т. е. единичная диагональная матрица). Получившиеся таким образом комбинации образуют новую матрицу $H_{n,r}$:

$$H_{7,4} = \begin{array}{c} \begin{array}{cccc:ccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \end{array} \\ \left(\begin{array}{cccc:ccc} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right) \\ \underbrace{\qquad\qquad\qquad}_{H_{4,3}} \quad \underbrace{\qquad\qquad}_{E_{3,3}} \end{array}$$

Эта матрица называется *проверочной*. Основания к этому следующие.

Во-первых, видно, что элементы в каждой строке матрицы в колонках с первой по четвертую совпадают с установленными ранее уравнениями проверки (6.9).

Прежде чем сказать «во-вторых», выявим связь $G_{n,k}$ и $H_{n,r}$. Видно, что $H_{n-r,r} = P^{\mathrm{T}}_{r,k}$, где $P^{\mathrm{T}}_{r,k}$ — транспонированная проверочная подматрица.

> ***Транспонированной** называется матрица, строки которой являются столбцами, а столбцы — строками исходной матрицы.*

Для рассматриваемой канонической производящей матрицы $G_{7,4}$

$$P_{3,4} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}; \quad P^{\mathrm{T}}_{3,4} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix},$$

что совпадает с $H_{7-3,3}$. Таким образом, становится ясным правило построения проверочной матрицы:

$$H_{n,r} = P^{\mathrm{T}}_{r,k} + E_{r,r},$$

где $E_{r,r}$ — единичная диагональная матрица $r \times r$.

Важным оказывается следующее замечательное свойство проверочной матрицы:

**Произведение принятой кодовой комбинации на строки проверочной матрицы дает синдром ошибки**

$$Q = U' \times H^{\text{т}}_{n,r}.$$

***Синдром ошибки** — это признак, по которому ошибка передачи может быть локализована.*

Синдром ошибки совпадает со столбцом проверочной матрицы — он указывает на ошибочный бит.

Пусть $U_5 = U_1 \oplus U_2 = (1100011)$, а при приеме получена комбинация $U'_5 = (11\mathbf{1}0011)$, т. е. ошибочным является 3-й бит. Тогда

$$(1100011)\begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = (111),$$

что совпадает с третьим столбцом проверочной матрицы и соответствует 3-му информационному биту. Для исправления он должен быть инвертирован.

### 6.3.3. Кодер и декодер систематического кода

При использовании помехоустойчивых и, в частности, систематических кодов кодирование знаков первичного алфавита осуществляется в два этапа. На начальном этапе знаки первичного алфавита от источника поступают в *первичный кодер* — устройство, производящее кодирование по заданной схеме без учета возможных искажений, т. е. формируется только информационная часть кода. Далее вторичный кодер добавляет к ней проверочные знаки и формирует разрешенные помехоустойчивые кодовые комбинации. Ситуация иллюстрируется рис. 6.4, на котором показана схема кодера для рассмотренного выше (7,4)-кода.

Ясно, что в схему кодера заложены уравнения проверок.

Логика декодирования основывается на сравнении значений проверочных бит, составленных по принятой кодовой комбинации (обозначим их $p''_j$), с непосредственно принятыми проверочными битами ($p'_j$). Если $u'_i$ — принятые информационные биты кода, то в результате сравнения формируются новые проверочные биты

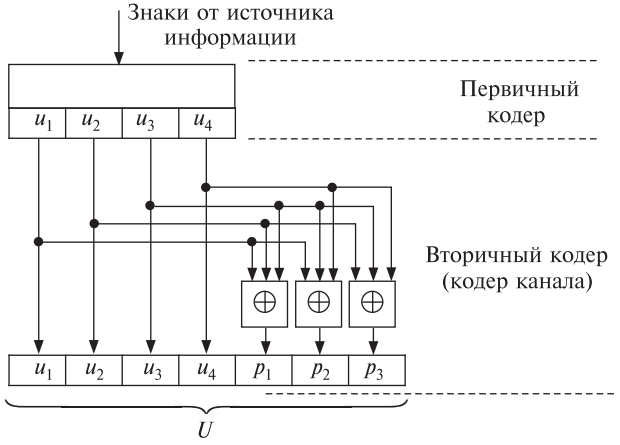



**Рис. 6.4.** Схема кодера систематического (7,4)-кода

$(q_j)$ по следующему правилу:

$$q_j = p'_j \oplus p''_j = p'_j \oplus \sum_{i=1}^{k} c_{ji} u'_i, \quad j = 1, ..., r.$$

Совокупность $q_j$ образует новый кодовый вектор $Q = (q_1, \ldots, q_r)$, который и является *синдромом ошибок*. Значение «0» в любой позиции $Q$ свидетельствует об отсутствии ошибки в бите кода, связанного с этой позицией; если синдром весь состоит из нулей («нулевой синдром»), то это означает, что вся кодовая комбинация передана безошибочно; присутствие одной или нескольких «1» говорит о наличии ошибок передачи. При этом каждому ненулевому синдрому однозначно соответствует ошибка в определенном бите принятой кодовой комбинации. Все ненулевые синдромы заносятся в таблицу — она называется *таблицей исправлений*, которая и определяет алгоритм работы декодера. Общее число ненулевых синдромов, очевидно, равно $2^r - 1$; из них для проверки требуется $n$, а для исправления только $k$ векторов.

Составим таблицу исправлений для рассмотренного в предыдущем разделе (7,4)-кода и уравнениями проверки (6.9).

Предположим, что ошибочно передан бит $u_1$, а все остальные знаки кода переданы верно. Это повлечет отличие (несовпадение) тех проверочных бит $p''_j$ и $p'_j$, в которые входит $u_1$, — это $p_1$ и $p_2$; соответственно, значения $q_j$ окажутся равными: $q_1 = 1$, $q_2 = 1$ и $q_3 = 0$. Таким образом, ошибке в $u_1$ соответствует синдром $Q = (110)$. Ошибке в проверочном бите, например $p_3$, будет

Таблица 6.2

| Синдром | $Q_1$ | $Q_2$ | $Q_3$ | $Q_4$ | $Q_5$ | $Q_6$ | $Q_7$ |
|---|---|---|---|---|---|---|---|
| Конфигурация синдрома | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| Ошибочная позиция | $p_3$ | $p_2$ | $u_4$ | $p_1$ | $u_2$ | $u_1$ | $u_3$ |
| Выход дешифратора ошибок | 0000 | 0000 | 0001 | 0000 | 0100 | 1000 | 0010 |

отвечать синдром $Q = (001)$. Продолжив аналогичные рассуждения, получим полную таблицу исправлений (табл. 6.2).

Непосредственно ошибочный бит исправляет блок декодера, называемый *дешифратором*: если синдром указывает на ошибку в одном из проверочных бит, исправлений в информационных битах не требуется и дешифратор выдает «0» на все выходы; если синдром указывает на ошибку в каком-либо информационном бите, дешифратор посылает «1» на вход сумматора по mod 2 именно этого бита, в результате чего бит инвертируется и ошибка исправляется. Схема декодера систематического кода представлена на рис. 6.5.

Таким образом, на основании проведенного рассмотрения можно сделать следующие выводы.

1. Систематичечский код в каноническом представлении может быть задан одним из четырех способов: порождающей матрицей, проверочной матрицей, схемой кодера или схемой декодера. По любому их них можно определить все остальные, а также выяснить корректирующие свойства кода.

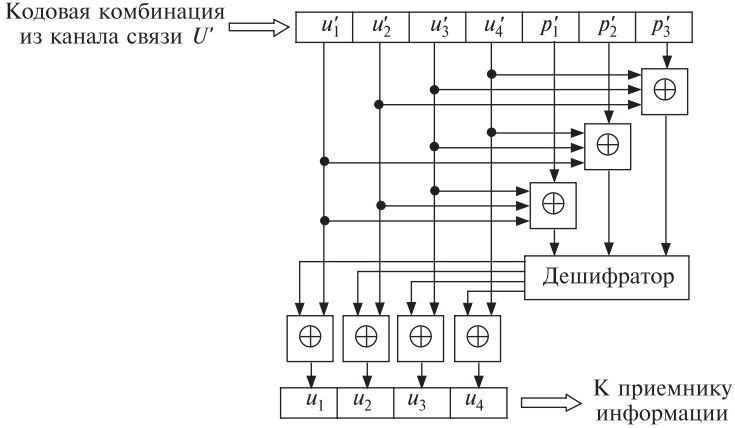



**Рис. 6.5.** Схема декодера систематического (7,4)-кода

2. Каждый столбец проверочной матрицы является синдромом ошибки соответствующего бита — фактически она является таблицей исправлений.

3. Все операции проверки алгоритмизируются и, следовательно, могут выполняться устройством.

## 6.4. Код Хемминга

Кодирование по методу, предложенному американским инженером Р. Хеммингом в 1948 г., позволяет построить оптимальный систематический код [43].

> ***Оптимальным** называется $(n,k)$-код, который обеспечивает минимальную вероятность ошибочного декодирования среди всех иных кодов с теми же $n$ и $k$.*

Как уже указывалось, систематический код может быть построен по производящей матрице или по проверочной. Хемминг построил проверочную матрицу из $n$ столбцов и $r$ строк, причем любой столбец представляет собой $r$-разрядный двоичный синдром, а его десятичное представление указывает номер ошибочного бита; синдромы расставлены в порядке возрастания:

$$H = \begin{array}{c} \begin{array}{ccccccc} q_1 & q_2 & q_3 & q_4 & \cdots & q_{n-1} & q_n \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 & ... & 1 & 1 \\ 0 & 1 & 1 & 0 & ... & 1 & 1 \\ 0 & 0 & 0 & 1 & ... & 1 & 1 \\ 0 & 0 & 0 & 0 & ... & 1 & 1 \\ ... & ... & ... & ... & ... & 1 & 1 \\ 0 & 0 & 0 & 0 & ... & 0 & 1 \end{pmatrix} \end{array}.$$

Проверочная матрица задает номера бит кодовой комбинации, которые участвуют в каждой из $r$ проверок. В качестве проверочных бит удобно выбрать такие, которые входят только один раз в каждую проверку, т. е. те, для которых $q_i$ содержит лишь один ненулевой бит; очевидно, это $q_1, q_2, q_4, q_8$ и т. д. — те биты, номера которых являются целой степенью 2. Другими словами, в отличие от канонического систематического кода, в коде Хемминга информационные и проверочные биты не разнесены в отдельные подматрицы, а чередуются. При этом принимается следующая нумерация бит (знаков кода): все биты кодовой комбинации получают номера, начиная с 1, *слева направо* (стоит напомнить, что информационные биты нумеруются с 0 и справа налево); контрольными (проверочными) оказываются биты с

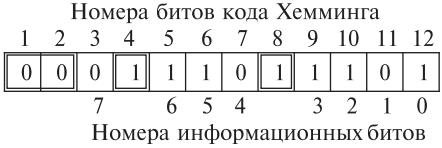



**Рис. 6.6.** Нумерация бит кода Хемминга

номерами 1, 2, 4, 8 и т. д. — все остальные являются информационными (рис. 6.6).

По проверочной матрице легко установить номера тех бит кодовой комбинации, которые «обслуживаются» данным проверочным; ясно также, что проверочные биты не контролируют друг друга.

Рассмотрим построение проверочной матрицы, таблицы проверок, а также самого кода Хемминга на примере передачи байтового первичного кода. Поскольку информационная часть содержит 8 бит, согласно (6.6) для исправления однократных ошибок требуется включения в код 4-х проверочных бит, т. е. речь идет о построении кода (12,8). Проверочная матрица для него будет иметь вид

$$H_{12,4} = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \end{matrix} \\ \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{matrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{matrix} \end{array}.$$

Номер столбца проверочной матрицы соответствует представлению синдрома ошибки в десятичной системе счисления; каждая ее строка определяет перечень проверяемых бит кодовой комбинации — представим их в виде табл. 6.3.

Из таблицы можно усмотреть следующую закономерность:

Таблица 6.3

| Проверочные биты | Контролируемые биты | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | ... |
| 2 | 2 | 3 | 6 | 7 | 10 | 11 | 14 | 15 | 18 | 19 | 22 | ... |
| 4 | 4 | 5 | 6 | 7 | 12 | 13 | 14 | 15 | 20 | 21 | 22 | ... |
| 8 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 24 | 25 | 26 | ... |
| 16 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | ... |
| 32 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | ... |

**Для любого номера проверочного бита $m$, начиная с него, $m$ бит подряд оказываются проверяемыми, затем следует группа $m$ непроверяемых бит; далее происходит чередование групп.**

**Пример 6.3.**

Рассмотрим построение кода Хемминга для конкретного байта первичного кода; пусть это будет (01110101). Размещаем его по информационным ячейкам кода Хемминга, а затем вычисляем значения проверочных бит в соответствии с таблицей проверок: $u_1 = 0$; $u_2 = 1$; $u_4 = 0$; $u_8 = 0$, после чего заполняем соответствующие проверочные ячейки:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |

т. е. получаем 12-битную кодовую комбинацию: $U = (010011100101)$.

Пусть в результате искажений на приемном конце получена комбинаций $U' = (010011100111)$, т. е. ошибочно передан 11-й бит. Декодер осуществляет проверки согласно строкам проверочной матрицы:

$$\begin{cases} h_1 = u_1 \oplus u_3 \oplus u_5 \oplus u_7 \oplus u_9 \oplus u_{11} = \\ = 1 \oplus 1 \oplus 1 = 1 & \Rightarrow \text{есть ошибка;} \\ h_2 = u_2 \oplus u_3 \oplus u_6 \oplus u_7 \oplus u_{10} \oplus u_{11} = \\ = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 1 & \Rightarrow \text{есть ошибка;} \\ h_3 = u_4 \oplus u_5 \oplus u_6 \oplus u_7 \oplus u_{12} = 0 & \Rightarrow \text{нет ошибки;} \\ h_4 = u_8 \oplus u_9 \oplus u_{10} \oplus u_{11} \oplus u_{12} = 1 & \Rightarrow \text{есть ошибка.} \end{cases}$$

Следовательно, синдром ошибки (нужно записывать в таком порядке: $h_4h_3h_2h_1$) $Q = (1011)$; по проверочной матрице получаем, что ошибка содержится в 11-м бите. Легко видеть, что номер ошибочного бита совпадает с суммой номеров проверочных бит, указавших на наличие ошибки; третий вариант состоит в том, чтобы рассматривать синдром как двоичное число и перевести его в десятичную систему счисления — в любом случае ошибка оказывается локализованной; для ее исправления достаточно инвертировать значение бита; наконец, этот же результат может быть получен матричными операциями (из них становится понятен порядок выявления и записи цифр синдрома): $Q = U' H^{\mathrm{T}}_{12,4}$.

Таблица 6.4

| Число информационных бит | Число проверочных бит | Избыточность $F(n,k)$ |
|---|---|---|
| 8 | 4 | 0,50 |
| 16 | 5 | 0,31 |
| 32 | 6 | 0,19 |

В табл. 6.4 приведена избыточность кодов Хемминга для различных длин передаваемых последовательностей. Из сопоставления видно, что выгоднее передавать и хранить более длинные кодовые комбинации.

## 6.5. Матричные коды

Весьма эффективными в плане простоты построения, высоких корректирующих свойств и низкой избыточности являются *матричные коды*.

Рассмотрим наиболее простой из них, включающий, помимо информационных, 1 бит четности. Информационные биты представляются в виде совокупности строк — матрицы, каждая строка и каждый столбец которой дополняется проверочным битом (битом четности). Пусть информационная часть кода, как всегда, содержит $k$ бит, а общее количество комбинаций, которые нужно закодировать, равно $m$. Тогда формируется исходная матрица размером $k \times m$, к которой затем справа добавляется столбец, содержащий проверки на четность каждой строки ($p$), а снизу — строка с проверками на четность каждого столбца ($q$):

$$\begin{array}{ccccc|c} u_{11} & u_{12} & u_{13} & \dots & u_{1k} & p_1 \\ u_{21} & u_{22} & u_{23} & \dots & u_{2k} & p_2 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ u_{m1} & u_{m2} & u_{m3} & \dots & u_{mk} & p_m \\ \hline q_1 & q_2 & q_3 & \dots & q_k & w \end{array}$$

$$p_j = \sum_{i=1}^{k} u_{ij}; \quad q_i = \sum_{j=1}^{m} u_{ij}.$$

Суммирование, естественно, выполняется по mod 2.

Можно также вычислить бит $w$, который будет контролировать правильность проверок:

$$w = \sum_{j=1}^{m} p_j \oplus \sum_{i=1}^{k} q_i.$$

**Для матричного кода минимальное кодовое расстояние равно произведению кодовых расстояний применяемых кодов.**

В рассматриваемом случае при проверке на четность строки или столбца $d_{\min} = 2$, поэтому для всего матричного кода $d_{\min} = 4$, что обеспечивает обнаружение трехкратной ошибки и исправление однократной. В последнем случае в процессе декодирования не совпадут одна проверка на четность по строке и одна по столбцу — на их пересечении и располагается ошибочный бит.

**Пример 6.4.**

Матричный код строится из набора информационных байт и включает 6 строк. Локализовать ошибку передачи, установить избыточность кода.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | **1** |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | **0** |
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | **1** |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | **1** |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | **0** |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | **0** |
| **1** | **0** | **1** | **0** | **0** | **0** | **1** | **0** | |

Проверяя строки и столбцы на четность, легко установить, что нарушение четности имеется в строке 5 и столбце 3, следовательно, элемент, стоящий на их пересечении, должен быть инвертирован.

Что касается избыточности, то передаваемых информационных бит 48, а проверочных — 14, следовательно, $F = 0{,}29$.

Относительно описанного метода помехоустойчивого кодирования необходимо сделать ряд замечаний:

1. Метод был продемонстрирован на примере, в котором использовалась проверка на четность — самый простой вариант контроля. Ничего не мешает применить какую-то другую схему (например, систематический код с $r$ проверочными битами) — в этом случае можно сконструировать код с весьма высоким значением $d_{\min}$.

2. Увеличивая длину информационной части, можно заметно понизить избыточность кода, например, если передавать 2 байта информации группами по 16 строк, то избыточность составит 0,125. Вообще наименьшей избыточность оказывается в том случае, если матрица является квадратной.

3. Недостатком матричного кода являются затраты времени на формирование матрицы, что вносит задержку в передачу.

## Контрольные вопросы и задания к гл. 6

1. В чем состоит основная идея построения помехоустойчивого кода?

2. Какими параметрами характеризуется помехоустойчивый код?

3. Что такое «модель ошибок»? Почему от нее зависит метод построения помехоустойчивого кода?

4. В чем состоит идея описания помехоустойчивого кода, предложенная Р. Хеммингом?

5. Каково соотношение понятий «вес кодовой комбинации» и «кодовое расстояние»?

6. Прокомментируйте связь $d_{\min}$ и кратности ошибки $\theta$ для кодов обнаруживающих и исправляющих однократную ошибку.

7. Постройте канонический систематический код (6, 3), исправляющий однократную ошибку, схему кодера к нему.

8. Что такое «синдром ошибки»? Каким образом строится таблица исправлений?

9. Для кода из задания 7 постройте таблицу исправлений и схему его декодера.

10. Напишите программу, демонстрирующую построение помехоустойчивого кода и его декодирования.

11. Каковы идеи построения кода Хемминга?

12. Постройте код Хемминга для двухбайтового первичного кода. Продемонстрирйте локализацию и исправление ошибки.

13. Постройте код Хемминга для однохбайтового первичного кода. Продемонстрирйте локализацию и исправление ошибки с помощью матричных операций.

14. Напишите программу, демонстрирующую построение помехоустойчивого кода Хемминга и его декодирования.

15. Определите кратность обнаруживаемой и исправляемой ошибки матричного кода 8×8.

16. Постройте матричный код 8×8. Продемонстрируйте локализацию и исправление ошибки.

# 7 Элементы криптографии

В данном разделе продолжится рассмотрение мер обеспечения сохранности информации, но совсем от иных угроз, нежели шумы в канале связи. Речь пойдет о криптографической защите, призванной исключить возможность несанкционированного доступа к содержанию передаваемого или хранимого сообщения. Ситуация моделируется следующим образом: некто является источником (или владельцем) конфиденциальной информации, которая предназначена для получения и использования только определенным кругом лиц. Безусловно, для передачи такой информации можно использовать специализированные и закрытые от общего доступа каналы связи — именно так организуется связь на уровне государственных руководителей, военных и т. п. Ясно, что создание такого канала и его обслуживание является весьма дорогостоящим. В то же время организаций и лиц, которым требуется ограничить доступ к собственной информации, очень и очень много — это экономическая, техническая, политическая информация и даже персональная информация отдельных людей. Безусловно, обеспечить всех закрытыми каналами связи невозможно.

С весьма давних времен проблема решается путем *шифрования*, т. е. такого преобразования исходного текста, которое затруднило бы его прочтение для лиц, не знающих шифра. Но это, в свою очередь, означает, что такие лица существуют — можно назвать их излишне любопытными, злоумышленниками, противниками или еще как-то — их современное название (которое и мы будем использовать далее) — *криптоаналитик*. В силу каких-то причин он решает задачу прочтения шифрованного сообщения. Таким образом, осуществляется противостояние и борьба двух начал: разработчик шифра стремится обеспечить его максимальную стойкость и невозможность прочтения шифрованного сообщения, криптоаналитик — по имеющемуся шифрованному сообщению прочесть его.

Хотя проблема секретной связи занимала человека с весьма давних времен, криптография как наука сформировалась только в середине 20-го века в значительной степени благодаря исследованиям К. Шеннона. До появления в 1948 г. его знаменитой работы «Математическая теория связи», с которой, как уже отмечалось, принято вести отсчет информатики как самостоятельной науки, в 1945 г. им написана статья «Теория связи в секретных системах», которая была включена в закрытый доклад Министерства обороны США «Математическая теория криптографии», вскоре после войны рассекреченного. К идеям Шеннона при рассмотрении некоторых методов шифрования мы еще вернемся.

## 7.1. Основные понятия

### 7.1.1. Терминология криптографии

Определим ряд основных понятий данного раздела.

> *Преобразование исходной информации, заключающееся в приведении составляющих ее элементов (слов, букв, цифр) с помощью специальных алгоритмов к виду, не позволяющему воспроизвести исходные данные без знания секрета обратного преобразования (восстановления) или специального ключа называется* ***криптографическим преобразованием****.*

Цель такого преобразования — защита информации от несанкционированного доступа и сокрытия от посторонних лиц, а также обеспечения ее подлинности и целостности.

*Алфавит* — конечное множество знаков, используемых для представления первичной информации.

*Открытый* (исходный) *текст* — упорядоченный набор знаков алфавита, обладающий семантической значимостью (смыслом).

*Шифрованный* (закрытый) *текст* (*криптограмма*) — данные, полученные после применения шифра к открытому тексту.

*Шифр* — алгоритм или однозначные отображения открытого текста в шифрованный.

*Криптографическая система* (*криптосистема*) — множество обратимых преобразований открытого текста в шифрованный.

*Ключ* — некоторый секретный параметр шифра (обычно — последовательность знаков алфавита), позволяющий выбрать

для шифрования только одно конкретное преобразование из всего множества преобразований, составляющих шифр*.

*Криптоанализ* — система методов нарушения конфиденциальности и целостности информации.

*Криптоаналитик* — человек, создающий и применяющий методы криптоанализа.

Криптография и криптоанализ составляют *криптологию* как единую науку о создании и взломе шифров.

*Криптографическая атака* — попытка криптоаналитика вызвать отклонения в атакуемой защищенной системе обмена информацией. Успешную криптографическую атаку называют взломом или вскрытием.

*Шифрование* — процесс нормального применения криптографического преобразования открытого текста на основе алгоритма и ключа, в результате которого возникает шифрованный текст.

*Расшифровывание* — процесс нормального применения криптографического преобразования шифрованного текста в открытый.

*Дешифрование* (дешифровка) — процесс извлечения открытого текста из шифрованного без знания шифра и/или криптографического ключа.

*Криптографическая стойкость* — способность криптографического алгоритма противостоять криптоанализу.

*Шифратор* — техническое устройство, реализующее шифровку и расшифровку информации по установленному алгоритму и ключу.

*Электронная (цифровая) подпись* — присоединяемое к тексту его криптографическое преобразование, которое позволяет при получении текста другим пользователем проверить авторство и целостность сообщения.

*Пароль* — секретная последовательность букв алфавита, используемая для аутентификации субъекта (не для шифрования, как ключ).

---

* В стандарте ГОСТ 28147-89 понятие ключ определено следующим образом: «Конкретное секретное состояние некоторых параметров алгоритма криптографического преобразования, обеспечивающее выбор одного преобразования из совокупности всевозможных для данного алгоритма преобразований».

### 7.1.2. Обзор криптографических методов

*Криптография* (от греч. $\kappa\rho\upsilon\pi\tau o\zeta$ — скрытый и $\gamma\rho\alpha\varphi\omega$ — пишу) — наука о методах обеспечения конфиденциальности (невозможности прочтения информации посторонним) и аутентичности (целостности и подлинности авторства, а также невозможности отказа от авторства) информации.

Изначально криптография изучала способы шифрования информации — обратимого преобразования открытого (исходного) текста на основе секретного алгоритма и/или ключа в шифрованный текст (шифртекст). Традиционная криптография образует раздел симметричных криптосистем, в которых зашифрование и расшифрование проводится с использованием одного и того же секретного ключа. Помимо этого раздела современная криптография включает в себя асимметричные криптосистемы, системы электронной цифровой подписи (ЭЦП), хеш-функции, управление ключами, получение скрытой информации, квантовую криптографию.

Криптографические методы наиболее часто подразделяются в зависимости от количества ключей, используемых в соответствующих криптоалгоритмах (рис. 7.1):

- *бесключевые*, в которых какие-либо ключи не используются;
- *одноключевые* — как следует из названия, в них используется один для всех авторизованных пользователей секретный ключ;

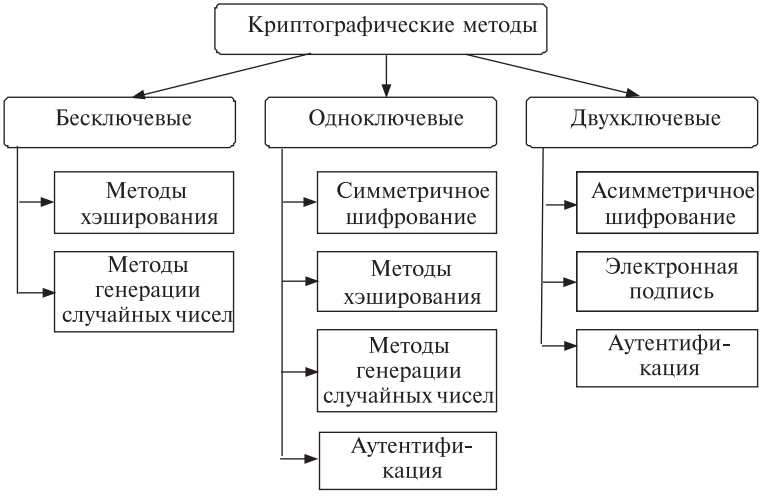



**Рис. 7.1.** Классификация криптографических методов

- *двухключевые*, использующие в своих построениях (алгоритмах) два ключа: секретный и открытый.

Для *шифрования* используются две основные схемы — с закрытым ключом (симметричное шифрования) и с открытым ключом (асимметричное шифрование).

*Электронная подпись* (ЭП) используется для подтверждения целостности и авторства электронного документа. ЭП строится по ассиметричной схеме шифрования; при этом секретный ключ используется автором документа для создания подписи, а открытый ключ — для ее проверки получателем.

*Методы хеширования* (ключевого и бесключевого) по информации произвольного размера с использованием секретного ключа или без него (бесключевое хеширование) вычисляют некую контрольную сумму фиксированного размера (битовую строку), однозначно соответствующую исходным данным. Такое криптографическое контрольное суммирование широко используется в других методах защиты информации — электронной подписи, аутенфикации.

*Аутентификация* позволяет проверить, что пользователь (или удаленный компьютер) действительно является тем, за кого он себя выдает. Простейшей схемой аутентификации является *парольная* — в качестве секретного элемента в ней используется пароль, который предъявляется пользователем при его проверке. Такая схема является для криптоаналитика слабоустойчивой, если для ее усиления не применяются специальные административно-технические меры.

*Генераторы случайных и псевдослучайных чисел* позволяют создавать последовательности случайных чисел, которые используются:

- для генерации секретных ключей, которые, в идеале, должны быть абсолютно случайными;
- во многих алгоритмах электронной подписи;
- во многих схемах аутентификации.

### 7.1.3. Постановка задачи шифрования

Шифрование обеспечивает реализацию трех аспектов безопасности информации:

- *конфиденциальность* — сокрытие информации от неавторизованных пользователей при передаче или при хранении;
- *целостность* — предотвращение изменения информации при передаче или хранении;

- *идентифицируемость* — аутентификации источника информации и предотвращения отказа отправителя информации от того факта, что данные были отправлены именно им.

Дадим математическую постановку вопросов и закономерностей шифрования.

Криптографическую систему можно рассматривать как совокупность следующих компонентов*:

- отдельный открытый текст $m$;
- пространство открытых текстов $M$ (*message*); $m \in M$;
- отдельный шифрованный текст $c$;
- пространство шифрованных текстов (криптограмм, *cipher text*) $C$; c $\in$ C;
- функция зашифрования $E$ (*encryption*), выполняющая криптографические преобразования над $M$;
- $k_1$ (*key*) — параметр функции $E$, называемый ключом зашифрования.

Таким образом, зашифрование может быть описано формулой

$$C = E_{k_1}(M), \tag{7.1}$$

т. е. в результате воздействия функции $E$ с параметром (ключом) $k_1$ на открытый текст из пространства $M$ формируется шифрованный текст $C$.

Как уже отмечалось, ключ может принадлежать определенному лицу или группе лиц и являться для них уникальным. Зашифрованная с применением конкретного ключа информация может быть расшифрована с использованием только этого же ключа или ключа, связанного с ним определенным соотношением.

Аналогичным образом можно представить и расшифрование:

- $M'$ — сообщение, полученное в результате расшифрования,
- $D$ (*decryption*) — функция расшифрования; так же, как и функция зашифрования, выполняет криптографические преобразования над шифротекстом;
- $k_2$ — ключ расшифрования (для двухключевых методов).

Тогда по аналогии с (7.1)

$$M' = D_{k_2}(C). \tag{7.2}$$

---

* Используются обозначения, заимствованные из книги Ж. Брассара Современная криптология (пер. с англ). — М.: ПОЛИМЕД, 1999. — 176 с.

Для получения в результате расшифрования открытого текста, совпадающего с оригиналом ($M' = M$), необходимо одновременное выполнение следующих (достаточно очевидных) условий:

- функция расшифрования должна соответствовать функции зашифрования;
- ключ расшифрования должен соответствовать ключу зашифрования.

При отсутствии верного ключа $k_2$ получить исходное сообщение $M' = M$ даже с помощью правильной функции $D$ невозможно*.

Как следует из рис. 7.1, алгоритмы шифрования можно разделить на две категории.

*Алгоритмы симметричного шифрования.* В них для расшифрования используется тот же самый ключ, что и для зашифрования, или ключ, связанный с ним каким-либо простым соотношением.

*Алгоритмы асимметричного шифрования.* В асимметричном шифровании ключ зашифрования $k_1$ вычисляется из ключа $k_2$ таким образом, что обратное вычисление было бы невозможным. Например, соотношение ключей может быть таким:

$$k_1 = a^{k_2} \bmod p,$$

где $a$ и $p$ — параметры алгоритма шифрования, имеющие достаточно большую размерность; mod — остаток от целочисленного деления.

Подобное соотношение ключей используется, в том числе, в алгоритмах электронной подписи.

С точки зрения передачи зашифрованной информации симметричное шифрование менее выгодно, чем асимметричное из-за того, что адресат должен заранее получить ключ для расшифрования информации. У асимметричного шифрования такой проблемы нет (поскольку открытый ключ можно свободно передавать по сети), однако скорость шифрования значительно ниже, чем у симметричного. Наиболее часто асимметричное шифрование используется в паре с симметричным — для передачи ключа симметричного шифрования, на котором шифруется основной объем данных.

---

* Под словом «невозможно» в данном случае обычно понимается невозможность вычисления за реальное время при существующих вычислительных ресурсах.

Симметричное шифрование бывает двух видов — *блочное* и *поточное.*

При *блочном шифровании* открытый текст предварительно разбивается на равные по длине блоки (например, 64 или 128 бит), после чего блоки поочередно шифруются; в результате получается блок шифрованного текста такой же длины. После этого аналогичному преобразованию подвергается следующий блок данных.

*Поточные шифры* преобразуют открытый текст в шифрованный по одному элементу за операцию (потоком элемент за элементом). Например, биты открытого текста складываются по модулю 2 с битами некоторой превдослучайной последовательности.

Современные симметричные криптосистемы представлены такими широко известными стандартами, как ГОСТ 28147-89 (Россия), DES и Rijndael (США), которые являются блочными шифрами.

## 7.2. Симметричное шифрование

### 7.2.1. Схема криптосистемы с симметричным шифрованием

Одноключевая схема (схема с симметричным шифрованием) представлена на рис. 7.2.

Трактовать схему нужно следующим образом. Источник сообщений создает открытый текст $m$. Генератор ключей порождает ключ шифра $k$, который передается в шифратор и по защищенному каналу (недоступному для вскрытия криптоаналитиком, например, через курьера) в дешифратор. Как будет показано ниже, новый ключ должен создаваться для каждого нового $m$. Метод шифрования определяется функцией $E_k(m)$, которая закладывается в алгоритм работы шифратора с ключом в качестве параметра. В результате после шифратора в основной канал

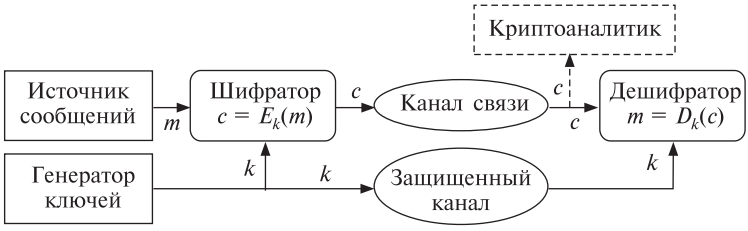



**Рис. 7.2.** Криптосистема с симметричным шифрованием

связи направляется шифрованное сообщение $c$. Следует предположить, что доступ к этому каналу может иметь криптоаналитик и, следовательно, текст $c$ оказывается в его распоряжении.

Алгоритм дешифровки $D_k(c)$, являющийся обратным по отношению к алгоритму прямого преобразования, заложен в дешифратор; при наличии ключа, совпадающего с ключем шифрования, с его помощью восстанавливается исходный текст по криптограмме; без ключа восстановление невозможно даже при согласованных прямом и обратном алгоритмах (методах шифрования).

### 7.2.2. Некоторые методы шифрования

В работе «Методы шифрования»* приводится следующая классификация основных методов криптографического закрытия информации:

1. Подстановка (замена).
2. Перестановка.
3. Гаммирование.
4. Аналитические преобразования.
5. Комбинированные.

Поскольку в данном учебнике не ставится задача систематического изложения криптографии, а лишь общих представлений о ней, ниже будут рассмотрены только некоторые из перечисленных методов.

Любой криптографический метод характеризуется такими показателями, как *стойкость* и *трудоемкость*:

- стойкость метода — это тот минимальный объем зашифрованного текста, статистическим анализом которого можно вскрыть исходный текст; другими словами, стойкость шифра определяет допустимый объем информации, который можно зашифровать при использовании одного ключа;
- трудоемкость метода определяется числом элементарных операций, необходимых для шифрования одного символа исходного текста.

#### 7.2.2.1. Шифрование методом замены (подстановки)

Это один из наиболее простых методов шифрования. Каждый символ исходного текста заменяются другими символами, взятыми из одного алфавита (*одноалфавитная* замена) или нескольких алфавитов (*многоалфавитная* замена).

---

* Методы шифрования. http://protect.htmlweb.ru/p11.htm

**Одноалфавитная замена.** Пусть открытый (исходный) текст представлен в алфавите $A = \{a_1, a_2, \ldots, a_n\}$, а функция преобразования $E_k(m)$ представляет собой сдвиг исходного алфавита на некую константу — значение ключа. В этом случае зашифрованный текст будет представлен в том же алфавите. Примем, что исходный русский алфавит содержит 32 буквы, объединив е и ё, а также ь и ъ; отдельным знаком добавлен пробел «_». Например, $k = 5$. Тогда все буквы нужно сдвинуть вправо (или влево – это вопрос согласования функций $E_k(m)$ и $D_k(c)$) на 5 позиций:

| А | Б | В | Г | Д | Е | Ж | З | И | Й | К | Л | М | Н | О | П | Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ь | Ы | Э | Ю | Я | _ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Е | Ж | З | И | Й | К | Л | М | Н | О | П | Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ь | Ы | Э | Ю | Я | _ | А | Б | В | Г | Д |

В этом варианте слово ИНФОРМАТИКА будет зашифровано как НТЩУХСЕЧНПЕ.

Представленную таблицу будем называть *таблицей замены* (по сути шифровальной таблицей), поскольку именно она отражает соответствие знаков исходного и шифрованного текстов. Строки таблицы удобно рассматривать в качестве двух линеек (наборов) алфавитов.

Безусловно, такой шрифт крайне ненадежен, поскольку, во-первых, изначально ясен язык исходного сообщения и, следовательно, число букв алфавита; даже не зная ключа, можно перебрать все допустимые сдвиги и ключ определить; во-вторых, одни и те же буквы исходного сообщения кодируются одними же буквами зашифрованными — проведя статистический анализ криптотекста (особенно, если он будет достаточно длинным), можно выявить вероятности появления отдельных букв и, сравнив, например, с табл. 2.1, получить какие-то сведения об открытом тексте.

Достоинством такого шифра является простота реализации, а недостатком — низкая криптостойкость. Однако на основе одноалфавитного подхода к шифрованию возможны варианты, заметно затрудняющие работу криптоаналитика.

**Перемешивание алфавита с помощью многократной замены.** В этом варианте используется, по сути, несколько функций $E_k(m)$, каждая из которых определяет свой сдвиг. Ключом будет набор сдвигов. Например, пусть $E^{(1)} = 6$, $E^{(2)} = -3$, $E^{(3)} = -8$, $E^{(4)} = 5$, следовательно, $k = 6(-3)(-8)5$. Это, в свою очередь означает, что текст вновь разбивается на блоки длиной, определяемой количеством функций сдвига (в рассматриваемом

примере 4); в пределах блока первый знак сдвигается по линейке алфавита на 6 позиций влево, второй — на 3 позиции вправо, третий — на 8 вправо, четвертый — на 5 влево. Таблица замены помимо исходного алфавита будет содержать четыре линейки шифрованных алфавитов:

| А | Б | В | Г | Д | Е | Ж | З | И | Й | К | Л | М | Н | О | П | Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ь | Ы | Э | Ю | Я | _ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ж | З | И | Й | К | Л | М | Н | О | П | Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ь | Ы | Э | Ю | Я | _ | А | Б | В | Г | Д | Е |
| Ю | Я | _ | А | Б | В | Г | Д | Е | Ж | З | И | Й | К | Л | М | Н | О | П | Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ь | Ы | Э |
| Ш | Щ | Ь | Ы | Э | Ю | Я | _ | А | Б | В | Г | Д | Е | Ж | З | И | Й | К | Л | М | Н | О | П | Р | С | Т | У | Ф | Х | Ц | Ч |
| Е | Ж | З | И | Й | К | Л | М | Н | О | П | Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ь | Ы | Э | Ю | Я | _ | А | Б | В | Г | Д |

Шифруем текст: ИНФО РМАТ ИКА_.

Буква И ищется по первой линейке (со сдвигом 6) — ей соответствует О;

Н — по второй (со сдвигом −3) — соответствующая буква К;

Ф — по третьей (со сдвигом −8) — буква М;

О — по четвертой (со сдвигом 5) — буква У;

Р — вновь по первой — буква Ц и т. д.

Окончательно получаем ОКМУЦТШЧОЗШЬ.

Дополнительным достоинством данного метода шифрования является «смазывание» статистических закономерностей появления букв в исходном тексте.

**Двухалфавитная подстановка** также представляется таблицей замены с двумя линейками, только в алфавите шифрованного текста указывается произвольный набор знаков, не соответствующий исходному алфавиту. Примером могут служить «пляшущие человечики» из одноименного рассказа А. Конан Дойля (в нем, кстати, содержится и поучительное описание процесса расшифровки).

Пусть мы построили следующую таблицу замены:

| А | Б | В | Г | Д | Е | Ж | З | И | Й | К | Л | М | Н | О | П | Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ь | Ы | Э | Ю | Я | _ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | Ж | Z | N | V | L | П | # | F | S | Б | J | % | 9 | Э | W | @ | Q | 5 | $ | + | Y | 3 | 4 | G | = | 2 | С | Ы | Н | 8 | U |

Тогда ИНФОРМАТИКА будет зашифрована следующим образом: F9+Э@ %65FБ6

Безусловно, нестандартный алфавит затрудняет для криптоаналитика выявление алфавита исходного текста, однако в шифрованном сообщении сохраняются статистические закономерности его языка.

**Метод ключа Вижинера и его варианты.** В шифре Вижинера ключом служит набор из нескольких неповторяющихся букв. Буквы исходного алфавита нумеруются от 0 до $N-1$ (но возможны и другие варианты нумерации, например в соответствии с кодами ASCII). В соответствии с этими номерами получают номера и буквы ключа. Для русского алфавита:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| А | Б | В | Г | Д | Е | Ж | З | И | Й | К | Л | М | Н | О | П | Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ь | Ы | Э | Ю | Я | _ |

Далее записывается исходный текст и под ним повторяющееся ключевое слово; пусть в нашем случае ключом будет слово *шифр*.

Шифруем фразу ИНФОРМАТИКА_ ИНТЕРЕСНАЯ_ НАУКА — записываем ее строкой, над которой располагаем соответствующие номера букв:

| 8 | 13 | 20 | 14 | 16 | 12 | 0 | 18 | 8 | 10 | 0 | 31 | 8 | 13 | 18 | 5 | 16 | 5 | 17 | 13 | 0 | 30 | 31 | 13 | 0 | 19 | 10 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| И | Н | Ф | О | Р | М | А | Т | И | К | А | _ | И | Н | Т | Е | Р | Е | С | Н | А | Я | _ | Н | А | У | К | А |
| Ш | И | Ф | Р | Ш | И | Ф | Р | Ш | И | Ф | Р | Ш | И | Ф | Р | Ш | И | Ф | Р | Ш | И | Ф | Р | Ш | И | Ф | Р |
| 24 | 8 | 20 | 16 | 24 | 8 | 20 | 16 | 24 | 8 | 20 | 16 | 24 | 8 | 20 | 16 | 24 | 8 | 20 | 16 | 24 | 8 | 20 | 16 | 24 | 8 | 20 | 16 |
| 1 | 21 | 9 | 30 | 9 | 20 | 20 | 3 | 1 | 18 | 20 | 16 | 1 | 21 | 7 | 0 | 9 | 13 | 6 | 29 | 24 | 7 | 20 | 29 | 24 | 27 | 30 | 16 |
| Б | Х | Й | Я | Й | Ф | Ф | Г | Б | Т | Ф | Р | Б | Х | З | А | Й | Н | Ж | Ю | Ш | З | Ф | Ю | Ш | Ы | Я | Р |

Далее суммируются номера букв текста и шифра и в случае превышения 31 складываются по модулю* 31 — получается новое значение $Z$, которое и заносится в таблицу:

$$Z_i = (z_{a_i} + z_{s_i}) \bmod 31,$$

где $z_{\text{a}}$ и $z_s$ — номера букв в одной колонке текста и шифра. Например, в первой колонке «И + Ш», т. е. $Z = (8 + 24) \bmod 31 = 1$, что соответствует номеру буквы Б. Можно сначала заполнить строку с новыми номерами, а затем по ней — криптограмму.

---

* Результат сложения $(a + b) \bmod c$ равен остатку от целочисленного деления суммы $a + b$ на $c$.

Вариантом шрифта Вижинера можно считать шифры Бофора, в котором

$$Z_i = (z_{s_i} - z_{a_i}) \bmod 31 \quad \text{или} \quad Z_i = (z_{a_i} - z_{s_i}) \bmod 31.$$

Еще одним вариантом метода Вижинера является *шифр с автоключом*, в котором в качестве ключа используется само сообщение. Ключ, задающий начальный сдвиг текста, используется в этом случае один раз:

| | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| И | Н | Ф | О | Р | М | А | Т | И | К | А | _ | И | Н | Т | Е | Р | Е | С | Н | А | Я | _ | Н | А | У | К | А |
| Ш | И | Ф | Р | И | Н | Ф | О | Р | М | А | Т | И | К | А | _ | И | Н | Т | Е | Р | Е | С | Н | А | Я | _ | Н |

Криптостойкость шифра Вижинера определяется двумя обстоятельствами: во-первых, наблюдается эффект рассеивания статистических свойств исходного текста; во-вторых, длиной ключевого слова. Недостатком является необходимость хранения и передачи ключевого слова. Если оно достаточно длинное, то его трудно запомнить, а если его где-то хранить, то оно может быть украдено.

По современным стандартам шифрование по методу Вижинера не считается хорошо защищенным; основным же вкладом в теорию шифрования является открытие того, что неповторяющиеся ключевые последовательности могут быть образованы с использованием либо самих сообщений, либо функций от сообщений.

#### 7.2.2.2. Шифрование методом перестановки

**Перестановка с фиксированным периодом.** Исходный текст разбивается на блоки оговоренной длины $d$ и в пределах каждого из них осуществляется перестановка букв в соответствии с ключом. Например, пусть $d = 5$, а ключ $k = 31452$. В шифруемом слове информатика 11 букв, следовательно, оно должно быть дополнено незначащими буквами в начале или конце слова (также вопрос согласования алгоритмов шифрования и дешифрования) до целого числа блоков: ИНФОР МАТИК АСОЩЬ; результат перестановки в блоках: ФИОРН ТМИКА ОАЩЬС — после объединения ФИОРНТМИКАЩАЩЬС. Ясно, что в этом варианте кодирования количество знаков в исходном и шифрованном текстах, вообще говоря, могут не совпадать. Перестановку можно осложнить вторым ключом $k_2$, который будет устанавливать порядок транспозиции блоков. Например, $k_2 = 231$ тогда после шифрования получим ТМИКАОАЩЬСФИОРН.

**Блочная перестановка.** Выбирается размер блока шифрования в $n$ столбцов и $m$ строк; ключ формируется из натурального ряда чисел 1,2,...,n с помощью их случайной перестановки.

Шифрование проводится в следующем порядке:

1. Строится заготовка матрицы $n \times m$, цифрами ключа обозначаются номера колонок.

2. Шифруемый текст записывается последовательными строками по $m$ знаков в строку. Размер текста не ограничен.

3. Зашифрованный текст выписывается колонками в порядке возрастания их номеров согласно ключу.

Зашифруем текст ИНФОРМАТИКА_ ИНТЕРЕСНАЯ_НАУКА блоком шифрования длиной $n = 6$. Перед этим необходимо случайным образом сгенерировать ключ и неповторяющихся цифр от 1 до 6; пусть это будет $k = 463152$. Размещаем исходный текст блоками по 6 знаков последовательно в строки матрицы (две буквы в конце последней строки пришлось добавить для завершения блока):

| 4 | 6 | 3 | 1 | 5 | 2 |
|---|---|---|---|---|---|
| И | Н | Ф | О | Р | М |
| А | Т | И | К | А | _ |
| И | Н | Т | Е | Р | Е |
| С | Н | А | Я | _ | Н |
| А | У | К | А | Д | А |

Зашифрованный тест формируется из колонок матрицы, выбираемых в порядке, задаваемом ключом; получаем:

ИАИСАНТННУФИТАКОКЕЯАРАР_ДМ_ЕНА

Имеются и другие варианты шифрования методом перестановки, например усложненная по таблице, усложненная по маршрутам.

Крипкостойкость простой перестановки однозначно определяется размерами используемой матрицы перестановки. Например, при использовании матрицы 16×16 число возможных перестановок достигает $1{,}4^{26}$. Стойкость усложненных перестановок еще выше. Однако следует иметь в виду, что при шифровании перестановкой полностью сохраняются вероятностные характеристики исходного текста, что облегчает криптоанализ.

### 7.2.2.3. Метод гаммирования

Суть метода состоит в том, что на коды символов шифруемого текста накладываются коды случайной последовательности чисел, которую называют также *гаммой* (отсюда название метода — «*гаммирование*»).

Наиболее просто продемонстрировать суть метода на двоичных кодах, в которые компьютером переводятся все знаки алфавитов исходных сообщений, например коды ASCII. Посредством генератора ключей (рис. 7.2), содержащего датчик случайных (на практике — псевдослучайных) чисел, формируется последовательность 0 и 1, длина которой совпадает с числом бит исходного текста. Далее выполняется побитное сложение $\oplus$ (по модулю 2 — XOR — исключающее ИЛИ) битов исходного текста и гаммы. Поскольку гамма является случайной последовательностью, результат ее наложения также будет абсолютно случаен и, следовательно, не нести никакой информации об открытом тексте.

Ясно, что для расшифровки криптотекста в дешифраторе должно быть повторно произведено его побитное суммирование $\oplus$ с битами гаммы.

**Пример 7.1.**

Требуется зашифровать методом гаммирования текст КРИПТО.

Примем за основу числовые значения знаков алфавита открытого текста, использованные ранее; числа представим в 16-ричном виде:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| А | Б | В | Г | Д | Е | Ж | З | И | Й | К | Л | М | Н | О | П | Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ь | Ы | Э | Ю | Я | _ |

КРИПТО: 0A 10 08 0F 12 0E

Представляем в двоичном виде, добавляем гамму — случайную последовательность двоичных чисел и суммируем $\oplus$ побитно.

Исх. текст: 0000 1010 0001 0000 0000 1000 0000 1111 0001 0010 0000 1110

Гамма: 0101 0011 1001 0100 0100 0110 1011 0110 1111 1001 1000 0101

Шифр. текст: 0101 1001 1000 0100 0100 1110 1011 1001 1110 1011 1000 1011

В 16-ричном представлении: 59 84 4E B9 EB 8B

Можно произвести расшифровку полученного текста, сложив его с гаммой:

Шифр. текст: 0101 1001 1000 0100 0100 1110 1011 1001 1110 1011 1000 1011

Гамма: 0101 0011 1001 0100 0100 0110 1011 0110 1111 1001 1000 0101

Расшифр.: 0000 1010 0001 0000 0000 1000 0000 1111 0001 0010 0000 1110

0 A 1 0 0 8 0 F 1 2 0 E

Безусловно, вместо использованного числового представления знаков (как порядковый номер в алфавите) может быть применена иная нумерация, например ASCII-коды.

Стойкость гаммирования определяется следующими требованиями к гамме:

- для каждого сообщения нужно использовать новую гамму (повторное использование гаммы недопустимо);
- для формирования гаммы использовать аппаратные генераторы случайных чисел на основе физических процессов;
- длина гаммы должна быть не меньше длины защищаемого сообщения.

При использовании современных генераторов псевдослучайных чисел реальным становится создание бесконечной гаммы, что, в свою очередь, приводит к бесконечной теоретической стойкости зашифрованного текста.

### 7.2.3. Совершенная стойкость шифра. Требования, предъявляемые к ключам

Основной характеристикой алгоритма шифрования является *криптостойкость* — потенциальная способность шифра противостоять раскрытию. Стойкий шифр должен удовлетворять требованиям:

- количество ключей должно быть настолько большим, чтобы перебор всех возможных преобразований $E_k$ был бы невозможным;
- по криптограмме $c = E_k(m)$ было бы невозможно (очень трудно) определить ключ $k$ и/или исходный текст $m$;
- правило Кирхгофа: известность криптоаналитику алгоритма преобразования не должна снижать надежность системы шифрования, а ее криптостойкость определяется только секретностью и качеством используемых криптографических ключей.

Таким образом, без знания секретного ключа расшифрование должно быть практически невыполнимым, даже при известном алгоритме шифрования.

Вопросы криптостойкость рассмотрел К. Шеннон в умомянутой выше работе «Теория связи в секретных системах». Им сформулирован ряд требований к системам шифрования повышенной стойкости. Для иллюстрации принципиальной возмож-

ности решения этой проблемы* введем в рассмотрение наряду с множествами $M$ открытых сообщений и $C$ шифротекстов множество $K$ всех возможных ключей. Будем полагать, что криптосистема находится в условиях атаки только шифрованного текста, т. е. в ситуации, когда криптоаналитик располагает только рядом различных криптограмм, на основании которых он должен восстановить примененный ключ $k$ и, тем самым, выявить исходный текст. С позиций теории информации данная задача имеет решение (т. е. текст может быть раскрыт), если средняя взаимная информация ($I(M,C)$) между множествами открытых текстов $M$ и криптограмм $C$ положительна — криптограмма содержит в себе информацию относительно исходного текста. Следовательно, для защищенности от несанкционированного доступа, необходимо обеспечить выполнение условия $I(M,C)=0$. Это, в свою очередь, означает отсутствие взаимной информации между любой возможной парой конкретных $m_i$ и $c_j$, условием чего, как было показано при рассмотрении свойств информации (п. 2.2, замечание 3) является равенство всех безусловных (априорных) и соответствующих им условных (апостериорных) вероятностей:

$$p(m_i) = p_{c_j}(m_i) \neq 0 \quad \text{и} \quad p(c_j) = p_{m_i}(c_j) \neq 0.$$

Другими словами, алгоритм шифрования должен быть таким, чтобы он обеспечивал статистическую независимость открытых текстов и криптограмм. В этом случае говорят, что система обладает *совершенной секретностью*, а соответствующий ее шифр обладает *совершенной стойкостью*.

Можно показать, что необходимым условием совершенной секретности является следующее требование:

$$N_K \geqslant N_M,$$

где $N_M$ — общее число возможных открытых текстов (сообщений); $N_K$ — количество потенциальных ключей. Таким образом, для системы с совершенной секретностью характерно следующее: если криптоаналитик перехватил криптограмму $c_j$, то дополнительной информации, которая облегчила бы ему дешифровку сообщений, он не получит.

Если же в системе шифрования число сообщений $N_M$, число ключей $N_K$ и число шифротекстов $N_C$ совпадает, т. е. $N_M =$

---

* Сам К. Шеннон обозначал ситуацию как «*теоретическая секретность*», подчеркивая, по-видимому, ее принципиальную возможность, а не прикладное значение и практическую возможность реализации.

$= N_K = N_C = N$, то система имеет совершенную секретность тогда и только тогда, когда:

- все ключи выбираются *равновероятно* и *независимо* от открытого текста:

$$p(k) = p_m(k) = \frac{1}{N};$$

- любой фиксированный открытый текст преобразуется разными ключами $k_1$ и $k_2$ в различные криптограммы $c_1$ и $c_2$, т. е.

$$c_1 = E_{k_1}(m) \neq c_2 = E_{k_2}(m); \quad k_1 \neq k_2 \quad \forall m \in M.$$

При невыполнении этих условий будет существовать сообщение $m_i$, у которого для данного $c_j$ не существует ключа дешифрации $c_j$ в $m_i$. Отсюда следует, что для некоторых пар $i$ и $j$ условная вероятность окажется равной нулю:

$$p_{y_j}(x_i) = 0.$$

В этом случае криптоаналитик может исключить из рассмотрения определенные нешифрованные сообщения, упростив, таким образом, свою задачу.

**Пример 7.2.**

Пусть рассматривается схема шифрования, в которой имеется по три исходных текстов, зашифрованных текстов и ключей с равной вероятностью появления, т. е.

$$N_m = N_c = N_k = N = 3,$$

т. е. $M = \{m_0, m_1, m_2,\}$, $C = \{c_0, c_1, c_2,\}$ и $K = \{k_0, k_1, k_2,\}$.

Пусть вероятности появления любого из исходных текстов и криптограмм одинаковы и равны:

$$p(m_i) = p(c_j) = \frac{1}{3}; \quad i, j = 0, 1, 2.$$

Преобразование $m$ в $c$ выполняется по правилу

$$c_s = E_{k_j}(m_i),$$

где $s = (i + j) \bmod N$.

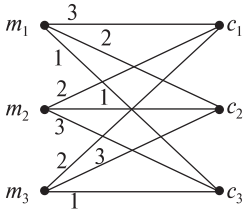



**Рис. 7.3.** Соответствие между множествами $M$ и $C$

Смысл последнего выражения в том, что при чисто случайном выборе $i$ и $j$ оказывается, что и $s$ не превышает значение 2 и также случайно.

Соответствие между множествами $M$ и $C$ иллюстрировано на рис. 7.3 — каждое исходное сообщение $m_i$ может посредством некоторого ключа быть переведенным в криптограмму $c_j$. Если выбор ключа в каждом конкретном случае осуществляется случайным образом, то априорная и апостериорная вероятности для каждой пары $m_i \leftrightarrow c_j$ оказываются равны

$$p(c_j) = p_{m_i}(c_j) = \frac{1}{3}.$$

Криптоаналитик, перехвативший один из шифротекстов $c_s$, не сможет определить, какой из трех ключей использовался и, следовательно, которое из $m_i$ является верным. Таким образом, схема шифрования обладает совершенной секретностью.

## 7.3. Шифрование с открытым ключом

### 7.3.1. Общее представление об ассиметричной криптосистеме

Сложным местом практической реализации любой криптосистемы оказывается проблема распространения ключей. Для того чтобы был возможен обмен конфиденциальной информацией между двумя субъектами, ключ должен быть сгенерирован одним из них, а затем каким-то образом, опять же в конфиденциальном порядке, передан другому. В симметричной криптографической системе (см. рис. 7.2) это достигается за счет использования закрытого (недоступного для криптоаналитика) канала связи. При этом возникает ряд проблем:

*во-первых*, как показал проведенный выше анализ, для обеспечения криптостойкости шифра должно существовать множество ключей (не меньше числа исходных сообщений) и в каждом акте шифрования ключ должен выбираться случайным образом; это, в свою очередь, означает, что при передаче сообщения по открытому каналу нужно параллельно передавать ключ по закрытому; использование одного ключа при передаче многих сообщений упрощает работу криптоаналитика;

*во-вторых*, ключ должен быть достаточно длинным (содержать много знаков), например в методе гаммирования длина ключа (гаммы) совпадает с длиной сообщения; в этом случае становится непонятным, зачем сообщение передавать по открытому каналу и почему бы не передать его сразу по закрытому?

Решением проблемы распространения ключей стала *асимметричная* криптосистема, или система *с открытым ключом*. Суть состоит в том, что получателем конечной (шифрованной) информации генерируются два ключа, связанные между собой по определенному правилу. Один ключ открытый (*public key*) и может быть передан любому, кто желает послать сообщение адресату. Другой ключ секретный (*private key*) хранится только у владельца. Отправитель зашифровывает сообщение открытым ключом получателя и пересылает ему. При этом *зашифрованный текст в принципе не может быть расшифрован тем*

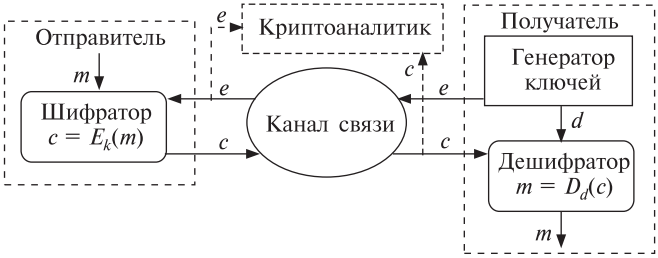



**Рис. 7.4.** Структура криптосистемы с открытым ключом

*же открытым ключом.* Расшифрование сообщения возможно только с использованием секретного ключа, который известен лишь самому адресату.

Значимыми оказываются следующие преимущества асимметричных шифров перед симметричными:

- не нужно предварительно передавать секретный ключ по специальному закрытому каналу и, следовательно, такой канал не нужен;
- только одной стороне известен ключ расшифрования, который он держит в секрете;
- пару ключей можно не менять значительное время;
- в больших сетях число ключей в асимметричной криптосистеме значительно меньше, чем в симметричной;
- становится возможной связь абонента со многими отправителями, которые могут свободно получить открытый ключ из общедоступной базы данных, например банка ключей.

Работа криптосистемы с открытым ключом иллюстрируется рис. 7.4.

Получатель генерирует два ключа — открытый и общедоступный $e$ и секретный $d$, предназначенный только для дешифратора получателя. Безусловно, алгоритмы (функции) прямого и обратного преобразования ($E_e$ и $D_d$) согласовываются между собой; если шифрование и расшифрование выполняются некими техническими устройствами, то эти алгоритмы могут быть неизвестны ни отправителю, ни получателю, а заложены в программу работы устройств. Шифратор отправителя посредством функции $E_e$ с открытым ключом $e$ переводит исходный текст в шифрованный. Криптоаналитику, вообще говоря, доступна информация из открытого канала связи, т. е. шифрованное сообщение $c$ и открытый ключ $e$. Однако характер функции $E_e$ таков, что по $c$ и $e$ невозможно восстановить $m$. Расшифровка возмож-

на только функцией $D_d$, для использования которой необходим закрытый ключ $d$.

В криптографических системах с открытым ключом используются так называемые *необратимые*, или *односторонние* функции. Функция $y = f(x)$ называется односторонней, если для вычисления $y$ по $x$ существует алгоритм *полиномиальной сложности*, а для определения $x$ по $y$, т. е. $x = f^{-1}(y)$ известны только алгоритмы *экспоненциальной сложности* (см. п. 9.7). Полиномиальным называется такой алгоритм вскрытия ключа, у которого число необходимых операций пропорционально $n^a$, где $n$ — число неизвестных параметров, $a$ — некоторое целое. Алгоритмом вскрытия экспоненциальной сложности называется такой алгоритм, у которых число необходимых операций пропорционально $a^n$. Иначе говоря, найти $y$ по $x$ легко, а $x$ по $y$ трудно (при $n = 60$ выполнение алгоритма сложности $n^3$ требует доли секунды, $3^n$ — около $10^{15}$ лет).

В нашем случае это означает, что для каждой пары функций $(E, D)$ имеется свойство: зная $E_e$, невозможно решить уравнение $E_e(m) = c$, т. е. по данному шифротексту $c$ невозможно восстановить исходное сообщение $m$. Из этого, в свою очередь, следует, что по данному $e$ невозможно определить соответствующий ключ расшифрования $d$. $E_e$ является односторонней функцией.

Таким образом, основой алгоритмов, используемых в ассиметричных системах, являются необратимые преобразования. При этом под необратимостью понимается не теоретическая необратимость, а *практическая невозможность* вычислить обратное значение, используя современные вычислительные средства за приемлемый интервал времени.

Поэтому для гарантии надежной защиты информации к системам с открытым ключом предъявляются два важных (но достаточно очевидных) требования:

- преобразование исходного текста должно быть необратимым и исключать возможности его восстановления на основе открытого ключа;
- определение секретного ключа на основе открытого также должно быть невозможным на современном технологическом уровне.

Используемые в настоящее время криптосистемы с открытым ключом опираются на один из следующих типов необратимых преобразований:

- вычисление логарифма в конечном поле;

- вычисление корней алгебраических уравнений;
- разложение больших чисел на простые множители.

К недостаткам криптосистем с открытым ключом следует отнести:

- шифрование-расшифрование проходит на два-три порядка медленнее, чем обработка того же текста симметричным алгоритмом;
- требуются существенно бомльшие вычислительные ресурсы, поэтому на практике асимметричные криптосистемы используются в сочетании с другими алгоритмами.

### 7.3.2. Формирование ключей и шифрование в криптосистеме RSA

Одной из наиболее популярных является криптосистема с открытым ключом RSA, разработанная в 1977 году и получившая название в честь ее создателей — Райвест, Шамир, Адлеман (Rivest–Shamir–Adleman). Их алгоритм был основан на том, что нахождение больших простых чисел в вычислительном отношении осуществляется легко, но разложение на множители произведения двух таких чисел практически невыполнимо. Доказано, что раскрытие шифра RSA эквивалентно такому разложению. Высокая защищенность алгоритма RSA стала одной из причин его использования, в частности в банковских компьютерных сетях, особенно для работы с удаленными клиентами (обслуживание кредитных карточек).

Рассмотрим реализацию алгоритма RSA, предварив ее рядом определений:

- под *простым* числом понимается такое число, которое делится только на единицу и на само себя;
- *взаимно простыми* называются такие числа, которые не имеют ни одного общего делителя, кроме единицы;
- под результатом операции $A \bmod B$ понимается остаток от целочисленного деления $A$ на $B$ ($48 \bmod 9 = 3$).

При проверке числа на простоту можно воспользоваться малой теоремой Ферма: число $H$ является простым, если для любого $G < H$ справедливо соотношение

$$G^{H-1} \bmod H = 1.$$

Рассмотрим ситуацию, когда имеется один получатель шифрованной информации и, вообще говоря, много отправителей.

Для подготовки ключей получатель производит следующие операции*:

1. Выбираются два больших простых числа $q$ и $p$ (например, по 100 десятичных разрядов), находится их произведение $n = pq$; число $n$ будет ограничивать сверху допустимый объем передаваемого шифрованного фрагмента (например, при размере $q$ и $p$ по 100 десятичных разрядов, $n$ будет содержать 200 десятичных разрядов и длина шифруемого фрагмента составит около 660 бит, поскольку один десятичный разряд соответствует $\log_2 10 = 3{,}32$ двоичным).

2. Вычисляется функция Эйлера $\Phi(n) = (p-1)(q-1)$.

3. Выбирается произвольное достаточно большое число $e$, взаимно простое с $\Phi(n)$, — оно будет принято в качестве открытого ключа.

4. Выбирается произвольное $d$, удовлетворяющее уравнению $ed \bmod \Phi(n) = 1$ и условию и $d < \Phi(n)$ — это будет значение секретного ключа.

Таким образом, получатель располагает набором чисел: $q$, $p$, $n$, $\Phi(n)$, $e$ и $d$. В общий доступ помещаются числа $\{e, n\}$ — они составляют открытый ключ и позволяют осуществлять шифрование. Однако при знании $\{e, n\}$ и любого из чисел $q, p, \Phi(n)$ можно вычислить секретный ключ $\{d, n\}$ — по указанной причине эти числа должны быть недоступны криптоаналитику или вообще быть уничтожены, поскольку для дальнейшей работы не нужны. На этом заканчивается подготовительная работа, связанная с генерацией ключей.

Шифрование выполняется следующим образом:

1. Знаки исходного сообщения $(m_i)$ каким-нибудь способом кодируются цифрами $(m_i \to w_i)$; зашифрованное сообщение также будет представлено в цифровом виде $(c_i \to u_i)$; цифровые коды должны удовлетворять условию $2 \leqslant w_i \leqslant n-2$ (т. е. исключаются числа 1 и $n-1$).

2. Функцией, используемой при шифровании открытого текста, является

$$u_i = (w_i)^e \bmod n.$$

Такая функция считается односторонней, поскольку неизвестны способы, позволяющие по значению $u_i$ для достаточно больших $e$ и $n$ вычислить $w_i$, т. е. восстановить исходное сообщение.

---

* Обоснование алгоритма приводится в книге Р.Д. Аветисян, Д.О. Аветисян «Теоретические основы информатики» (с. 79–82).

Расшифровка криптограммы с помощью секретного ключа выполняется в соответствии с правилом

$$w_i = (u_i)^d \bmod n.$$

**Пример 7.3**.

Требуется зашифровать сообщение «РЕКА» алгоритмом RSA.

*Шаг 1.* Выбираем два произвольных простых числа; в нашем примере для простоты будем использовать маленькие числа; пусть $p = 11$ и $q = 3$, т. е. $n = 11 \cdot 3 = 33$.

*Шаг 2.* Находим функцию Эйлера: $\Phi(33) = (11-1)(3-1) = 20$.

*Шаг 3.* Выбираем $e$ взаимно простое с $\Phi(27)$, например 7. Открытым ключом будет $\{7, 33\}$.

*Шаг 4.* Из условия $ed \bmod \Phi(n) = 1$ выбираем $d$: $7d \bmod 20 = 1$; решениями этого уравнения могут быть 3, 23, 43 и т. д.; конечно, мы выберем наименьшее $d = 3$ (исключительно из соображений простоты дальнейших расчетов); тогда секретным оказывается ключ $\{3, 33\}$.

Перед шифрованием, как указывалось, нужно знакам открытого текста поставить в соответствие числа; пусть в нашем примере А $\to$ 3, К $\to$ 6, Р $\to$ 7, У $\to$ 4; в числовом формате представления исходное сообщение приобретает вид (7, 4, 6, 3); шифруем его с помощью ключа $\{7, 33\}$:

$$u_1 = 7^7 \bmod 33 = 823543 \bmod 33 = 28;$$
$$u_2 = 4^7 \bmod 33 = 16384 \bmod 33 = 16;$$
$$u_3 = 6^7 \bmod 33 = 279936 \bmod 33 = 30;$$
$$u_4 = 3^7 \bmod 33 = 2187 \bmod 33 = 9.$$

Таким образом, зашифрованный текст выглядит так: (28, 16, 30, 9).

Расшифруем его на основе секретного ключа $\{3, 33\}$:

$$w_1 = 26^3 \bmod 33 = 21952 \bmod 33 = 7;$$
$$w_2 = 16^3 \bmod 33 = 4096 \bmod 33 = 4;$$
$$w_3 = 30^3 \bmod 33 = 27000 \bmod 33 = 6;$$
$$w_4 = 9^3 \bmod 33 = 729 \bmod 33 = 3.$$

Таким образом, открытый текст восстановлен.

Можно также оценить максимальную длину сообщения, которое единовременно может передаваться при таком шифровании: число десятичных разрядов — 2, значит, число двоичных $2 \cdot 3{,}32 = 6$ бит.

Заключительные замечания.

1. Из рассмотренного примера может сложиться впечатление, что, зная $u_i, e$ и $n$, можно восстановить $w_i$ — по $u_i$ и $n$ перебрать какое-то количество возможных чисел, mod от которых

равен $u_i$; после этого вычислить от этих чисел корень степени $e$ и установить целочисленные значения — это и будут возможные $w_i$, соответствующие $u_i$ (в этом легко убедиться, например, для $u_4 = 9$). Ситуация связана с тем, что при рассмотрении примера мы выбрали малые $p, q$ и $e$; при их больших значениях задача восстановления $w_i$ по $u_i, e$ и $n$, как уже отмечалось, является неразрешимой.

Криптостойкость алгоритма RSA основывается на предположении, что исключительно трудно определить секретный ключ по известному, поскольку для этого необходимо решить задачу о существовании делителей целого числа. Для чисел, состоящих из 200 цифр (а именно числа такой длины рекомендуется использовать), традиционные методы требуют выполнения огромного числа операций (около $10^{23}$) — время разложения таких чисел на простые множители на сегодняшний день выходит за пределы современных технологических возможностей.

2. Если в системе несколько получателей, то генерацию ключей производит каждый и, следовательно, у каждого имеется свой секретный ключ.

3. Недостатком алгоритма RSA, как и других, используемых в криптосистемах с открытым ключом, является большой объем вычислительной работы в процессе шифрования-расшифрования, что на порядки снижает скорость по сравнению с закрытыми системами. По этой причине на практике применяются гибридные системы.

4. Важная проблема практической реализации — ключи какой длины следует использовать? В конце 1995 года удалось практически реализовать раскрытие шифра RSA для 500-значного ключа. Для этого с помощью сети Интернет было задействовано 1600 компьютеров.

Сами авторы RSA рекомендуют использовать следующие размеры модуля $n$:

- 768 бит — для частных лиц;
- 1024 бит — для коммерческой информации;
- 2048 бит — для особо секретной информации.

5. В настоящее время алгоритм RSA активно используется как в виде самостоятельных криптографических продуктов, так и в качестве встроенных средств в популярных приложениях. Однако он не единственный, применяемый в криптосистемах с открытым ключом; популярны и значимы с практической точки

зрения алгоритмы Эль–Гамаля, Диффи–Хеллмана, DSA, ГОСТ Р 34.10-2012.

## 7.4. Электронная подпись

### 7.4.1. Общие принципы использования электронной подписи

Необходимость криптографического контроля целостности пересылаемого по открытым сетям электронного документа появилась в связи с задачей идентификации источника и проверки достоверности полученной информации (как ее содержания, так и автора). В основе такого контроля лежит понятие электронной цифровой подписи (ЭЦП, в настоящее время — ЭП). *ЭП — это особый реквизит электронного документа, который позволяет установить отсутствие искажения информации в нем с момента формирования ЭП и подтвердить принадлежность ЭП владельцу.* ЭП представляет собой уникальную последовательность символов, полученную в результате криптографического преобразования данных электронного документа, которая добавляется к исходному документу. ЭП используется в качестве аналога собственноручной подписи.

Применение электронной подписи обеспечивает:

- удостоверение источника документа;
- защиту от изменений документа;
- невозможность отказа от авторства;
- значительное сокращение времени обмена документами;
- усовершенствование и удешевление процедур подготовки, доставки, учета и хранения документов;
- возможность использования единой и персональной ЭП при подготовке различных документов;
- юридическую силу электронным документам наравне с бумажными, подписанными собственноручной подписью.

При построении электронной подписи используются два алгоритма. Первый уже был рассмотрен выше — это криптосистема с открытым ключом; при этом секретный ключ хранится у источника (автора), подписывающего документ; открытый (общедоступный) ключ позволяет получателю(лям) проверить подлинность подписи и сохранность документа.

Использование другого алгоритма — *хеширования* (hashing) — обусловлено тем, что подписываемые документы могут иметь достаточно большой объем и шифровать его полностью

по схеме с открытым ключом оказывается нерационально (выше указывалось, что алгоритмы ассиметричного шифрования относительно медленны и требуют значительных вычислительных ресурсов). Используемое решение состоит в том, что по информации исходного документа с помощью специальных криптографических преобразований посредством односторонних функций — они называются *хеш-функциями* — вычисляют так называемый *хеш* документа. Хеш — это выходная битовая строка фиксированной длины, получаемая после обработки документа независимо от его объема*. Хеш-функция работает таким образом, что обеспечивает связь всех битов исходных данных с битами хеша. В результате изменение хотя бы одного бита переданной информации сказывается на битах хеша.

Далее на основании хеша и секретного ключа формируется электронная подпись, которая передается вместе с документом. Преимущество такого подхода в том, что объем хеша исходного документа во много раз меньше, чем сам документ, а алгоритмы вычисления хеша являются более быстрыми, чем алгоритмы шифрования.

На приемном конце осуществляются две операции: (1) по полученной ЭП и с помощью открытого ключа определяется хеш отправленного документа; (2) информация пришедшего документа обрабатывается той же самой (что у отправителя) хеш-функцией и вычисляется его хеш. В случае совпадения обоих хешей делается заключение о сохранности документа.

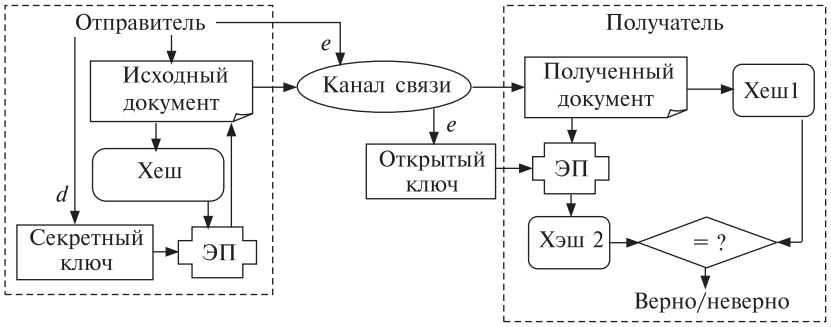



**Рис. 7.5.** Схема использования электронной подписи

* Двумя наиболее распространенными хеш-функциями являются MD5, генерирующая 128-битную контрольную последовательность, и SHA, которая производит последовательность длиной 160 бит.

На рис. 7.5 показана схема взаимодействия отправителя и получатель с применением хеш-преобразований и ЭП.

### 7.4.2. Вычисление и проверка подлинности электронной подписи

Проследим весь цикл создания и использования электронной подписи. Как уже указывалось, генерация ЭП начинается с предварительного хеширования — вычисления некоторой контрольной суммы $S$ от числовых кодов $(w_i)$ всех знаков сообщения. Хеш-функции могут быть различными — мы используем достаточно простую хеш-функцию *квадратичной свертки*, которая позволяет последовательно вычислять значения суммы при обработке $i$-го знака сообщения $(S_i)$ по значению на предыдущем шаге и числовому коду знака $i$:

$$S_i = (S_{i-1} + w_i)^2 \bmod n,$$

где $S_0 = 0$, а $n$ — простое число, входящее в оба ключа*. После обработки последнего символа формируется хеш всего сообщения $S$.

Вычисление ЭП $P$ выполняется по значению $S$ с помощью секретного ключа $d$:

$$P = S^d \bmod n.$$

ЭП присоединяется к исходному сообщению $M$ — образуется новое сообщение $PM$, которое и пересылается.

Процедура проверки подлинности ЭП и самого сообщения после его получения заключается в следующем.

1. От полученного сообщения отделяется ЭП.

2. Открытым ключом отправителя по ЭП находится хеш переданного сообщения по формуле

$$S'' = P^e \bmod n.$$

3. Вычисляется хеш $S'$ полученного сообщения по указанной выше формуле.

4. Сравнивается $S'$ и $S''$; ЭП признается подлинной, если значения хешей совпадают.

Таким образом, с помощью секретного ключа $\{d, n\}$ электронная подпись создается, присоединяется к электронному документу и передается вместе с ним. Открытым ключом $\{e, n\}$ проверяются подлинность подписи и сохранность документа.

---

* Алгоритм генерации ключей может быть принят, например, тем же, что в описанной выше криптосистеме RSA.

**Пример 7.4**.

На основании описанных алгоритмов требуется генерировать электронную подпись на сообщении ИНФОРМАТИКА. Значения ключей возьмем из примера 7.3: $p = 11$, $q = 3$, $n = 33$, $\Phi(33) = 20$, открытый $\{7, 33\}$, секретный $\{3, 33\}$.

Числовые коды знаков исходного текста пусть соответствуют их порядковым номерам в алфавите:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| А | Б | В | Г | Д | Е | Ж | З | И | Й | К | Л | М | Н | О | П | Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ь | Ы | Э | Ю | Я | _ |

Проводим вычисления:

| № | Знак | Код знака $w_i$ | Значения хеш-функции |
|---|---|---|---|
| 0 | | | $S_0 = 0$ |
| 1 | И | 09 | $S_1 = (S_0 + w_1)^2 \bmod n = (0 + 9)^2 \bmod 33 = 15$ |
| 2 | Н | 14 | $S_2 = (S_1 + w_2)^2 \bmod n = (15 + 14)^2 \bmod 33 = 16$ |
| 3 | Ф | 21 | $S_3 = (S_2 + w_3)^2 \bmod n = (16 + 21)^2 \bmod 33 = 16$ |
| 4 | О | 15 | $S_4 = (S_3 + w_4)^2 \bmod n = (16 + 15)^2 \bmod 33 = 4$ |
| 5 | Р | 17 | $S_5 = (S_4 + w_5)^2 \bmod n = (4 + 17)^2 \bmod 33 = 12$ |
| 6 | М | 13 | $S_6 = (S_5 + w_6)^2 \bmod n = (12 + 13)^2 \bmod 33 = 31$ |
| 7 | А | 01 | $S_7 = (S_6 + w_7)^2 \bmod n = (31 + 1)^2 \bmod 33 = 1$ |
| 8 | Т | 19 | $S_8 = (S_7 + w_8)^2 \bmod n = (1 + 19)^2 \bmod 33 = 4$ |
| 9 | И | 09 | $S_9 = (S_8 + w_9)^2 \bmod n = (4 + 9)^2 \bmod 33 = 4$ |
| 10 | К | 11 | $S_{10} = (S_9 + w_{10})^2 \bmod n = (4 + 11)^2 \bmod 33 = 27$ |
| 11 | А | 01 | $S_{11} = (S_{10} + w_{11})^2 \bmod n = (27 + 1)^2 \bmod 33 = 25$ |
| | | Хеш | $S = 25$ |
| | | ЭП | $P = S^d \bmod n = 25^3 \bmod 33 = 16$ |

Таким образом, будет отправлено сообщение с ЭП:

16 09 14 21 15 17 13 01 19 09 11 01

Пусть получателю пришло сообщение

16 09 14 21 15 17 13 01 19 09 11 09

и он желает проверить его истинность. С этой целью осуществляется последовательность операций:

| № | Код знака $w_i$ | Значение хеш-функции | Знак |
|---|---|---|---|
| | Хеш″ | Из ЭП: $S'' = P^e \bmod n = 16^7 \bmod 33 = 25$ | |
| 0 | | $S'_0 = 0$ | |
| 1 | 09 | $S'_1 = (S'_0 + w_1)^2 \bmod n = (0 + 9)^2 \bmod 33 = 15$ | И |
| 2 | 14 | $S'_2 = (S'_1 + w_2)^2 \bmod n = (15 + 14)^2 \bmod 33 = 16$ | Н |
| 3 | 21 | $S'_3 = (S'_2 + w_3)^2 \bmod n = (16 + 21)^2 \bmod 33 = 16$ | Ф |
| 4 | 15 | $S'_4 = (S'_3 + w_4)^2 \bmod n = (16 + 15)^2 \bmod 33 = 4$ | О |

| № | Код знака $w_i$ | Значение хеш-функции | Знак |
|---|---|---|---|
| 5 | 17 | $S'_5 = (S'_4 + w_5)^2 \bmod n = (4 + 17)^2 \bmod 33 = 12$ | Р |
| 6 | 13 | $S'_6 = (S'_5 + w_6)^2 \bmod n = (12 + 13)^2 \bmod 33 = 31$ | М |
| 7 | 01 | $S'_7 = (S'_6 + w_7)^2 \bmod n = (31 + 1)^2 \bmod 33 = 1$ | А |
| 8 | 19 | $S'_8 = (S'_7 + w_8)^2 \bmod n = (1 + 19)^2 \bmod 33 = 4$ | Т |
| 9 | 09 | $S'_9 = (S'_8 + w_9)^2 \bmod n = (4 + 9)^2 \bmod 33 = 4$ | И |
| 10 | 11 | $S'_{10} = (S'_9 + w_{10})^2 \bmod n = (4 + 11)^2 \bmod 33 = 27$ | К |
| 11 | 09 | $S'_{11} = (S'_{10} + w_{11})^2 \bmod n = (27 + 9)^2 \bmod 33 = 9$ | И |
| | Хеш′ | $S' = 9$ | |

Вывод: $S' \neq S''$, следовательно, полученный документ не совпадает с исходным.

В заключении необходимо отметить, что желающему использовать ЭП необходимо получить *сертификат электронной подписи* — документ, который подтверждает принадлежность открытого ключа (ключа проверки) ЭП владельцу сертификата. Выдаются сертификаты удостоверяющими центрами.

Удостоверяющий центр — это юридическое лицо, согласно Закону «Об электронно-цифровой подписи» выполняющее следующие функции:

- генерацию ключевой пары — из набора возможных случайным образом выбирается секретный ключ и вычисляется соответствующий ему открытый ключ; при этом гарантируется сохранение в тайне секретного ключа ЭП;
- выдача по запросу лица или организации, приостановка, возобновление и аннулирование действия идентифицирующего сертификата открытого ключа подписи;
- подтверждение подлинности электронной цифровой подписи в электронном документе в отношении выданных им сертификатов ключей подписей.

Таким образом, владелец получает секретный (закрытый) ключ ЭП, доступ к которому имеет только он сам и который позволяет генерировать электронную подпись и прикреплять ее к электронному документу. Открытый ключ ЭП однозначно связан с секретным ключом и предназначен для проверки подлинности подписи получателями документа.

Сертификат можно рассматривать в качестве дополнительного средства подтверждения подлинности документа и подписи на нем автора.

## Контрольные вопросы и задания к гл. 7

1. В чем схожесть и различие помехоустойчивого кодирования и шифрования сообщения?

2. Предложите алгоритм шифрования-дешифрования методом замены, отличный от тех, что описаны в тексте главы.

3. Предложите алгоритм шифрования-дешифрования с использованием книги (как шифровальной таблицы).

4. Предложите алгоритм шифрования-дешифрования, в котором для получения одного символа шифротекста используются несколько символов исходного текста.

5. Предложите алгоритм шифрования-дешифрования на основе кодов ASCII.

6. Предложите шифр, основанный на преобразовании отдельных битов символов текста, и опишите его алгоритм.

7. Предложите алгоритм дешифрования крипрограммы, полученной методом блочной перестановки, описанным в п. 7.2.2.2.

8. Напишите программу работы шифратора-дешифратора по введенному тексту (криптограмме) и ключу для какого-либо алгоритма замены.

9. Напишите программу работы шифратора-дешифратора по введенному тексту (криптограмме) и ключу для какого-либо алгоритма перестановки.

10. Напишите программу работы шифратора-дешифратора по методу гаммирования; битовая гамма должна формироваться программой случайным образом.

11. Что такое криптостойкость? Какими факторами она определяется?

12. Каким образом из теории информации Шеннона можно получить условие абсолютной криптостойкости?

13. В чем отличия симметричных и асимметричных криптосистем? В чем их достоинства и недостатки?

14. Каков принцип работы криптосистемы с открытым ключом?

15. На конкретном примере продемонстрируйте шифрование текста с помощью алгоритма RSA.

16. В чем достоинства и недостатки применения электронной подписи?

17. Поясните схему организации и порядок использования электронной подписи.

18. Поясните, имеет ли смысл подписывать электронной подписью документ без его предварительного шифрования?

19. Предложите пример создания электронной подписи к сообщению на основании ключей, использованных в задании 15.

20. Проверьте подлинность полученного сообщения и ЭП из задания 19.

# 8 Хранение информации

## 8.1. Классификация данных. Проблемы представления данных

После обсуждения особенностей кодирования и передачи информации вполне естественным представляется рассмотреть вопросы, связанные с хранением информации. Ранее мы дали определение формальной информации как результата последовательности бинарных выборов (см. п. 2.2). Однако на практике (и на бытовом уровне) информация понимается как *сведения* о чем-либо. Эти определения не противоречат друг другу, если мы не будем отслеживать смысловую сторону сведений и пытаться оценивать их важность (значимость), т. е. сохраним формальность подхода.

Наряду и параллельно с термином «*информация*» при описании информационных процессов часто используется термин «*данные*». Определим его следующим образом:

> ***Данные** — это сведения, характеризующие какую-то систему, явление, процесс или объект, представленные в определенной форме и предназначенные для дальнейшего использования.*

К этому определению необходимо сделать следующие замечания, разъясняющие соотношение между понятиями *информация* и *данные*:

- *данные — это конкретная форма представления содержания информации* (например, информацию о результатах наблюдения за температурой окружающей среды можно представить в виде числового массива (таблицы), но можно и в виде графика, и в виде текстового описания посредством некоторого языка);
- в отличие от ненаправленной (неадресной, рассеянной) информации, существующей в природе независимо от нас и на-

ших потребностей в ней, данными называется только такая информация, которая *имеет значение для потребителя* и, следовательно, предусматривается ее использование для решения каких-либо задач; другими словами, практический статус и важность данных выше, нежели у природной информации.

Безусловно, при решении практических задач с помощью технических устройств формы представления информации всегда конкретны и в информации кто-то заинтересован, поэтому употребление термина «*данные*» вполне оправдано.

Содержание понятия «*данные*» весьма обширно. Оно охватывает как какую-то отдельную величину, например год рождения человека или его имя, так и показания какого-либо датчика или производственные сведения фирмы. На бытовом уровне данные отождествляются со сведениями, и поэтому не любой информационный массив считается данными. Например, текст литературного произведения или учебника, картина художника, фильм не рассматриваются как данные, однако данными признаются сведения, в них содержащиеся. В компьютерных системах такого различия нет и любая информация, представленная в допустимой для компьютера форме — тексты, рисунки, музыка и др., — считаются данными. В информатике к данным относятся также тексты программ, хранящиеся на внешних носителях или загруженные для исполнения в память компьютера. Именно такое расширенное по содержанию толкование термина *данные* будет подразумеваться далее.

Данным приписываются несколько классификационных признаков. Важнейшим из них является *тип данных*. Тип данных определяет:

- набор их допустимых значений;
- правила их обработки (преобразования);
- порядок их размещения в ОЗУ и ВЗУ при хранении;
- порядок доступа к ним (т. е. обращение и извлечение при необходимости с места хранения).

Допустимый набор типов данных и их особенности определяются программной системой или языком программирования, на котором система написана. При этом возможности языков по разнообразию допустимых типов данных, а также построению новых типов различаются весьма сильно. Ясно, что чем более широкой и гибкой оказывается типизация данных в программной системе

или языке, тем больше возможностей предоставляется пользователю в решении задачи оптимального представления, хранения и применения данных. Типизация данных влияет и на компактность самой исполняемой программы. Например, в языке BASIC отсутствует тип данных «*записи*»; в результате для создания и использования базы данных пришлось бы организовывать параллельную обработку нескольких массивов.

Следующим признаком является деление данных на *элементарные* (одиночные, простые) и *структурированные* (сложные).

К *элементарным данным* относятся символы, числа (целые и вещественные) и логические данные. Общей и обязательной особенностью одиночных данных является то, каждое из них имеет *одно значение* и *собственное имя*. *Значение* — это содержимое тех ячеек памяти, где данное располагается. *Имя* (его называют также *идентификатор*) — это обозначение данного в тексте программы. Правила построения идентификаторов элементарных данных определяются языком программирования, на котором программа написана.

Элементарные данные являются «кирпичиками», объединением которых строятся *сложные данные*. Вариантов объединения существует много — это приводит к появлению множества типов *структур данных*.

> *Информационный массив, объединяющий данные и связи (отношения) между ними называется* ***структурированными данными.***

> *Перечень объединяемых одиночных данных, их характеристики, а также особенности связей между ними образуют* ***структуру данных.***

Примерами структурированных данных является страница из классного журнала с фамилиями учеников, датами занятий и отметками, телефонный справочник, организационная структура учреждения и т. п.

Перечень допустимых структур данных, как уже было сказано, определяется языком программирования или прикладной программой. Он может быть фиксированным (нерасширяемым), как в языке BASIC или прикладных программах без встроенных возможностей программирования. В языках объектного программирования (DELPHI, C++ и др.) и ряде прикладных систем наряду с зарезервированными типами структур данных допуска-

ется создание новых типов, причем элементами структуры могут быть сложные данные, например массив записей.

Сложные данные, как и элементарные, имеют *значения* и *идентификаторы*. Значения размещаются в ячейках ОЗУ по определенным схемам (см. п. 8.3.3). Правила построения идентификаторов устанавливаются языком программирования или программной системой. Исключение составляют правила формирования имен файлов — они задаются операционной системой и должны соблюдаться всеми работающими в ней программами и языками. Например, в MS-DOS в качестве имен файлов были допустимы комбинации из латинских букв, цифр и некоторых спецсимволов общей длиной не более 8 знаков; в Windows разрешены имена длиной до 255 знаков без ограничений применяемого набора символов*.

По возможности изменения значений данных (как простых, так и структурированных) в ходе их обработки подразделяют на *переменные* и *постоянные* (*константы*). Из названия очевидно, что *переменные* могут изменять свое значение по ходу исполнения программы, а *константы* — нет. На уровне операционной системы различие между переменными и постоянными величинами отсутствует, поэтому у них одинаковый порядок размещения в ОЗУ и доступа к ним. Разделение может производиться в языке программирования и, соответственно, в созданной с его помощью прикладной программе — это служит дополнительной мерой синтаксического контроля корректности программы.

В зависимости от того, на каком этапе обработки данные используются, они подразделяются на *исходные* (*входные*), *промежуточные* и *выходные*. К *исходным* относятся данные, необходимые для исполнения программы и вводимые в нее до или в процессе работы. Исходные данные могут быть предварительно записаны на некотором носителе и вводиться с него, поступать по линиям связи от каких-то датчиков или с других компьютеров, вводиться пользователем программы посредством устройств ввода. *Промежуточные* данные формируются в ходе исполнения программы и, чаще всего, пользователю недоступны; они не отображаются на устройствах вывода, но существуют в ОЗУ или на ВЗУ. Идентификаторы промежуточным данным присваивает

---

* На самом деле, для идентификации файла вместе с путем его расположения в директории отводится 260 знаков, т. е. чем длинней путь до файла, тем меньше возможная длина его имени.

разработчик программы или задает сама программа по заложенным в нее правилам. *Выходные* данные являются результатом работы программы — ради них и выполняется обработка входных. Выходные данные, предназначенные для человека, представляются в требуемой для него форме (тексты, рисунки, звуки); при хранении выходных данных на носителях или передаче по сетям сохраняется двоичный компьютерный формат их представления. Таким образом, работу программы можно рассматривать как действия по преобразованию входных данных в выходные через необходимые для этого промежуточные. С точки зрения самой программы все эти виды равноправны, т. е. обрабатываются только в соответствии с их типом, а не функциональным назначением или этапом.

Представление данных при их хранении и обработке требует решения трех основных задач:

- определить способы представления элементарных (простых) данных;
- определить способы объединения данных в структуры;
- установить способы размещения информации на материальном носителе.

Выделяют три *уровня* представления данных — *концептуальный, логический* и *физический*. На *концептуальном* уровне определяется общая структура информационного массива — она называется *моделью данных*. Известны и используются несколько моделей данных: *иерархическая*, *сетевая*, *реляционная*, *объектно-ориентированная*. В соответствии с выбранной моделью данных строится информационная система, в которой данные будут храниться, а также программы, ведущие их обработку (*манипулирование данными*). *Логический* уровень определяет способы представления элементарных данных, их перечень при объединении в структуру, а также характер связей между ними в рамках выбранной модели данных. *Физический* уровень определяет форматы размещения созданной логической структуры данных на внешних носителях информации (магнитных или оптических дисках, бумаге, в памяти компьютера). Представление данных является важным фактором, обеспечивающим, с одной стороны, компактный (т. е. экономный с точки зрения расходования поверхности или объема носителя) способ записи информации при хранении, с другой стороны, быстрый доступ к нужным данным при их использовании. Далее бу-

дут рассмотрены варианты решения перечисленных задач в компьютерных системах.

## 8.2. Представление элементарных данных в ОЗУ

Как уже сказано, различными типами элементарных данных являются символы, целые числа, вещественные числа и логические данные. Логический и физический уровни их представления определяются конструктивными особенностями ОЗУ компьютера. В частности, поскольку память компьютера имеет байтовую структуру, к ней привязывается представление любых данных. Более точным является утверждение, что для представления *значений* элементарных данных в памяти компьютера используется *машинное слово*:

> ***Машинное слово** — это совокупность двоичных элементов, обрабатываемая как единое целое в устройствах и памяти компьютера.*

С технической точки зрения машинное слово объединяет запоминающие элементы, служащие для записи 1 бит информации, в единую *ячейку памяти*. Количество таких объединяемых элементов кратно 8, т. е. целому числу байт. Например, в одной из первых крупных отечественных ЭВМ БЭСМ-6 длина машинного слова составляла 48 бит (6 байт), в первых персональных машинах IBM — 16 бит (2 байта). В архитектуре современных компьютеров длина машинного слова определяется шириной шины данных (32 или 64 бит). Доступ к машинному слову в операциях записи и считывания осуществляется по номеру ячейки памяти, который называется *адресом ячейки*.

> *Запоминающие устройства, в которых доступ к данным осуществляется по адресу ячейки, где они хранятся, называются устройствами с **произвольным доступом***.*

Время поиска нужной ячейки, а также продолжительность операций считывания или записи в ЗУ произвольного доступа одинаково для всех ячеек независимо от их адреса.

---

* Именно по этой причине в англоязычной литературе вместо ОЗУ используется термин RAM — **R**andom **A**ccess **M**emory — «*память с произвольной выборкой*».


**Рис. 8.1.** Представление значений элементарных данных с помощью 16-битных машинных слов: *а* — символы; *б* — целые числа со знаком; *в* — вещественные числа с плавающей запятой; *г* — логические данные

Для логического уровня важно то, что представление значений любых элементарных данных должно быть ориентировано на использование машинных слов *определенной* и *единой* для данного компьютера длины, поскольку их представление на физическом уровне выполняется именно в ячейках ОЗУ (на ВЗУ элементарные данные в качестве самостоятельных не представляются и доступ к ним отсутствует).

Рассмотрим особенности представления всех типов элементарных данных с помощью 16-битного машинного слова. Порядок размещения значений данных представлен на рис. 8.1.

Для представления *символов* (*литерных данных*) машинное слово делится на группы по 8 бит, в которые и записываются двоичные коды символов. Ясно, что в 16-битном машинном слове могут быть записаны одновременно два символа (рис. 8.1,*а*). Значениями одиночных литерных данных являются коды символов. Множество допустимых значений данных этого типа для всех кодировок, основанных на однобайтовом представлении, составляет $2^8 = 256$; двубайтовая кодировка *Unicode* допускает 65536 значений. Совокупность символов образует *алфавит*, т. е. для них

установлен лексикографический порядок следования в соответствии с числовым значением кода; это, в свою очередь, позволяет определить над множеством символьных данных операции математических отношений $>$, $<$, $=$. Непосредственно над одиночной символьной переменной определена единственная операция — изменение значения с одного кода на другой, что эквивалентно замене самой переменной. Все остальные действия производятся со сложными символьными данными, например типа String в языке PASCAL (см. п. 8.3).

В представлении *целых чисел со знаком* (например, тип Integer в языке PASCAL) старший бит (15-й), как уже обсуждалось в гл. 4, отводится под запись знака числа (0 соответствует «+», 1 — «−»), а остальные 15 двоичных разрядов — под запись прямого (для положительного) или обратного (для отрицательного) двоичного кода числа (рис. 8.1,*б*). При этом возможные значения чисел ограничены интервалом [−32768, 32767]. Наряду с описанным, используется и другой формат представления целых чисел — *беззнаковый*; очевидно, он применим только для записи положительных чисел. В этом случае под запись числа отводятся все 16 двоичных разрядов и интервал разрешенных значений оказывается [0, 65535] (в PASCAL'е такой числовой тип называется Word). Помимо математических отношений над целыми числами определены операции сложения, вычитания и умножения (в тех случаях, когда они не приводят к переполнению разрядной сетки), а также целочисленного деления и нахождения остатка от целочисленного деления.

Представление *вещественных чисел с плавающей запятой* также рассматривалось ранее. При записи числа оно переводится в нормализованную форму с выделением и отдельным хранением знака мантиссы, знака порядка, порядка и мантиссы. Для представления числа отводится несколько машинных слов. Ситуация, соответствующая числовому типу Single в языке PASCAL, когда для представления числа отводится два машинных слова, проиллюстрирована на рис. 8.1,*в*. Этой формой охватывается диапазон модулей мантиссы $[1{,}5 \cdot 10^{-45}; 3{,}4 \cdot 10^{38}]$ с 7-ю десятичными цифрами мантиссы. Запись мантиссы занимает с 0-го по 23-й разряд. Поскольку мантисса выбирается такой, чтобы ее модуль отвечал условию $0{,}1_2 \leqslant |M_2| < 1$, всегда значение старшего разряда числа 0 (целых) — оно не отображается при записи, а значение следующего разряда всегда 1. В процессе выполнения операций может произойти переполнение разряд-

ной сетки (на 1 разряд) или, наоборот, ее освобождение (т. е. в первом отображаемом разряде окажется 0) — по этой причине после каждой операции с числами в такой форме выполняется *нормализация результата*, которая состоит в таком изменении порядка числа и сдвиге мантиссы, чтобы первой значащей цифрой снова оказалась 1 (см. п. 4.3.3). Изменение порядка в представлении числа эквивалентно перемещению разделителя целой и дробной частей числа; как уже говорилось, такая форма получила название «*с плавающей запятой*». Благодаря применению плавающей запятой выполняется автоматическое масштабирование чисел в ходе вычислений, что снижает погрешность их обработки. Над вещественными числами определены все четыре арифметические операции. Помимо этого имеются операции преобразования вещественного типа к целому (например, round и trunc в PASCAL'e).

*Логические данные* могут принимать одно из двух значений — 0 или 1 (0 соответствует логическому *False*, 1 — *True,* причем принимается False < True). Для их записи было бы достаточно отвести всего один двоичный разряд. Однако в ОЗУ компьютера отсутствует доступ к отдельному биту, поэтому для представления логических данных выделяется целый байт, в младший разряд которого и помещается значение. Таким образом, в машинном слове логические данные располагаются в 0-м и в 8-м битах (см. рис. 8.1,*г*). Над логическими данными определены операции: логическое умножение (конъюнкция, $\wedge$), логическое сложение (дизъюнкция, $\vee$), логическое отрицание ($\neg$). Примером логических данных может служить тип Boolean в PASCAL'e.

Значения элементарных данных формируются в ходе исполнения программы и имеют физическое представление в ОЗУ. В отличие от них идентификаторы данных существуют *только на уровне логического представления* — они используются для обозначения данных в тексте программы, однако при трансляции программы с языка программирования в машинный код имена заменяются номерами ячеек, в которых данные размещаются. При исполнении такой программы обращение к данным выполняется по адресу ячейки, а не идентификатору. Адреса могут быть *абсолютными* — в этом случае они не изменяются при загрузке программы в ОЗУ — именно такой способ адресации применяется в исполняемых программных файлах с расширением .com. Однако в силу некоторых особенностей распределения памяти компьютера размер таких программ не может превышать

64 Кб. В исполняемых файлах с расширением .exe на этапе трансляции устанавливаются *относительные* адреса данных, которые конкретизируются при размещении программы в ОЗУ — это несколько замедляет начало исполнения, зато снимает указанное выше ограничение на размер программы.

В заключение следует заметить, что в некоторых прикладных программах в качестве элементарных используются и другие типы данных, например тип Data или «Денежный» в MS Excel и MS Access. Однако они являются самостоятельными только для пользователя программы; самой же программой они сводятся к некоторой комбинации рассмотренных выше элементарных данных.

## 8.3. Структуры данных и их представление в ОЗУ

Можно указать ряд причин, поясняющих необходимость и удобство использования данных, организованных в некоторую структуру:

- отражение организацией данных логики задачи, объективно существующих взаимосвязей и взаимообусловленности данных;
- оптимизация последовательности обработки данных;
- широкое применение при обработке данных циклических конструкций — в них при переборе нельзя автоматически менять имя переменной, однако можно изменять индексы;
- неудобство использования большого количества одиночных данных, поскольку это ведет к необходимости использования многих имен.

Перечисленные причины приводят к тому, что в современных языках и системах программирования резервируется широкий спектр различных структур данных и, помимо этого, предусматривается возможность создания структур, удобных и необходимых пользователю.

Относительно структур данных необходимо сделать следующие общие замечания:

- логический уровень организации данных отражается в тексте программы — им определяется порядок обработки данных;
- физический уровень представления структур в ОЗУ имеет всего две разновидности: *последовательные списки* и *связные списки* (см. п. 8.3.3); на ВЗУ все структуры представляются в виде *файлов*;

- обработка данных возможна только после их размещения в ОЗУ; с ВЗУ определены только операции записи и чтения;
- идентификаторы, как и у одиночных данных, существуют только в тексте программы и на этапе трансляции переводятся в адреса ячеек памяти.

### 8.3.1. Классификация и примеры структур данных

Как следует из приведенного выше определения, структурирование данных предполагает существование (или установление) между ними каких-то отношений (связей). В зависимости от характера этих отношений можно выделить несколько классификационных признаков структур данных.

Первым из них рассмотрим отношение *порядка*. По порядку данных структуры делятся на *упорядоченные* и *неупорядоченные*.

В *упорядоченных* структурах элементы размещаются по порядку в соответствии со значением некоторого признака. Наиболее простым признаком является *порядковый номер элемента*; установление порядка в соответствии с номером называется *нумерацией*. При этом если весь набор имеет один общий идентификатор (например, M), то отдельным данным присваиваются собственные идентификаторы — *индексы* (например, M(5) или NAS(Z3)). Чаще всего индекс задается целым числом, хотя это необязательно — в качестве индекса может выступать любой знак из конечного алфавита. Лексикографический порядок индексов определяет *отношение следования* между элементами структуры, т. е. элемент M(6) следует за элементом M(5), а элемент NAS(Z3) располагается перед элементом NAS(Z3 + 1). Примером структур, в которых упорядочение выполняется по номеру элемента, являются *массивы*. Порядковый номер элемента можно считать *внешним* признаком, который может присваиваться элементу независимо от его значения. Например, регистрационный номер документа определяется только временем его поступления в учреждение, а не его содержанием. Помимо нумерации в структурах данных используется упорядочение *по значению* некоторого *внутреннего признака*, например размещение фамилий в алфавитном порядке или группы предприятий в порядке убывания их рентабельности, — такое упорядочение называется *ранжированием*.

Примером неупорядоченных структур являются *множества* — в них не определен порядок элементов; единственное, что

можно установить для каких-то конкретных данных, — их принадлежность (или не принадлежность) выбранному множеству.

Следующим классификационным признаком структур является *однородность*. К *однородным* относятся структуры, содержащие элементарные данные только одного типа. *Неоднородные* структуры объединяют данные разных типов. Примерами однородных структур являются массивы, множества, стеки. К неоднородным структурам относятся записи.

Еще одним признаком является *характер отношений* между элементами. По взаимной подчиненности элементов структуры данных подразделяются на *линейные* и *нелинейные*.

В *линейных* структурах все элементы *равноправны*. К ним относятся массив, множество, стек, очередь.

В *нелинейных* структурах между элементами существуют *отношения подчиненности* или они могут быть связаны логическими условиями. К ним относятся деревья, графы, фреймы.

Основываясь на выделенных классификационных признаках, рассмотрим и охарактеризуем некоторые структуры данных.

***Массив** — упорядоченная линейная совокупность однородных данных.*

Комментарии к определению:

- термин «*упорядоченная*» означает, что элементы массива пронумерованы;
- термин «*линейная*» свидетельствует о равноправии всех элементов;
- термин «*однородных*» означает следующее: в том случае, когда массив формируется из элементарных данных, это могут быть данные лишь одного какого-то типа, например массив чисел или массив символов; однако возможна ситуация, когда элементами массива окажутся сложные (структурные) данные, например массив массивов, в этом случае «*однородных*» означает, что все элементы имеют одинаковую структуру и размер.

*Количество индексов, определяющих положение элемента в массиве, называется **мерностью** массива.*

Так, если индекс единственный, массив называется *одномерным*; часто такой массив называют также *вектором*, строкой или столбцом. Для записи элементов одномерного массива используется обозначение $m_i$; в языках программирования приняты обозначения m(i) или m[i].

Массив, элементы которого имеют два индекса, называется *двумерным* или *матрицей*. Пример обозначения: G[3,5]; при этом первый индекс является номером строки, а второй индекс — номером столбца, на пересечении которых находится данный элемент.

Массивы с тремя индексами называются *трехмерными* и т. д. Максимальная мерность массива может быть ограничена синтаксисом некоторых языков программирования либо не иметь таких ограничений.

Максимальное значение индексов определяет *размер* массива. Размер массива указывается в блоке описания программы, поскольку при исполнении программы для хранения элементов массива резервируется необходимый объем памяти. Если в процессе исполнения программы размер массива не изменяется (или не может быть изменен), то в этом случае говорят о массивах *фиксированного размера*; если определение размеров массива или их изменение происходит по ходу работы программы, то такие массивы называются *динамическими* (динамически описываемыми).

Допустимый набор операций над элементами массива определяется типом данных (элементарных или структурированных), из которых массив сформирован. В некоторых языках программирования над массивом в целом определена операция присваивания M := <выражение> — в этом случае всем элементам массива присваивается одинаковое значение, равное вычисленному значению выражения; возможна также операция присваивания для двух одинаковых по типу, размеру и размерности массивов M2 := M1 — выполняется *поэлементное* присваивание значений (M2(i,j,k,...) := M1(i,j,k,...)).

Особое место занимают символьные массивы — они называются *строками* или *строковыми данными* (например, тип String в PASCAL'e). С ними возможен целый набор операций, неопределенных для одиночных символьных данных. В первую очередь, это операция *конкатенации* (объединения) строк с формированием новой строки. Помимо этого имеются операции замещения части строки, а также определения ее числовых характеристик.

***Стек (магазин) и очередь** являются упорядоченными, линейными, неоднородными структурами.*

Эти структуры реализуются в виде специальным образом организованных областей ОЗУ компьютера либо в качестве самос-

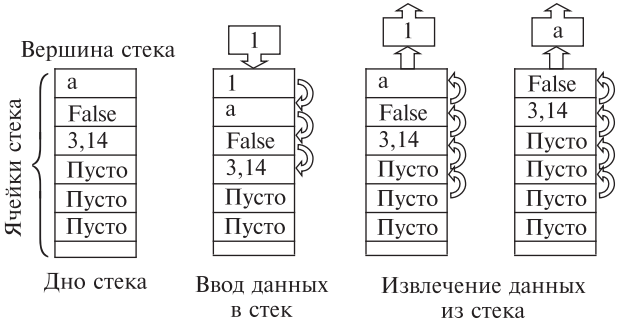



**Рис. 8.2.** Структура стека

тоятельных блоков памяти. В стеке ячейки памяти (или регистры стековой памяти) соединяются друг с другом таким образом, что при занесении данных в первую ячейку содержимое всех остальных сдвигается в соседние вниз, при считывании — содержимое сдвигается вверх по ячейкам, как показано на рис. 8.2. Другими словами, вход в стек возможен только через первую ячейку (*вершину стека*), поэтому извлекаться первой будет та информация, которая была занесена последней*, подобно пассажиру переполненного автобуса.

Отличие очереди от стека только в том, что информации извлекается в порядке «*первым вошел — первым вышел*», т. е. со дна стека.

Таким образом, данные имеют *порядок расположения*, и они *равноправны*, поэтому структура является упорядоченной и линейной. Однако в общем случае в ячейках стека могут содержаться данные разных типов — по этому признаку структура оказывается неоднородной.

Описанный способ организации данных оказывается удобным при работе с подпрограммами, обслуживании прерываний, решении многих задач.

***Дерево***, или *иерархия* является примером *нелинейной* структуры.

В ней элемент каждого уровня (за исключением самого верхнего) входит в один и только один элемент следующего (более высокого) уровня. Элемент самого высокого уровня называется *корнем*, а самого нижнего уровня — *листьями*. Схема та-

---

* Часто стек называют памятью типа *LIFO* (***L**ast **I**nput **F**irst **O**utput*: «*последним вошел — первым вышел*»).

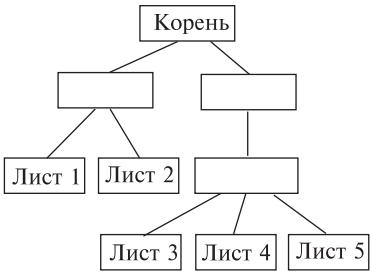



**Рис. 8.3.** Структура дерева

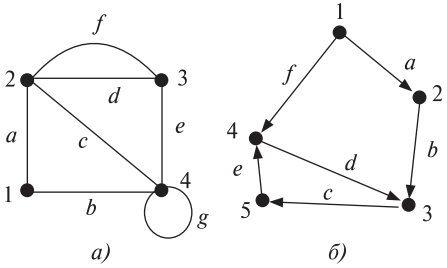



**Рис. 8.4.** Примеры графов: *а* — граф 1 (неориентированный); *б* — граф 2 (ориентированный)

кой структуры показана на рисунке. Отдельные элементы могут быть однородными или нет. Примером подобной организации служат файловые структуры на внешних запоминающих устройствах компьютера.

**Граф.** Часто отношения между данными представляются в виде *графа* — совокупности точек и линий, в которой каждая линия соединяет две точки (рис. 8.4). В информатике точка получает смысл элемента структуры (системы, данных и пр.), а линии — смысл отношения между элементами. Ознакомимся с рядом терминов, связанных с использованием графов.

Точки называются *вершинами* графа, линии — *ребрами*. Если ребро соединяет две вершины, то говорят, что ребро *инцидентно* этим вершинам, а сами вершины называются *смежными*. Число ребер, инцидентных вершине, называется *степенью* вершины. Если два ребра инцидентны одной и той же паре вершин, они называются *кратными*. Ребро, у которого совпадают обе вершины, называется *петлей*. На рис. 8.4,*а* граф 1 задается списком вершин $\{1, 2, 3, 4\}$ и списком ребер, в котором для каждого ребра указывается пара инцидентных ему вершин: $a(1,2); b(1,4); c(2,4); d(2,3); e(3,4); f(2,3); g(4,4)$. Смежные пары вершин: $(2,3), (2,4), (1,2), (1,4), (3,4)$. Ребро $g$ является петлей; ребра $d$ и $f$ — кратные. Степени вершин 2 и 4 равны 4; вершины 3 — 3; вершины 1 — 2.

Ребро, соединяющие вершины, может иметь направление, тогда оно называется *ориентированным* и изображается стрелкой. Граф, в котором все ребра ориентированные, называется ориентированным; его ребра часто называют *дугами*. Дуги называют *кратными*, если они соединяют одни и те же вершины и совпадают по направлению. При обозначении дуги всегда снача-

ла указывается вершина, из которой она начинается, например на рис. 8.4,*б* это $(2, 3)$.

*Маршрут* — это последовательность ребер, в котором конец предыдущего ребра совпадает с началом следующего, например $a, c, e$ на графе 1. Маршрут, в котором конечная вершина совпадает с начальной, называется *циклом*, например $c, e, d$ на графе 2. Граф называется *связным*, если между любыми двумя его вершинами имеется маршрут. Связный граф с $n$ вершинами содержит не менее $n - 1$ ребро.

По рассмотренной ранее классификации *граф является упорядоченной, нелинейной, неоднородной структурой.* Понятие графа благодаря его наглядности и высокой общности в информатике выступает в качестве основного средства описания структур данных, систем, порядка выполнения действий. Примером может служить блок-схема программы. Графы используются, в частности, при описании конечных автоматов (см. гл. 11).

С точки зрения практического использования большой интерес представляет еще один тип данных, образующих упорядоченную, линейную, неоднородными структуру. Такую структуру можно рассматривать как массив неоднородных структурированных данных. Структура называется *таблицей*, а ее отдельная строка — *записью* (*логической записью*). Ввиду большой значимости данной структуры она будет рассмотрена более подробно.

### 8.3.2. Понятие логической записи

***Логическая запись** — поименованная совокупность элементарных данных, имеющая смысловую завершенность.*

Пример записи — строка списка студентов:

| Фамилия | Год рождения | Год поступления в вуз | Курс | Номер зачетной книжки |
|---|---|---|---|---|
| Васильев | 1995 | 2012 | 3 | 5544332211 |

Определение требует ряда комментариев и дополнений:

1. Логическая запись объединяет не любые разрозненные (по смыслу) данные, а те, что относятся и характеризуют некоторую систему или объект, — именно в этом плане следует понимать сочетание «*смысловая завершенность*» в определении. Запись в целом отражает различные свойства (атрибуты) системы.

2. Логическая запись имеют многоуровневую структуру. Элементами самого нижнего уровня являются элементарные дан-

ные — символы, числа, логические данные. Элементарные данные хранятся и считываются целиком, доступ к их частям невозможен. Совокупности элементарных данных, имеющих определенный смысл, но не обладающих смысловой завершенностью, образуют поля, каждое из которых соответствует одному атрибуту системы. Поле характеризуется типом элементарных данных, из которых оно строится, а также информационным размером (т. е. указанием количества байт, которое отводится для представления данного поля в записи).

3. Поля записи связаны между собой. Связи могут носить функциональный характер (значение одного поля посредством некоторого преобразования (правила) определяет значение другого; например, две первые цифры поля «Номер зачетной книжки» равны двум последним поля «Год поступления»), либо связи могут быть причинно-следственными (например, поле «Год рождения» определяется значением поля «Фамилия»).

4. Логические записи сами могут объединяться и образовывать структуры, которые определяются моделью данных. Например, совокупность указанных выше записей для всех студентов, обучающихся в одной группе, образуют массив, который называется базой данных (реляционного типа). Обращение к базе при сохранении и использовании осуществляется по ее идентификатору (ГРУППА_101). Возможны и более высокие структурные объединения, например структуры, элементами которых будут базы данных (объединение баз данных по всем группам факультета). Программные системы, позволяющие создавать и использовать базы данных, называются системами управления базами данных (СУБД).


**Рис. 8.5.** Иллюстрация многоуровневой иерархической структуры данных

5. Логическая запись имеет собственный идентификатор, по которому можно обратиться к записи в целом (например, порядковый номер студента в группе). Поля также имеют идентификаторы, по которым они становятся доступны для просмотра или изменения значения. Идентификатор поля строится из идентификатора базы, идентификатора записи и собственно имени поля, например:

ГРУППА_101(13).Фамилия.

Таким образом, существует иерархическая многоуровневая структура данных, показанная на рис. 8.5. Каждый выше расположенный уровень содержит низлежащие в качестве составных элементов. В этой иерархии запись является первым элементом структуры, обладающим смысловой завершенностью и, следовательно, самостоятельностью. Более высокие структуры образуются повторением записей с одинаковой и неизменной структурой.

### 8.3.3. Организация структур данных в ОЗУ

Структура информационного массива определяется один раз на этапе его создания и в процессе использования уже не изменяется. В языках программирования это достигается описанием структуры в блоке описаний программы; в СУБД — установлением перечня и последовательности полей записи на начальном этапе создания базы данных. Всякое изменение структуры (например, введение дополнительного поля записи или удаление имеющегося) эквивалентно созданию новой структуры. Что же касается количества записей в структурированном информационном массиве, то при представлении его в ОЗУ компьютера возможны две ситуации: либо под него выделяется область ОЗУ фиксированного размера, либо размер области при необходимости может меняться.

В первом варианте в начале работы программы происходит резервирование областей ОЗУ для хранения информационных массивов. С этой целью в тексте программы указывается, какого типа и размера информационные массивы будут в дальнейшем использованы. В процессе выполнения программы могут меняться *значения* элементов информационного массива, но не его размер. По этой причине в случае, если размер (число элементов) массива не известен заранее (например, количество учеников в классе), приходится осуществлять избыточное резервирование, что, безусловно, приводит к нерациональному использованию памяти компьютера. Именно таким образом происходит резервирование памяти при описании массивов и других структурных данных в языке PASCAL. Отсутствие возможностей *динамического* описания массивов (т. е. введение новых массивов или изменение размеров имеющихся в процессе выполнения программы) считается одним из существенных недостатков этого языка программирования.

Информационные массивы, допускающие изменение размера (но не структуры!), называются *динамическими*. В этом случае

| № записи | № ячейки | Содержание |
|---|---|---|
| 1 | 3000 | Запись A |
| 2 | 3012 | Запись B |
| 3 | 3024 | Запись C |
| ... | ... | ... |
| $N$ | $3000 + (N-1) \cdot 12$ | Запись $N$ |
| | | |

*а)*

| № записи | № ячейки | Содержание |
|---|---|---|
| 1 | 4000 | Запись A |
| 2 | 4012 | Запись B |
| 3 | 4024 | Запись C |
| ... | ... | ... |
| $N$ | $4000 + (N-1) \cdot 12$ | Запись $N$ |
| $N+1$ | $4000 + N \cdot 12$ | Запись $N+1$ |

*б)*

| № записи | № ячейки | Содержание |
|---|---|---|
| 1 | 4000 | Запись A |
| 2 | 4012 | Запись C |
| 3 | 4024 | Запись D |
| ... | ... | ... |
| $N-1$ | $4000 + (N-2) \cdot 12$ | Запись $N$ |
| $N$ | $4000 + (N-1) \cdot 12$ | Запись $N+1$ |

*в)*

**Рис. 8.6.** Схемы последовательного динамического размещения данных в ОЗУ: *а* — размещение в ОЗУ; *б* — добавление записи; *в* — удаление записи

данные могут иметь *последовательное* или *связное* представление в ОЗУ.

*Последовательное* представление иллюстрируется рис. 8.6. В этом варианте данные (отдельные записи) размещаются в соседних последовательно расположенных ячейках памяти. На размещение одной записи может потребоваться несколько ячеек (машинных слов), но их количество *одинаково* для каждой из за-

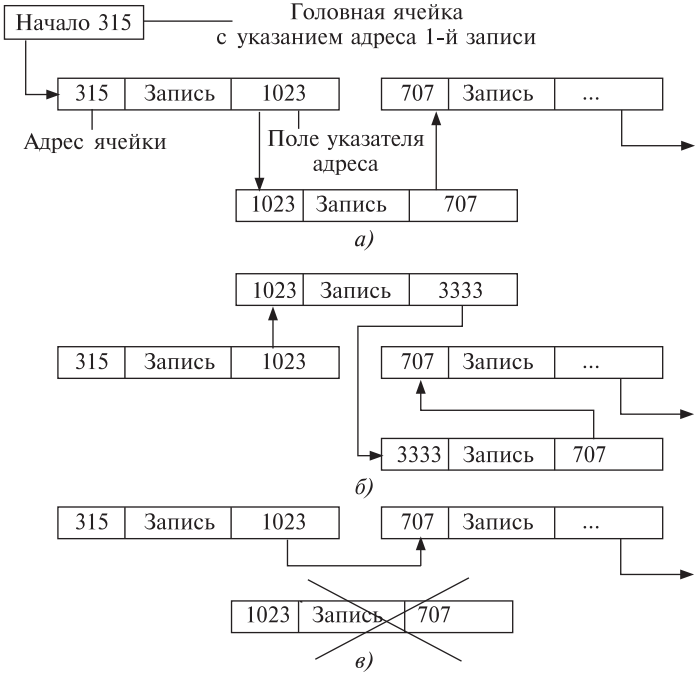



**Рис. 8.7.** Схема связного динамического размещения данных в ОЗУ: *а* — организация связей между записями; *б* — добавление записи; *в* — исключение записи

писей (в представленном на рис. 8.6,*а* примере — 12), поэтому идентификатор записи однозначно связывается с номером первой ячейки, начиная с которой запись размещается. Физический порядок следования полностью соответствует логическому. Такая совокупность записей называется *последовательным списком*. Для его хранения в ОЗУ выделяется блок ячеек фиксированного размера. При выполнении команды обрабатывающей программы «*Добавить запись*» происходит увеличение размера массива на одну строку в конце блока и при необходимости массив перезаписывается в ОЗУ (возможно, с изменением адресов). В добавленной строке размещается новая запись, как это показано на рис. 8.6,*б*. При изъятии каких-то записей по команде «*Удалить запись*» соответствующая строка очищается и после перезаписи заполняется содержимым следующих ячеек (рис. 8.6,*в*).

*Связное* представление данных основано на том, что в записи дописывается дополнительное поле, в котором размещается *указатель адреса*, т. е. ссылка на то место в ОЗУ, где распола-

гается следующая запись. При этом физический порядок размещения записей не соответствует логическому — записи располагаются в любых свободных ячейках ОЗУ, причем не обязательно подряд. Такие структуры называются *связными списками*. Их удобство состоит в гибкости структуры — без перезаписи остальных элементов можно легко добавлять новые или исключать имеющиеся — для этого достаточно лишь изменить состояние поля указателя адреса, что иллюстрируется рис. 8.7.

Недостаток описанного способа представления информационного массива в ОЗУ состоит в том, что в нем невозможно напрямую обратиться к нужной записи — поиск ее осуществляется по цепочке переходов, безусловно, увеличивая время доступа к данным.

## 8.4. Представление данных на внешних носителях

### 8.4.1. Иерархия структур данных на внешних носителях

Основными информационными единицами при сохранении данных на внешних носителях являются:

- логическая запись;
- физическая запись;
- файл;
- каталог (папка).

*Логическая запись* при хранении на внешних носителях является той же информационной единицей, что и при хранении в ОЗУ. Отличие состоит в том, что при хранении на носителе запись является *минимальным* и *неделимым* элементом представления данных. Это означает, что после размещения записи на носителе отсутствует доступ к ее отдельным полям, а операции переноса на носитель и считывание с него производятся целиком со всей записью. Поскольку обработка записей при их хранении не происходит, не требуется и различия типов данных, т. е. запись может состоять из одного элементарного данного, группы данных или содержать структурированные данные. Единственной характеристикой отдельной записи является ее длина, а допустимыми операциями — перенос на носитель и считывание и него.

После размещения данных на носителе они превращаются в *физическую запись*.

> ***Физическая запись*** *— элемент поверхности или объема носителя, на котором в соответствии с физическими принципами функционирования носителя размещаются данные, составляющие логическую запись.*

Объединение физических записей образует *файл*.

> ***Файл*** *— определенным образом оформленная совокупность физических записей, рассматриваемая как единое целое и имеющая описание в системе хранения информации.*

Комментарии к определению:

1) «*оформленная совокупность записей*» означает, что, помимо непосредственно записей, файл всегда имеет имя (идентификатор) и признак конца файла *EOF* (End-Of-File); по имени файл отыскиваться на носителе; признак *EOF* необходим, поскольку по нему устанавливается ближайшее к данному файлу свободное место, в которое можно вести запись следующего файла, а при пересылке данных с носителя в ОЗУ по нему определяется граница информационного массива;

2) «*как единое целое*» означает, что при обращении к файлу отсутствует доступ к отдельным его записям; файл записывается и считывается только целиком; в операционных системах над файлами определен целый ряд действий: копирование, перемещение, удаление, переименование и некоторые другие, однако, в конечном счете, все они сводятся только к операциям чтения и записи, а также изменениям в описании файла;

3) «*описание в системе*» означает сохранение на носителе не только самих файлов, но и сведений о них и их размещении; эти сведения используются в операциях с файлами.

Любые файлы содержат данные, закодированные с помощью двоичного алфавита. Однако способы кодирования и назначение файлов могут быть различными. По этой причине файлам приписывается еще одна характеристика — *тип*. Тип входит в идентификатор файла и указывается в виде *расширения* имени, например *Глава_8.doc*, *proba.pas* или *calc.exe*. Принципиально различными по типам следует считать *программные* (исполняемые) *файлы* и *файлы данных*. *Программные файлы* содержат тексты программ в машинном коде; они могут быть загружены в ОЗУ и исполняться. *Файлы данных* формируются в результате работы какой-либо программы; они не являются исполняемыми и служат только в качестве хранилищ данных. Многие программные системы при формировании файлов данных приписывают

им вполне определенные расширения — по ним можно установить, какой программой файл создан; например, расширения *txt, doc, docx, rtf* имеют файлы, подготовленные в текстовых редакторах, *jpg, png, gif, cdr* — графические файлы, *pas, bas, c* — файлы с текстами программ и т. д. Тип файла, как и его собственно имя, являются частью описания файла и сохраняются системой, ведающей размещением файлов на носителе.

Самым верхним уровнем представления данных на внешних носителях являются *структуры файлов — каталоги* (в операционной системе Windows принят термин «*папки*») — в них помещаются файлы, объединенные каким-то признаком, например принадлежности к одной программной системе или одной информационной базе. Как правило, каталоги допускают образование *вложенных структур*, т. е. подкаталогов (или, что то же самое, каталогов в каталогах). Каталоги образуют иерархическую структуру (см. п. 8.3.1), поэтому правомочно использование термина «*дерево каталогов*». При этом каталог, располагающийся на вершине иерархии, называется *корневым*.

Создает и поддерживает файловые структуры, определяет максимальный уровень вложенности каталогов, а также производит все операции с файлами и каталогами часть операционной системы компьютера — *файловая система*.

### 8.4.2. Особенности устройств хранения информации

Не вдаваясь глубоко в техническую сторону, рассмотрим некоторые особенности устройств, используемых для хранения информации в компьютерах.

> *Устройства, выполняющие операции, связанные с сохранения и считывания данных на материальном носителе, называются* ***внешними запоминающими устройствами*** *(ВЗУ) или устройствами внешней памяти.*

Любое ВЗУ реализует один из двух возможных принципов размещения информации — *последовательный доступ* или *прямой доступ*. Первый вариант используется при сохранении информации на ленточных носителях, например магнитной или бумажной ленте, — в этом случае записи размещаются одна за другой, т. е. последовательно. Записи также считываются *последовательно*, и для того чтобы отыскать нужную запись, требуется просмотреть все предыдущие, подобно поиску кадра на кинопленке.

Носителя последовательного доступа в настоящее время практически не используются. Для реализации *прямого* доступа на носителе должны быть обозначены (пронумерованы) области для записи информации — такие области называются *блоками*. Блок, подобно ячейке ОЗУ, служит контейнером для размещения данных. Обратиться к данным для записи-считывания можно по номеру (идентификатору) блока. Операция разбиения поверхности носителя на блоки называется *форматированием* — она выполняется в обязательном порядке и предшествует использованию носителя. Блок обычно имеет строго определенную для данного носителя информационную емкость и может содержать только целое число физических записей — из-за этого часть блока длиной меньше, чем размер записи, оказывается пустой и не используется. Например, при длине записей по 150 байт в один блок размером 512 байт поместятся 3 записи, а 62 байта останутся свободными.

На носителях большой емкости, например жестких магнитных дисках (винчестерах) и flash-картах, блоки объединяются в группы — *кластеры* — это уменьшает общее количество адресов и, следовательно, ускоряет поиск и доступ к файлу Точнее, разбиение на кластеры выполняется в процессе первичного форматирования. Размер кластера может быть задан различным в зависимости от типа выбранной пользователем файловой системы:

- FAT — 16-разрядная файловая система, обеспечивающая совместимость с ранними версиями Windows; возможные размеры кластеров 32 и 64 Кб;
- FAT-32 — 32-разрядная файловая система для современных версий Windows; размер кластера от 512 байт до 16 Кб;
- NTFS — файловая система по умолчанию используемая в новых версиях Windows — он обеспечивает назначение прав доступа к отдельным файлам; кластеры от 512 байт до 64 Кб;
- exFAT — файловая система, используемая для flash-носителей; размер кластера может быть установлен в пределах от 512 байт до 32 Мб.

Выбор размера кластера определяется особенностями файлов — если, например, предполагается хранить много файлов малой длины, то и кластеры разумно выбирать небольшого размера.

На дисковых носителях имена файлов хранятся отдельно от физических записей. В определенном месте диска при его форматировании создается специальная область, в которой располагается *таблица размещения файлов* — FAT (***F**ile **A**llocation*


**Рис. 8.8.** Схема чтения-записи данных на ВЗУ

***Table***). В эту таблицу заносятся имена и атрибуты файлов (дата и время создания, размер, атрибуты доступа), а также номер кластера, с которого начинается размещение файла. Таким образом, обращение к файлу происходит в два этапа: сначала с помощью файловой таблицы по имени файла находится номер кластера, а затем считывающе-записывающее устройство ВЗУ устанавливается над ним и производит операции. Ситуация иллюстрируется рис. 8.8.

При обмене между ВЗУ и ОЗУ данные пересылаются не отдельными записями, а кластерами, размер которых совпадает с размером кластера ВЗУ; схема обмена представлена на рис. 8.9. Для организации обмена в ОЗУ выделяется специальная область — *буфер обмена*; размер буфера устанавливается при конфигурировании операционной системы компьютера. При пересылке из ОЗУ в ВЗУ данные (записи, входящие в файл) сначала из ОЗУ пересылаются в буфер, пока он не заполнится, затем целым блоком отправляются в подготовленный кластер ВЗУ. Считывание идет обратным путем. Обмен может происходить, минуя центральный процессор, — в этом случае одновременно с обменом может проводиться обработка данных (поступивших или иных).

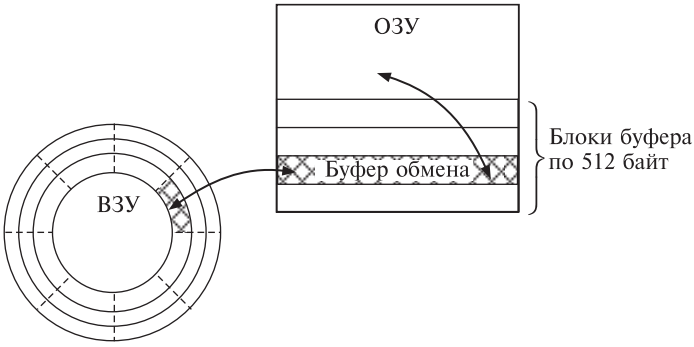



**Рис. 8.9.** Схема обмена блоками данных между ОЗУ и ВЗУ

Следует заметить, что, хотя организация прямого доступа к данным на ВЗУ весьма напоминает организацию произвольного доступа к ячейкам ОЗУ (то и другое выполняется по адресу; время доступа не зависит от адреса), между этими способами имеется различие. Из ячеек ОЗУ могут быть извлечены отдельные данные (например, элементы полей логической записи); кроме того, ОЗУ непосредственно связано с устройством обработки данных (центральным процессором). При желании использовать данные с ВЗУ, во-первых, сначала весь информационный массив должен быть перенесен в ОЗУ (с ВЗУ никакой обработки не ведется); во-вторых, нужные данные в этом массиве отыскиваются последовательным образом. Другими словами, прямой доступ оказывается некой комбинацией произвольного и последовательного.

## Контрольные вопросы и задания к Гл. 8

1. Для чего при представлении данных в компьютере необходима их типизация?

2. Возможно ли изменение (преобразования) *типа* одиночной переменной? Приведите примеры.

3. Разнесите понятия: *«переменная», «значение переменной», «имя переменной», «тип переменной»*. На каких этапах хранения и обработки данных эти понятия определены?

4. Чем обусловлена необходимость структурирования данных?

5. Приведите примеры практических задач, при решении которых целесообразно использовать массивы, записи, таблицы.

6. Приведите примеры иерархической организации данных.

7. Приведите примеры отношений между данными, представленных с помощью неориентированных и ориентированных графов.

8. В чем преимущества и недостатки *последовательных* и *связных* списков в ОЗУ?

9. Разнесите понятия: *логическая запись, физическая запись, файл* при хранении данных на ВЗУ.

10. С какой целью выполняется форматирование дискового носителя?

11. Как связан тип доступа к данным со способом их хранения?

12. Почему на дисковых носителях невозможен произвольный доступ к данным?

13. Выдвиньте причины, по которым становится целесообразным объединение блоков в кластеры при использовании магнитных дисковых носителей.

14. Каковы функции *FAT* в организации размещения файлов на дисковом носителе?

# Часть II

## АЛГОРИТМЫ. МОДЕЛИ. СИСТЕМЫ

Компьютер нужен человеку для решения задач практики. Примерами таких задач могут быть: описание поведение тела, двигающегося в среде с сопротивлением; описание последствий ядерной войны; построение оптимального варианта транспортных перевозок; прогнозирование результатов сброса промышленных отходов в водоем, проектирование сооружения, написание книги и т. п. Несмотря на значительное различие задач, просматриваются общие моменты в порядке их решения: *во-первых*, требуется выделить систему и построить ее информационную модель — ею определяется набор данных и их взаимосвязи; *во-вторых*, должен быть установлен порядок обработки данных.

Это звенья одной последовательности решения, поэтому представляется вполне оправданным рассмотреть их совместно, причем мы начнем с обсуждения второй составляющей — обработки данных.

В общем случае *обработка состоит в преобразовании по некоторым правилам исходной данных, в результате чего появляются новые данные*. Бесспорно важным оказывается то обстоятельство, что преобразование должно осуществлять некоторое техническое устройство в автоматическом режиме (т. е. без участия человека на каждом этапе преобразования). В связи с этим возникает ряд взаимосвязанных задач, требующих решения:

- определение правил обработки информации с учетом того, что она представлена в дискретной форме;
- установление, каким требованиям должно удовлетворять устройство, производящее обработку;
- определение того, каким образом данные и последовательность обработки может быть представлена для исполнения устройству.

Ответы на последний вопрос (представление данных) частично получены нами в гл. 8. Общие подходы к решению проблем обработки дискретной информации изучаются в теории алгоритмов, к рассмотрению элементов которой мы и приступаем.

# 9 Элементы теории алгоритмов

Начнем с рассмотрения элементов теории алгоритмов. Это рассмотрение будет построено в два этапа: на первом мы постараемся построить возможно более строгое определение алгоритма и обсудим его свойства, а на втором этапе изучим теоретические модели алгоритмов, а также исследуем проблему алгоритмической разрешимости задач.

## 9.1. Нестрогое определение алгоритма

Исторически термин «*алгоритм*» произошел от фамилии узбекского математика IX века Мухаммада ибн Муса ал-Хорезми, который впервые сформулировал правила четырех основных арифметических действий. Поначалу именно эти правила назывались алгоритмами, но затем термин получил дальнейшее развитие в первую очередь в математике — алгоритмом стал называться любой способ вычислений, единый для некоторого класса исходных данных, например нахождение производной функции. Одна из важнейших задач обучения математике состоит именно в освоении общих вычислительных алгоритмов. Другими словами, если школьника учат перемножать столбиком два числа, то при этом предполагается, что он осваивает не умножение конкретных выбранных чисел, а универсальный способ (алгоритм), который в дальнейшем может быть применен для нахождения произведения любой пары конечных чисел. Подобных примеров можно найти немало, причем не только в математике — термин «*алгоритм*» стал общеупотребимым. В связи с этим возникает вопрос: можно ли построить общее и точное определение алгоритма (понятие «*любой алгоритм*»), например для того, чтобы, пользуясь им, различить, является ли алгоритмом какая-то совокупность указаний или нет? На уровне здравого смысла можно сказать, что *алгоритм — это точно определенная (однозначная) последовательность простых (элементарных) действий, обеспечивающая решение любой задачи из некоторого*

*класса.* Однако данное утверждение нельзя принять в качестве строгого определения алгоритма, поскольку в нем использованы другие неопределенные понятия — однозначность, элементарность и пр. Понятие можно уточнить, указав перечень *общих свойств*, которые характерны для алгоритмов. К ним относятся:

1. *Дискретность* алгоритма означает, что *алгоритм разделен на отдельные шаги (действия), причем, выполнение очередного шага возможно только после завершения всех операций на предыдущем шаге.* При этом набор промежуточных данных *конечен* и получается по определенным правилам из данных предыдущего шага.

2. *Детерминированность* алгоритма состоит в том, *что совокупность промежуточных величин да любом шаге однозначно определяется системой величин, имевшихся на предыдущем шаге.* Данное свойство означает, что результат выполнения алгоритма не зависит от того, кто (или что) его выполняет (т. е. от *исполнителя алгоритма*), а определяется только входными данными и шагами (последовательностью действий) самого алгоритма.

3. *Элементарность* шагов: *закон получения последующей системы величин из предыдущей должен быть простым и локальным.* Какой шаг (действие) можно считать элементарным, определяется особенностями исполнителя алгоритма.

4. *Направленность* алгоритма: *если способ получения последующих величин из каких-либо исходных не приводит к результату, то должно быть указано, что следует считать результатом исполнения алгоритма.*

5. *Массовость* алгоритма: *начальная система величин может выбираться из некоторого множества.*

Последнее свойство означает, что один алгоритм, т. е. одна и та же последовательность действий в общем случае может применяться для решения некоторого класса (т. е. многих) задач. Для практики и, в частности, решения задачи на компьютере, это свойство существенно, поскольку, как правило, пользовательская ценность программы оказывается тем выше, чем больший круг однотипных задач она позволяет решить. Однако для построения алгоритмической теории это свойство не является существенным и обязательным.

Понятие алгоритма, в какой-то мере определяемое перечислением свойств 1–5, также нельзя считать строгим, поскольку

в формулировках свойств использованы термины «*величина*», «*способ*», «*простой*», «*локальный*» и другие, точный смысл которых не установлен. В дальнейшем данное определение мы будем называть *нестрогим* (иногда его называют *интуитивным*) понятием алгоритма.

Возникает вопрос: так ли уж важно и необходимо иметь точное определение алгоритма, если и без него возможно составление и применение алгоритмов (причем, даже без употребления самого термина)? Тем более, что интуитивное понятие алгоритма хотя и не обладало строгостью, но было настолько ясным, что практически до XX в. не возникало ситуаций, когда математики разошлись бы в суждениях относительно того, является ли алгоритмом тот или иной конкретный процесс. Положение существенно изменилось в начале XX в., когда были сформулированы такие проблемы, алгоритмическое решение которых являлось не очевидным. Действительно, для доказательства существования алгоритма необходимо просто решить данную задачу, пользуясь набором известных приемов, или в их отсутствии предложить новые приемы — в таких ситуациях достаточно и интуитивного понятия алгоритма, чтобы удостовериться в том, что описанный процесс есть алгоритм. Гораздо сложнее доказать факт невозможности построения алгоритма решения некоторой задачи (или класса задач) — без точного определения алгоритма эта проблема теряет смысл.

Другим основанием, потребовавшим построения точного определения алгоритма, явилось неопределенность понятия «*элементарность шага*» при выполнении алгоритмических действий. Пока математика изучала числовые объекты, действия с ними сводились к некоторой последовательности вычислительных операций и элементарными считались арифметические операции, а также несколько логических, связанных с проверкой отношений между величинами (равенство, неравенство, больше, меньше и пр). Однако позднее математика перешла к исследованию свойств и действий со сложными объектами — векторами, матрицами, множествами, функциями — и понятие элементарности шага алгоритма перестало быть очевидным. Например, можно ли считать элементарным шагом взятие интеграла или транспонирование матрицы?

Наконец, в тех ситуациях, когда задача допускает построение нескольких алгоритмов решения, то с теоретической и с практической точек зрения оказывается существенным вопрос их сопос-

тавления и выбора наиболее эффективного, что также невозможно без строгого определения алгоритма.

Таким образом, возникла необходимость в точном определении понятия «*любой алгоритм*», т. е. максимально общем определении, под которое подходили бы все мыслимые виды алгоритмов. В 20-е гг. XX в. построение определения алгоритма стало одной из центральных математических проблем. Определение это, с одной стороны, должно было соответствовать сущности интуитивного понятия алгоритма, а с другой стороны, быть формально строгим. Попытки формулировки такого понятия привели к появлению в 30-х гг. XX века теории алгоритмов как самостоятельного направления математики, которая совместно с математической логикой изучает основные средства математики — методы доказательств, способы построения аксиоматических теорий, свойства математических процедур и пр. Когда же в 1940-е — 1950-е гг. началось бурное развитие вычислительной техники и наук, связанных с ее функционированием и использованием, то выяснилось, что в основе этих наук также должна лежать *теория алгоритмов*, поскольку *компьютер может реализовать только такие процедуры, которые представимы в виде алгоритмов*. Любая программа есть не что иное, как запись алгоритма на языке, который может «понять» исполнитель — компьютер. Таким образом, с практической точки зрения также представляется важным уточнение понятия алгоритма.

Уточнение понятия алгоритма выполняется в рамках *алгоритмических моделей*. Модель определяет набор средств, использование которых допустимо при решении задачи, т. е. перечень элементарных шагов, способы определения следующего шага и т. п. Алгоритмические модели могут быть *теоретическими* и *практическими*. С теоретической точки зрения наибольший интерес представляют модели, которые, с одной стороны, были бы *универсальными*, т. е. позволяли бы описать любой алгоритм, а с другой стороны — максимально *простыми*, т. е. использовали бы минимум средств решения задачи. Требование простоты важно для того, чтобы выделить действительно необходимые элементы и свойства алгоритма и обеспечить доказательство общих утверждений об этих свойствах. В практических и прикладных моделях более значимым является удобство программирования и эффективность вычислений, поэтому их средства — набор элементарных шагов и пр. — намного богаче и сложнее, что затрудняет теоретический анализ.

В теоретических подходах к построению строгого определения понятия алгоритма исторически выделились три основных направления.

*Первое направление* связано с рассмотрением алгоритмов, позволяющих вычислить значение числовых функций, зависящих от целочисленных значений аргументов — такие функции получили название *вычислимых*. Понятие вычислимой функции не является строгим, как и понятие алгоритма. Однако, благодаря работам А. Черча, К. Геделя, С. Клини, была обоснована тождественность класса всюду определенных вычислимых функций с классом *частично рекурсивных* функций, который определяется строго. Это позволило свести проблему алгоритмической разрешимости к доказательству возможности (или невозможности) построения рекурсивной функции, решающей задачу. Именно этим путем А. Черчу удалось доказать неразрешимость одной из проблем математической логики — исчисления истинности предикатов.

*Второе направление* связано с машинной математикой. Основная идея этого направления состоит в том, что алгоритмические процессы — это процессы, которые может совершать соответствующим образом устроенная «*машина*». В соответствии с этой идеей были описаны в точных математических терминах довольно узкие классы машин, однако при этом было доказано, что посредством этих машин можно осуществить или имитировать все алгоритмические процессы, которые фактически когда-либо описывались математиками. Данный подход развивался в работах Э. Поста и А. Тьюринга. Доказательство алгоритмической разрешимости задачи сводится к доказательству существования машины, ее решающей.

*Третье направление* связано с понятием нормальных алгоритмов, введенным и разработанным российским математиком А.А. Марковым в начале 50-х гг. XX в.

В последующих разделах все три направления будут рассмотрены подробнее. Но прежде чем перейти к этому, хотелось бы отметить то обстоятельство, что строго формализованный подход в определении понятия «*алгоритм*» используется лишь непосредственно в самой математической теории алгоритмов, где исследуются общие свойства алгоритмов, проводится доказательство алгоритмической разрешимости и пр. В практических же приложениях, в том числе в информатике, строгая формализация может привести к значительному усложнению задачи; в то

же время можно указать ряд ситуаций, в которых допустимо отступление от нее.

1. *Применение исполнителей, способных выполнять сложные команды.* Определение термина «*исполнитель алгоритма*» достаточно очевидно:

> ***Исполнитель алгоритма** — это субъект или устройство, которые способны правильно интерпретировать описание алгоритма и выполнить содержащийся в нем перечень действий.*

Указания по выполнению действий для каждого исполнителя формулируются посредством некоторого *языка*, включающего набор служебных слов, обозначающих действия (*команды*), а также синтаксические правила их объединения. Совокупность допустимых команд образует *систему команд исполнителя* (СКИ). Различаться СКИ разных исполнителей могут детальностью описания действий. Как будет показано ниже, элементарным (простейшим) действием при обработке дискретной информации является замена одного знака другим. Однако можно построить абстрактное или реальное устройство, которое будет способно выполнять целую *цепочку* подобных элементарных действий по заложенному в него правилу — некое *комплексное* (интегральное) действие. При построении алгоритма для такого исполнителя разработчику достаточно указать последовательность интегральных команд, а их преобразование в цепочку истинно элементарных шагов будет выполняться самим исполнителем. Такая «многоступенчатая» алгоритмизация широко применяется при управлении компьютером. Истинно элементарными следует считать действия процессора (хотя их общее число у современных процессоров достигает нескольких сотен и даже тысяч) — их называют *машинными командами*, а их обозначения — *машинными кодами*. Первым (низшим) уровнем, на котором происходит отход от машинных кодов, является *код ассемблера* — внутренний (т. е. аппаратно-зависимый) язык. Объединения элементарных действий в сложные команды на этом уровне еще не происходит и общее количество команд ассемблера совпадает с числом команд процессора, однако используется символьная форма представления машинных кодов и регистров процессора — *мнемоники*, что удобнее для пользователя, чем двоичный машинный код. Перевод мнемоник в машинные команды осуществляет программа — *ассемблер*; именно с ней имеет дело программист как с исполнителем. Команды, объединя-

ющие ряд элементарных действий, появляются в языках программирования высокого уровня, например в тексте программы достаточно написать «*Write*», а уже транслятор языка переведет ее в последовательность элементарных шагов: прерываний, пересылок и пр. По отношению к программисту исполнителем в этом случае оказывается транслятор языка программирования. Еще большую степень интеграции элементарных команд может обеспечить прикладная программа, которая является исполнителем по отношению к конечному пользователю. СКИ такого исполнителя включает все команды управления, представленные в виде меню, экранных кнопок, окон настройки и других элементов интерфейса. Использование одной команды может вызвать цепочку сложных действий, например выравнивание многих строк текста.

Таким образом, при записи алгоритмов возможны ситуации, когда язык представления алгоритма является формальным, но в нем используются сложные команды, которые самим исполнителем переводятся на уровень истинно элементарных действий.

2. *Допустимость нестрогой формализации представления алгоритмов*, если исполнителем является человек. Человек обладает собственным мышлением и знаниями, опираясь на которые он может компенсировать неточности описания алгоритма, выполнить действия и добиться результата. Подобные алгоритмы следует считать еще менее строгими, чем те, что были рассмотрены в начале раздела, поскольку они, как правило, не обладают всеми перечисленными свойствами. Примерами могут служить кулинарные рецепты, инструкции по применению бытовых приборов, алгоритмы решения школьных задач.

3. *Расширение применимости понятия алгоритма* на последовательность любых дискретных действий. По определению алгоритм должен быть обязательно связан с обработкой дискретной информации. Однако этот же термин используется и для обозначения действий по управлению исполнителем, напрямую не производящим преобразование информации. Например, в школьном курсе информатики широко применяются учебные исполнители «*Чертежник*», «*Паркетчик*», «*Черепашка*», СКИ которых включает перемещение по экрану и выполнение некоторых операций («*начертить линию*», «*положить плитку*» и т. п.). То же относится к инструкциям по управлению какими-либо агрегатами и устройствами.

## 9.2. Рекурсивные функции

Для дальнейшего рассмотрения нам понадобится ряд определений. Пусть имеются два множества $X$ и $Y$.

> *Если некоторым элементам множества $X$ поставлены в соответствие однозначно определенные элементы множества $Y$, то говорят, что задана* ***частичная функция*** *из $X$ в $Y$.*

> *Совокупность тех элементов множества $X$, у которых есть соответствующие элементы в $Y$, называется* ***областью определения функции,*** *а совокупность тех элементов $Y$, которые соответствуют элементам $X$, называются* ***совокупностью значений функции.***

> *Если область определения функции из $X$ в $Y$ совпадает с множеством $X$, то функция называется* ***всюду определенной****.*

Исходная идея построения точного определения алгоритма, опирающегося на понятие рекурсивной функции, состоит в том, что любые данные (безусловно, дискретные) можно закодировать натуральными числами в некоторой системе счисления и тогда всякое их преобразование сведется к последовательности вычислительных операций, а результат обработки также будет представлять собой целое число. В данном подходе любой алгоритм, единый для данной числовой функции, вычисляет ее значение, а его элементарными шагами оказываются обычные арифметические и логические операции. Такие функции получили название *вычислимых.*

Пусть имеется класс функций типа $y(x_1, x_2, \ldots, x_n)$, особенностями которых является то, что все аргументы функции $x_1, \ldots, x_n$ целочисленны и значение функции $y$ также выражается целым числом. Другими словами, рассматриваются функции, аргументы и значения которых дискретны.

> *Функция $y(x_1, x_2, \ldots, x_n)$ называется* ***эффективно вычислимой,*** *если существует алгоритм, позволяющий вычислить ее значение по известным значениям аргументов.*

Поскольку понятие алгоритма в этом определении берется в интуитивном смысле, то и понятие эффективно вычислимой функции оказывается интуитивным. Тем не менее, при переходе от алгоритмов к вычислимым функциям возникает одно очень

существенное обстоятельство. Совокупность процессов, удовлетворяющих условиям 1–5 из п. 9.1 и, следовательно, подпадающих под интуитивное понятие алгоритма, весьма обширна и мало обозрима. Напротив, совокупность вычислимых функций для самых разных пониманий процессов, удовлетворяющих перечисленным условиям, оказалась одной и той же и притом легко описываемой в обычных математических терминах. Эта точно описанная совокупность числовых функций, совпадающая с совокупностью всех вычислимых функций при самом широком до сих пор известном понимании алгоритма, носит название совокупности *рекурсивных функций.*

Любая алгоритмическая модель, в том числе рекурсивные функции, должна предусматривать определение элементарных шагов алгоритма и способов построения из них какой-то последовательности преобразований, обеспечивающих необходимую последовательность обработки данных. В рекурсивной модели такими «элементарными шагами» являются так называемые *простейшие* числовые функции $S^1, 0^n$ и $I^n_m$, комбинацией которых строятся все более сложные и которые *определяются* следующим образом:

- $S^1(x) = x + 1$ — это *одноместная* (т. е. имеет один аргумент) *функция непосредственного следования*;
- $0^n(x_1, x_2, \ldots, x_n) = 0$ — это $n$-местная функция *тождественного равенства нулю*;
- $I^n_m(x_1, ..., x_n) = x_m$, $1 \leqslant m \leqslant n$; $n = 1, 2, \ldots$, — $n$-местная функция *тождественного повторения значения одного из своих аргументов.*

Перечисленные простейшие функции всюду определены и интуитивно вычислимы. Над ними определяются операции (в дальнейшем они называются *операторами*), обладающие тем свойством, что их применение к функциям, вычислимым в интуитивном смысле, порождает новые функции, также заведомо вычислимые в интуитивном смысле. Частичные функции, которые можно получить при помощи этих операторов из простейших функций называются частично рекурсивными. *Гипотеза Черча* состоит в том, *что класс построенных таким образом частично рекурсивных функций совпадает с классом функций, допускающим алгоритмическое вычисление.*

Переходим к рассмотрению операторов, обеспечивающих преобразование простейших функций.

**Суперпозиция частичных функций.** Пусть $m$-местные функции

$$f_1(x_1,\ldots,x_m); f_2(x_1,\ldots,x_m);\ldots,f_n(x_1,\ldots,x_m)$$

подставляются в $n$-местную функцию $g(x_1,\ldots,x_n)$. В результате получается $n$-местная функция

$$h(x_1,\ldots,x_n)=g(f_1(x_1,\ldots,x_m);\ldots,f_n(x_1,\ldots,x_m))$$

Говорят, что функция $h$ получена из функций $g,f_1,\ldots,f_n$ *суперпозицией* (или подстановкой). Символически такая подстановка обозначается следующим образом: $S^{n+1}(g,f_1,\ldots,f_n)$, где индекс сверху обозначает количество функций, подставляемых в качестве аргументов.

Если мы умеем вычислять функции $g,f_1,\ldots,f_n$, то функция $h$ также может быть вычислена. Ясно также, что если все функции $g,f_1,\ldots,f_n$ всюду определены, то и функция $h$ также всюду определена. Таким образом, если функции $g,f_1,\ldots,f_n$ интуитивно вычислимы, то будет интуитивно вычислимой и функция $h$.

**Пример 9.1.**

Найти значение $S^2(S^1,0^1)$.

Для этого значение простейшей функции $0^1$ должно быть подставлено в $S^1(x)=x+1$. Но $0^1(x)=0$, следовательно, $h(x)=S^2(S^1,0^1)=S^1(0^1)=$ $=0+1=1$.

**Пример 9.2.**

Найти значение $S^3(I_2^2,I_1^1,0^1)$.

В этом случае конечная функция будет двуместной ($n=3-1=2$), следовательно $h(x_1,x_2)=I_2^2(I_1^1,0^1)=I_2^2(x_1,0)=0$.

**Примитивная рекурсия.** Пусть заданы какие-либо числовые частичные функции: $n$-местная $g(x_1,\ldots,x_n)$ и $(n+2)$-местная $h(x_1,\ldots,x_n,k,y)$. Говорят, что $(n+1)$-местная частичная функция $f$ образуется из функций $g$ и $h$ посредством *примитивной рекурсии*, если для всех натуральных значений $x_1,\ldots,x_n,y$ справедливо:

$$\begin{aligned} &f(x_1,\ldots,x_n,0)=g(x_1,\ldots,x_n);\\ &f(x_1,\ldots,x_n,y+1)=h(x_1,\ldots,x_n,y,f(x_1,\ldots,x_n,0)) \end{aligned} \tag{9.1}$$

Поскольку областью определения функций является множество всех натуральных чисел, частичная функция $f$, удовлетворяющая условиям (9.1), существует для каждых частичных функций $g$ и $h$ и функция эта будет единственной. Условия (9.1) за-

дают также последовательность определения значений $f$ на различных шагах рекурсии:

$$\begin{aligned} &f(x_1,\ldots,x_n,0) = g(x_1,\ldots,x_n);\\ &f(x_1,\ldots,x_n,1) = h(x_1,\ldots,x_n,1,f(x_1,\ldots,x_n,0));\\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\ &f(x_1,\ldots,x_n,m+1) = h(x_1,\ldots,x_n,m+1,f(x_1,\ldots,x_n,m)). \end{aligned} \tag{9.2}$$

Символически примитивная рекурсия обозначается $f = R(g,h)$; в этой записи $R$ рассматривается как символ двуместной частичной операции, определенной на множестве всех частичных функций. Из соотношений (9.2) вытекает, в частности, что если $g$ и $h$ являются всюду определенными, то и $f$ также является всюду определенной. Из (9.2) видно также то важное обстоятельство, что если мы умеем находить значения функций $g$ и $h$, то значения функции $f(a_1,\ldots,a_n,m+1)$ можно вычислять «механически», находя последовательно значения на предыдущих шагах.

Введем определение.

*Частичная функция $f(x_1,\ldots,x_n)$ называется **примитивно рекурсивной**, если ее можно получить с помощью конечного числа операций суперпозиции и примитивной рекурсии, исходя лишь из простейших функций $S^1, 0^n$ и $I^n_m$.*

Если операции суперпозиции и примитивной рекурсии применить к всюду определенным функциям, в результате будет получена также всюду определенная функция. В частности, все *примитивно рекурсивные функции всюду определены.*

**Пример 9.3.**

Доказать, что двуместная функция $f(x,y) = x+y$ является примитивно-рекурсивной.

Данная функция может быть представлена в форме (9.1):

$$x + 0 = x = I^1_1(x);$$
$$x + (y+1) = (x+y) + 1 = S^1(x+y).$$

Следовательно, функция $f(x,y)$ образуется из примитивно рекурсивных функций операцией примитивной рекурсии и, следовательно, она сама примитивно рекурсивна.

**Пример 9.4.**

Найти значение функции $f(3,2)$, если она задана следующими соотношениями:

$$f(0,x) = 0; \quad f(y+1,x) = f(y,x) + x.$$

В данном случае $g(x) = 0$, $h(x, y, z) = y + z$. Так как $f(0, x) = g(x) = 0$ при любом $x$, то и $f(0, 2) = 0$, а другие значения можно вычислить последовательно:

$$f(1, 2) = h(1, 0, 2) = 0 + 2 = 2;$$
$$f(2, 2) = h(2, 2, 2) = 2 + 2 = 4;$$
$$f(3, 2) = h(3, 4, 2) = 4 + 2 = 6.$$

Несложно доказать, что в данном примере $f(x, y) = xy$.

**Операция минимизации.** Пусть задана некоторая функция $f(x, y)$. Зафиксируем значение $x$ и выясним, при каком $y$ значение $f(x, y) = 0$. Более сложной оказывается задача отыскания *наименьшего* из тех значений $y$, при которых $f(x, y) = 0$. Поскольку результат решения такой задачи, очевидно, зависит от $x$, то и наименьшее $y$ является функцией $x$. Примем обозначение

$$\varphi(x) = \mu_y[f(x, y) = 0]$$

(читается: «*наименьшее $y$ такое, что $f(x, y) = 0$*», а $\mu_y$ называется *$\mu$-оператором* или *оператором минимизации*).

Подобным же образом определяется функция многих переменных:

$$\varphi(x_1, \ldots x_n) = \mu_y[f(x_1, \ldots x_n, y) = 0].$$

Для вычисления функции $\varphi$ можно предложить следующую процедуру:

1. Вычисляем $f(x_1, \ldots x_n, 0)$; если значение равно нулю, то полагаем $\varphi(x_1, \ldots x_n) = 0$. Если $f(x_1, x_n, 0) \neq 0$, то переходим к следующему шагу.

2. Вычисляем $f(x_1, \ldots x_n, 1)$; если значение равно нулю, то полагаем $\varphi(x_1, \ldots x_n) = 1$. Если $f(x_1, x_n, 0) \neq 0$, то переходим к следующему шагу. И т. д.

Если окажется, что для всех $y$ функция $f(x_1, \ldots x_n, 0) \neq 0$, то функция $\varphi(x_1, \ldots x_n)$ считается неопределенной.

**Пример 9.5.**

Рассмотрим функцию $f(x, y) = x - y$, которая может быть получена с помощью оператора минимизации:

$$f(x, y) = \mu_z(y + z = x) = \mu_z[I_3^2(x, y, z) + I_2^3(x, y, z) = I_3^1(x, y, z)].$$

Вычислим, например, $f(7, 2)$, т. е. значение функции при $y = 2$ и $x = 7$. Положим $y = 2$, а $x$ будем придавать последовательные значения:

$$z = 0; \quad 2 + 0 = 2 \neq 7;$$
$$z = 1; \quad 2 + 1 = 3 \neq 7;$$

$$z = 2; \quad 2 + 2 = 4 \neq 7;$$
$$z = 3; \quad 2 + 3 = 5 \neq 7;$$
$$z = 4; \quad 2 + 4 = 6 \neq 7;$$
$$z = 5; \quad 2 + 5 = 7 = 7.$$

Таким образом, найдено значение функции $f(7, 2) = 5$.

Введем определение:

*Частичная функция $f(x_1, \ldots, x_n)$ называется **частично рекурсивной**, если ее можно получить конечным числом операций суперпозиции, примитивной рекурсии и минимизации, исходя лишь из простейших функций $S^1$, $0^n$ и $I_m^n$.*

Класс частично рекурсивных функций шире класса примитивно рекурсивных функций, так как все примитивно рекурсивные функции являются всюду определенными, а среди частично рекурсивных функций встречаются функции не всюду определенные, а также нигде не определенные.

Понятие частично рекурсивной функции является одним из главных понятий теории алгоритмов. Значение его состоит в следующем. С одной стороны, каждая стандартно заданная частично рекурсивная функция вычислима путем некоторой процедуры механического характера, отвечающей интуитивному представлению об алгоритмах. С другой стороны, какие бы классы точно очерченных алгоритмов ни строились, во всех случаях неизменно оказывалось, что вычислимые посредством них числовые функции являлись частично рекурсивными. Поэтому общепринятой является научная гипотеза, формулируемая как *тезис Черча*:

**Класс алгоритмически (или машинно-) вычислимых частичных числовых функций совпадает с классом всех частично рекурсивных функций.**

Этот тезис дает алгоритмическое истолкование понятие частично рекурсивной функции. Его нельзя доказать, поскольку он связывает нестрогое математическое понятие интуитивно вычислимой функции со строгим математическим понятием частично рекурсивной функции. Однако исследования, проводившиеся весьма многими математиками в течение нескольких десятилетий, выявили полную целесообразность считать *понятие частично рекурсивной функции научным эквивалентом интуитивного понятия вычислимой частичной функции.*

Тезис Черча оказался достаточным, чтобы придать необходимую точность формулировкам алгоритмических проблем и в ряде

случаев сделать возможным доказательство их неразрешимости. Причина заключается в том, что обычно в алгоритмических проблемах математики речь идет не об алгоритмах, а о вычислимости некоторых специальным образом построенных функций. В силу тезиса Черча вопрос о вычислимости функции равносилен вопросу о ее рекурсивности. Понятие рекурсивной функции строгое. Поэтому обычная математическая техника позволяет иногда непосредственно доказать, что решающая задачу функция не может быть рекурсивной. Именно этим путем самому Черчу удалось доказать неразрешимость основной алгоритмической проблемы логики предикатов — проблемы тождественной истинности формул исчисления первой ступени.

## 9.3. Алгоритм как абстрактная машина

### 9.3.1. Общие подходы

Точное описание класса частично рекурсивных функций вместе с тезисом Черча дает одно из возможных решений задачи об уточнении понятия алгоритма. Однако это решение не вполне прямое, так как понятие вычислимой функции является вторичным по отношению к понятию алгоритма. Спрашивается, нельзя ли уточнить непосредственно само понятие алгоритма и уже затем при его помощи определить точно и класс вычислимых функций? Такое направление поиска привело к построению иного, нежели рекурсивные функции, класса моделей алгоритма. Основная его идея состоит в том, что алгоритмические процессы — это процессы, которые может осуществлять определенным образом устроенная машина, моделирующая тем самым выполнение отдельных операций человеком. Функционирование такой машины и есть выполнение некоторого алгоритма. Исходя из свойств алгоритма (п. 9.1), можно сформулировать общие требования к таким машинам:

- характер их функционирования должен быть *дискретным*, т. е. состоять из отдельных шагов*, каждый из которых выполняется только после завершения предыдущего;
- действия должны быть детерминированы, т. е. шаги выполняются в строгом порядке, а их результат определяется самим шагом и результатами предыдущих шагов;

---

* В дальнейшем указание по выполнению шага будем называть *командой*.

- перед началом работы машине предоставляются исходные данные из области определения алгоритма;
- за конечное число шагов работы машины должен быть получен результат (или информация о том, что считать результатом);
- машина должна быть универсальной, т. е. такой, чтобы с ее помощью можно было бы выполнить любой алгоритм.

Чем проще структура (т. е. устройство) описанной машины и чем элементарнее ее шаги, тем больше оснований считать, что ее работа и есть выполнение алгоритма. Чтобы ответить на вопрос, какие шаги работы машины следует отнести к элементарным, вернемся к тому обстоятельству, что нас интересует преобразование информации, представленной с помощью некоторого *конечного алфавита*. Требование конечности алфавита является очевидным следствием того обстоятельства, что решение должно быть получено за конечное число шагов. Если информация не представлена в дискретной форме, например вещественное число, то его обработка в общем случае может содержать бесконечное число шагов (например, нахождение цифр числа $\pi$ или извлечение квадратного корня из числа 2). Таким образом, алгоритм оказывается *конечной* последовательностью действий, производимых над данными, представленными с помощью конечного алфавита. С учетом сказанного становится понятным определение алгоритма, которое дает В.М. Глушков [14, с. 38]:

> ***Алгоритм** — это любая конечная система правил преобразования информации (данных) над любым конечным алфавитом.*

Пусть исходные данные из области определения алгоритма представлены посредством алфавита $A$ и образуют при этом конечную последовательность знаков $\{a_1 \dots a_n\}$ — такая последовательность называется *словом*. В результате выполнения алгоритма сформируется новое слово $\{b_1 \dots b_m\}$, представленное, в общем случае, в другом алфавите $B$. На первый взгляд для проведения такого преобразования в качестве элементарных выделяются следующие операции (шаги):

1) замена одного знака исходного слова $a_i$ знаком $b_j$ из алфавита $B$;

2) удаление знака исходного слова;

3) добавление к исходному слову знака из алфавита $B$.

Однако если в алфавиты включен знак, имеющий смысл *пустого знака*, добавление которого к слову слева или справа не из-

меняет этого слова (аналог числового нуля), то легко видеть, что операция (2) есть не что иное, как замена $a_i$ пустым знаком, а операция (3) есть замена пустого знака знаком $b_j$. Таким образом, все возможные алфавитные преобразования сводится к операции (1) — замене одного знака другим. Именно по этой причине функционирование абстрактной машины сводится к тому, что она обозревает символы (т. е. считывает и распознает их), записанные в памяти (в этом качестве выступает бесконечная лента), и в зависимости от своего состояния и того, каков обозреваемый символ, она заменяет его другим символом; после этого она переходит в новое состояние, читает следующий символ и т. д. до команды о прекращении работы. Поскольку подобные машины являются чисто модельным, теоретическим построением, они получили название *абстрактных машин* и рассматриваются в качестве одной из возможных универсальных алгоритмических систем.

Концепция алгоритма как абстрактной машины была выдвинута практически одновременно (1936–1937 гг.) английским математиком Аланом Тьюрингом и его американским коллегой Эмилем Постом. Высокая значимость их построений еще и в том, что они в абстрактной форме предвосхитили основные принципиальные черты реальных устройств по обработке данных (вычислительных машин), которые появились лишь спустя 8–9 лет.

### 9.3.2. Алгоритмическая машина Поста

На самом деле, Пост, в отличие от Тьюринга, не пользовался термином «*машина*», а называл свою модель *алгоритмической системой*. Мы же, как принято в литературе, все же будем говорить о *машине Поста*, подчеркивая тем самым единство подходов обоих авторов [40].

Абстрактная машина Поста состоит из бесконечной ленты, разделенной на равные секции, а также считывающе-записывающей головки. Каждая секция может быть либо пуста (т. е. в нее ничего не записано), либо заполнена (*отмечена*, т. е. в нее записана *метка*). Вводится понятие *состояние ленты* как информация о том, какие секции пусты, а какие отмечены (по-другому: состояние ленты — это распределение меток по секциям, т. е. это функция, которая каждому числовому номеру секции ставит в соответствие либо метку, либо знак «*пусто*»). Естественно, в процессе работы машины состояние ленты меняется. Состояние ленты и информация о положении головки характеризуют *состояние машины* Поста.

Таблица 9.1

| № | Команда | Запись команды | Описание действий машины |
|---|---|---|---|
| 1 | Шаг вправо | $X \to y$ | Сдвиг головки на одну секцию вправо |
| 2 | Шаг влево | $X \leftarrow y$ | Сдвиг головки на одну секцию влево |
| 3 | Установить метку | $XMy$ | В обозреваемую секцию ставится метка |
| 4 | Стереть метку | $XCy$ | Из обозреваемой секции удаляется метка |
| 5 | Передача управления | $x \nearrow y_1$, $x \searrow y_2$ | При отсутствии метки в обозреваемой секции управление передается команде $y_1$, при наличии — команде $y_2$ |
| 6 | Остановка | $x$ стоп | Прекращение работы машины |

Условимся обозначать головку знаком «$\nabla$» над обозреваемой секцией, а метку — знаком «$M$» внутри секции. Пустая секция никакого знака не содержит. За один такт (его называют *шагом*) головка может сдвинуться на одну секцию вправо или влево и поставить или удалить метку. Работа машины Поста заключает в переходе от одного состояния машины к другому в соответствии с заданной программой, которая строится из отдельных команд. Каждая команда имеет следующую структуру: $xKy$, где $x$ — номер исполняемой команды; $K$ — указание о выполняемом действии; $y$ — номер следующей команды (*наследника*).

Система команд машины, включающая 6 действий, представлена в табл. 9.1. Данный перечень должен быть дополнен следующими условиями:

- команда $\langle xMy \rangle$ может быть выполнена *только в пустой секции*;
- команда $\langle xCy \rangle$ может применяться *только к заполненной секции*;
- номер наследника любой команды ($y$) должен соответствовать номеру команды, обязательной имеющейся в данной программе.

Если данные условия не выполняются, происходит *безрезультатная* остановка машины, т. е. остановка без получения запланированного результата. В отличие от этой ситуации остановка по команде $\langle x$ стоп$\rangle$ является *результативной*, т. е. она происходит после того, как результат действия алгоритма получен. Кроме того, возможна ситуация, когда машина не останавливается никогда — это происходит, если ни одна из команд не содержит в качестве последователя номера команды остановки или программа не переходит к этой команде.

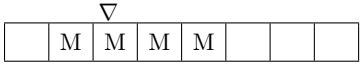
Еще одним исходным соображением является следующее: поскольку знаки любого конечного алфавита могут быть закодированы цифрами, преобразование исходного слова может быть представлено в виде некоторых правил обработки чисел. По этой причине в машине Поста предусматривается только запись (представление) целых положительных чисел.

Целое число $k$ записывается на ленте машины Поста посредством $k+1$ следующих подряд отмеченных секций, т. е. применяется *унарная* система счисления (см. п. 4.1). Соседние записи чисел на ленте разделяются одной или несколькими пустыми секциями. Ниже приведен пример записи чисел 0, 2 и 3.


Круг вычислительных задач, решаемых с помощью машины Поста, весьма широк. Однако как указывалось выше, на уровне элементарных шагов все сводится к постановке или удалению метки и сдвигу головки. В качестве примеров рассмотрим несколько задач, традиционно обсуждаемых при освоении машины Поста. Поскольку вид программы (последовательности команд машины) зависит от начального состояния машины, оно должно быть в явном виде указано в постановке задачи.

**Пример 9.6.**

На ленте записано некоторое число, и головка обозревает одну из помеченных секций (любую). Составить программу прибавления единицы к этому числу. Ситуация иллюстрируется рисунком:




Программа, обеспечивающая решение задачи, состоит из 4-х команд:

$$1 \begin{array}{l} \nearrow 3 \\ \searrow 1 \end{array} \quad 2 \to 1 \quad 3M4 \quad 4 \text{ стоп.}$$

Последовательное исполнение команд 1 и 2 приводит к тому, что головка за два такта работы машины сдвигается на одну позицию вправо. Это передвижение продолжается до тех пор, пока после очередного сдвига под головкой не окажется пустой ячейки — тогда по команде 3 в нее будет поставлена метка и по команде 4 машина остановится.

**Пример 9.7.**

На ленте записано некоторое число, и головка обозревает одну из свободных секций (любую) левее записи. Составить программу прибавления единицы к этому числу.

Программа:

$$1 \begin{cases} \nearrow 2 \\ \searrow 3 \end{cases} \quad 2 \to 1 \quad 3 \leftarrow 4 \quad 5M5 \quad 5 \text{ стоп.}$$

Комментарий к работе программы подобен приведенному выше, с той лишь разницей, что метка ставится перед исходным числом.

Можно показать (это предлагается сделать в контрольных заданиях к данному разделу), что с помощью машины Поста реализуются (хотя и довольно громоздко) все арифметические действия над числами в унарной системе счисления*. Числа же, как было показано ранее, могут быть использованы для кодирования любой дискретной информации. В частности, состояние ленты можно представить словом в двоичном алфавите, где 0 будет соответствовать пустой секции, а 1 — отмеченной. В процессе работы меняется состояние ленты и, следовательно, от исходного слова происходит переход к выходному, представленному в том же двоичном алфавите.

### 9.3.3. Алгоритмическая машина Тьюринга

Машина Тьюринга состоит из трех частей: ленты, считывающе-записывающей головки и логического устройства (рис. 9.1).

Лента выступает в качестве внешней памяти; она считается неограниченной (бесконечной) — уже это свидетельствует о том, что машина Тьюринга является модельным устройством, поскольку ни одно реальное устройство не может обладать памятью бесконечного размера.

Как и в машине Поста, лента разбита на отдельные ячейки, однако в машине Тьюринга неподвижной является головка, а лента передвигается относительно нее вправо или влево. Другим

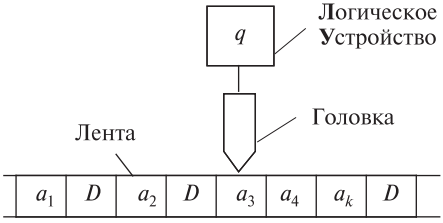



**Рис. 9.1.** Элементы машины Тьюринга

* Машина Поста обеспечивает весьма неплохую и полезную практику программирования. Недостатком оказывается чисто теоретический (т. е. непроверяемый) характер программ, однако он достаточно легко преодолевается, если построить эмуляцию машины на каком-либо языке программирования.

отличием является то, что она работает не в двоичном, а некотором произвольном конечном алфавите $A = \{\Delta, a_1, \ldots, a_n\}$ — этот алфавит называется *внешним*. В нем выделяется специальный символ — $\Delta$, называемый *пустым знаком*, — его посылка в какую-либо ячейку стирает тот знак, который до этого там находился, и оставляет ячейку пустой. В каждую ячейку ленты может быть записан лишь один символ. Информация, хранящаяся на ленте, изображается *конечной* последовательностью знаков внешнего алфавита, отличных от пустого знака.

Головка всегда расположена над одной из ячеек ленты. Работа происходит тактами (шагами). Система исполняемых головкой команд предельно проста: на каждом такте она производит замену знака в обозреваемой ячейке $a_i$ знаком $a_j$. При этом возможны сочетания:

- $j = i$ — это означает, что в обозреваемой ячейке знак не изменился;
- $i \neq 0$, $j = 0$ означает, что хранившийся в ячейке знак заменяется пустым, т. е. стирается;
- $i = 0$, $j \neq 0$ означает, что пустой знак заменяется непустым (с номером $j$ в алфавите), т. е. выполняется вставка знака;
- $i \neq j \neq 0$ соответствует замене одного знака другим.

Таким образом, в машине Тьюринга реализуется система предельно простых команд обработки информации, о которых шла речь в п. 9.3.1. Эта система команд обработки дополняется также предельно простой системой команд перемещений ленты: на ячейку влево, на ячейку вправо и остаться на месте, т. е. адрес обозреваемой ячейки в результате выполнения команды может либо измениться на 1, либо остаться неизменным. Однако, хотя фактически происходит перемещение ленты, обычно рассматривается *сдвиг головки* относительно обозреваемой секции — по этой причине команда сдвига ленты влево обозначается $R$ («*Right*»), сдвига вправо — $L$ («*Left*»), отсутствие сдвига — $S$ («*Stop*»). В дальнейшем мы будем говорить именно о сдвиге головки и считать $R, L$ и $S$ командами ее движения. Элементарность этих команд означает, что при необходимости обращения к содержимому некоторой ячейки, она отыскивается только посредством цепочки отдельных сдвигов на одну ячейку. Разумеется, это значительно удлиняет процесс обработки, зато позволяет обойтись без нумерации ячеек и использования команд перехода по адресу, т. е. сокращает количество истинно элементарных шагов, что важно в теоретическом отношении.

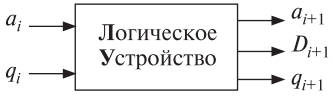



**Рис. 9.2.** Логическое устройство машины Тьюринга

Обработка информации и выдача команд на запись знака, а также сдвига ленты в машине Тьюринга выполняются *логическим устройством* (ЛУ). ЛУ может находиться в одном из состояний, которые образуют конечное множество и обозначаются $Q = \{q_1, \ldots, q_m, z\}$, причем состояние $z$ соответствует завершению работы, а $q_1$ является начальным (исходным). $Q$ совместно со знаками $R, L, S$ образуют *внутренний алфавит* машины. ЛУ имеет два входных канала $(a_i, q_i)$ и три выходных $(a_{i+1}, q_{i+1}, D_{i+1})$ (рис. 9.2).

Таким образом, команда, исполняемая машиной Тьюринга, имеет структуру $a_i q_i \to a_{i+1} D_{i+1} q_{i+1}$.

Понимать схему необходимо следующим образом: на такте $i$ на один вход ЛУ подается знак из обозреваемой в данный момент ячейки $(a_i)$, а на другой вход — знак, обозначающий состояние ЛУ в данный момент $(q_i)$. В зависимости от полученного сочетания знаков $(a_i, q_i)$ и имеющихся правил обработки (таблице команд) ЛУ вырабатывает и по первому выходному каналу направляет в обозреваемую ячейку новый знак $(a_{i+1})$, подает команду перемещения головки ($D_{i+1}$ из $R$, $L$ и $S$), а также дает команду на вызов следующего управляющего знака $(q_{i+1})$. Таким образом, элементарный шаг (такт) работы машины Тьюринга заключается в следующем: головка считывает символ из обозреваемой ячейки и в зависимости от своего состояния и прочитанного символа выполняет команду, в которой указано, какой символ записать (или стереть) и какое движение совершить. При этом и головка переходит в новое состояние. Схема функционирования такой машины представлена на рис. 9.3.

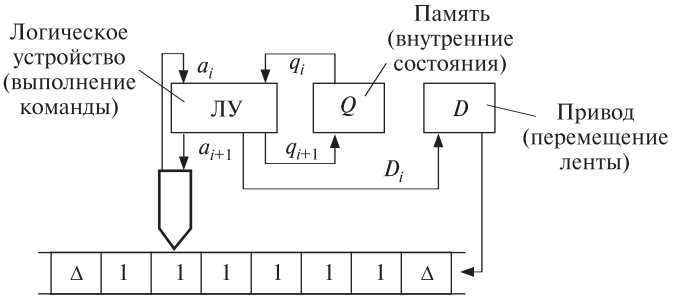



**Рис. 9.3.** Схема функционирования машины Тьюринга

В данной схеме отражено разделение памяти на *внешнюю* и *внутреннюю*. Внешняя представлена, как указывалось, в виде бесконечной ленты — она предназначена для хранения информации, закодированной в символах внешнего алфавита. Внутренняя память представлена двумя ячейками для хранения следующей команды в течение текущего такта: в $Q$ передается из ЛУ и сохраняется следующее состояние ($q_{i+1}$), а в $D$ — команда сдвига ($D_{i+1}$). Из $Q$ по линии обратной связи $q_{i+1}$ поступает в ЛУ, а из $D$ команда поступает на исполнительный механизм, осуществляющий при необходимости перемещение ленты на одну позицию вправо или влево.

Общее правило, по которому работает машина Тьюринга, можно представить следующей записью: $q_i a_j \to q'_i D_k a'_j$, т. е. после обзора символа $a_j$ головкой в состоянии $q_i$, в ячейку записывается символ $a'_j$, головка переходит в состояние $q'_i$, а лента совершает движение $D_k$. Для каждой комбинации $q_i a_j$ имеется *ровно одно* правило преобразования (правил нет только для $z$, поскольку, попав в это состояние, машина останавливается). Это означает, что логический блок реализует функцию, сопоставляющую каждой паре входных сигналов $q_i a_j$ одну и только одну тройку выходных $q'_i D_k a'_j$ — она называется *логической функцией машины* и обычно представляется в виде таблицы (*функциональной схемой машины*), столбцы которой обозначаются символами состояний, а строки — знаками внешнего алфавита. Если знаков внешнего алфавита $n$, а число состояний ЛУ $m$, то, очевидно, общее число правил преобразования составит $nm$.

Конкретная машина Тьюринга задается перечислением элементов множеств $A$ и $Q$, а также логической функцией, которую реализует ЛУ, т. е. набором правил преобразования. Ясно, что различных множеств $A, Q$ и логических функций может быть бесконечно много, т. е. и машин Тьюринга также бесконечно много.

Прежде чем обсуждать функционирование машины Тьюринга, введем еще одно понятие.

> *Совокупность состояний всех ячеек ленты, состояния ЛУ и положение головки называется* ***конфигурацией машины.***

Записать конфигурацию можно следующим образом: $\Delta a_1 q_i a_j \dots a_k \Delta$, которая означает, что в слове из $k$ символов обозревается секция номер $j$ и при этом управляющее устройство находится в состоянии $q_i$. Ясно, что конфигурация машины может содержать любое количество символов внешнего алфавита и

лишь один символ внутреннего. Часто конфигурацию записывают в виде $\alpha_1 q_i \alpha_2$, где $\alpha_1$ — слово на ленте слева от головки, $\alpha_2$ — слово на ленте справа от головки, включая обозреваемый знак. Слева от $\alpha_1$ и справа от $\alpha_2$ лента пуста. Конфигурация, изображенная на рис. 9.1, может быть записана следующим образом: $a_1 \Delta a_2 \Delta q a_3 a_4 a_k$, на рис. 9.3 — 1q1111.

Перед началом работы на пустую ленту записывается исходное слово $\alpha$ конечной длины в алфавите $A$; головка устанавливается над первым символом слова $\alpha$, ЛУ переводится в состояние $q_1$ (т. е. начальная конфигурация имеет вид $q_1\alpha$). Поскольку в каждой конфигурации реализуется только одно правило преобразования, начальная конфигурация однозначно определяет всю последующую работу машины, т. е. всю последовательность конфигураций вплоть до прекращения работы.

В зависимости от начальной конфигурации возможны два варианта развития событий:

1) после конечного числа тактов машина останавливается по команде остановки; при этом на ленте оказывается конечная конфигурация, соответствующая выходной информации;

2) остановки не происходит.

В первом случае говорят, что данная машина применима к начальной информации, во втором — нет. Вся совокупность входных конфигураций, при которых машина обеспечивает получение результата, образуют *класс решаемых задач*. Очевидно, применять машину Тьюринга для задачи, не входящей в класс решаемых, бессмысленно. С другой стороны, во многих случаях возможно расширение класса решаемых задач за счет создания другой машины Тьюринга. Возникает вопрос: можно ли построить такую универсальную машину (хотя бы на теоретическом уровне), которая решала бы любую задачу? Здесь мы подошли к вопросу об алгоритмической разрешимости, который будет исследован позднее.

**Пример 9.8.**

Рассмотрим решение обсуждавшейся в предыдущем разделе задачи о добавлении 1 к унарному числу посредством машины Тьюринга. Внешний алфавит может быть задан множеством $A = \{\Delta, 1\}$, где 1 соответствует заполненной секции, а $\Delta$ — пустому знаку, причем заполненные следуют друг за другом подряд. Внутренний алфавит задается множеством $Q = \{q, z\}$, где $q$ соответствует рабочему состоянию ЛУ, а $z$ — остановке. Набор всех правил преобразования (логическая функция) может быть представлен *функциональной схемой*:

| $A$ | $Q$ | |
|---|---|---|
| | $q$ | $z$ |
| 1 | $qR1$ | $zS1$ |
| $\Delta$ | $zS1$ | $zS\Delta$ |

Составляется функциональная схема в виде таблицы таким образом, что знаки, обозначающие колонки и строки, определяют входные параметры ЛУ, а в ячейке таблицы на их пересечении стоит выходная команда. В частности, если головка машины обозревает секцию ленты со знаком 1 и машина находится в рабочем состоянии ($q$), то результатом ее работе на данном такте должно стать повторение 1 в данной секции и переход на одну секцию вправо $R$ (при этом, как указывалось, лента сдвигается влево) — эта команда записывается как $qR1$. Если же в обозреваемой секции $\Delta$, а состояние ЛУ $q$, то $\Delta$ будет заменен 1, сдвига ленты не будет и машина остановится — $zS1$. При комбинации на входе $\Delta z$, как и $1z$, машина находится в нерабочем состоянии — не происходит ни изменения конфигурации, ни движения — по этой причине такие комбинации в функциональных схемах в дальнейшем отображаться не будут.

Пусть начальной является конфигурация $1q1111^*$. Тогда работа машины в соответствии с описанной логической функции будет происходить следующим образом:

**Такт 1.** Обозревается 1, в ЛУ состояние $q$. Выходная команда $qR1$, что эквивалентно перемещению головки по отношению ленты на 1 шаг вправо. Следовательно, образуется промежуточная конфигурация $11q111$.

**Такт 2.** Аналогичным образом осуществляется переход к конфигурации $111q11$.

**Такт 3.** Переход к конфигурации $1111q1$.

**Такт 4.** Переход к конфигурации $11111q\Delta$ (здесь для лучшего понимания правый $\Delta$ указан в явном виде).

**Такт 5.** Обозревается $\Delta$, в ЛУ состояние $q$. Выходная команда $zS1$ — вместо $\Delta$ в ячейку записывается 1, сдвига нет, работа прекращается. Конечная конфигурация $111111z$.

Задача решена.

**Пример 9.9.**

Имеется запись многоразрядного целого числа $n$ в десятичной системе счисления; построить машину Тьюринга, которая обеспечивала бы вычисление значение $n + 1$, если обозревается последняя цифра числа.

---

* Как указывалось выше, незначащие пустые секции слева и справа от заполненных в конфигурацию не включаются; поскольку в данной задаче несколько заполненных секций следуют подряд, только они и указываются в конфигурации.

Используем внешний алфавит $A = \{0, 1, \ldots, 9, \Delta\}$, в котором символ $\Delta$ соответствует пустому знаку. Внутренний алфавит, как и в предыдущей задаче, образуется двумя состояниями — рабочим ($q$) и остановкой ($z$) ($Q = \{q, z\}$). Исходное число $n$, а также результат — $n+1$ — записываются в десятичной системе, причем цифры размещаются по одной в соседних ячейках без пропусков. Функциональную схему представляется таблицей (для удобства строка будут соответствовать состоянию $q$, а столбцы — знакам внешнего алфавита):

| $a$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | $\Delta$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $q$ | $zS1$ | $zS2$ | $zS3$ | $zS4$ | $zS5$ | $zS6$ | $zS7$ | $zS8$ | $zS9$ | $qL0$ | $zS1$ |

Пусть начальной конфигурацией будет $21q9$.

**Такт 1.** $q9 \to qL0$, т. е. 9 будет заменена на 0 и головка сдвинется на разряд десятков — промежуточная конфигурация $2q10$.

**Такт 2.** $q1 \to zS2$, т. е. 1 будет заменена на 2 и произойдет остановка с конечной конфигурацией $2z20$, т. е. получен результат сложения $219 + 1$.

Пусть начальной будет конфигурация $99q9$.

**Такт 1.** $q9 \to qL0$, т. е. сформируется промежуточная конфигурация $9q90$.

**Такт 2.** $q9 \to qL0$ — возникнет конфигурация $q900$.

**Такт 3.** $q9 \to qL0$ — возникнет $q\Delta 000$.

**Такт 4.** $q\Delta \to zS1$ — возникнет $z1000$ и работа прекращается.

Таким образом, описанный алгоритм действительно обеспечивает суммирование любого целого десятичного числа и единицы. Ясно также, что при необходимости произвести сложение не с единицей, а с каким-то целым $m$, то данный алгоритм необходимо повторить $m$ раз. Умножение целых чисел также может быть сведено к сложению числа с самим собой. Следовательно, машины Тьюринга обладают важным свойством — возможностью построения новой машины путем объединения уже имеющихся — такая операция называется *композицией*.

По своему устройству машина Тьюринга крайне примитивна. Она намного проще самых первых компьютеров. Примитивизм состоит в том, что у нее предельно прост набор элементарных операций, производимых головкой, — считывание и запись, а также в том, что доступ к ячейкам памяти (секциям ленты) в ней происходит не по адресу, как в компьютерах, а последовательным перемещением вдоль ленты. По этой причине даже такие простые действия, как сложение или сравнение двух символов, машина Тьюринга производит за несколько шагов, а обычные операции

сложения и умножения требуют весьма большого числа элементарных действий. Однако машина Тьюринга была придумана не как модель (прототип) реальных вычислительных машин, а для того, чтобы показать принципиальную (теоретическую) возможность построения сколь угодно сложного алгоритма из предельно простых операций, причем сами операции и переход от одной из них к последующей машина выполняет автоматически.

Машина Тьюринга дает один из путей уточнения понятия алгоритма. В связи с этим возникают вопросы:

- насколько общим является понятие машины Тьюринга?
- можно ли считать, что способ задания алгоритмов с помощью машины Тьюринга является универсальным?
- может ли всякий алгоритм задаваться таким образом?

На эти вопросы современная теория алгоритмов предлагает ответ в виде следующей гипотезы:

**Всякий алгоритм может быть задан посредством тьюринговой функциональной схемы и реализован в соответствующей машине Тьюринга.**

Эта гипотеза получила название называется *тезиса Тьюринга*. Как и тезис Черча, ее нельзя доказать, так как она связывает нестрогое определение понятия алгоритма со строгим определением машины Тьюринга.

В принципе, эта гипотеза может быть опровергнута, если удастся привести пример алгоритма, который не может быть реализован с помощью тьюринговой функциональной схемы. Однако все известные до сих пор алгоритмы могут быть заданы посредством тьюринговых функциональных схем.

## 9.4. Нормальные алгоритмы Маркова

Кратко обсудим третий подход к уточнению (конкретизации) понятия алгоритма. По смыслу оно близко к идеям Тьюринга, однако в нем не используются представления о каких-либо машинах. Алгоритм задается системой подстановок, которые указывают, какие символы необходимо заменять и в каком порядке эти подстановки должны следовать. Такой подход был предложен А.А. Марковым. В начале 50-х годов было введено понятие *нормальноˊго алгоритма* (сам Марков называл их *алгорифмами*).

Вновь рассмотрим некоторый алфавит $A$, содержащий конечное число знаков (букв). Введем ряд определений:

***Слово*** *— это любая конечная последовательность знаков алфавита.*

*Число символов в слове называется его* ***длиной****.*

*Слово, длина которого равна 0, называется* ***пустым****.*

*Слово $s$ называется* ***подсловом*** *слова $q$, если $q$ можно представить в виде $q = rst$, где $r$ и $t$ — любые слова в том же алфавите (в том числе и пустые).*

Теперь можно определить понятие алгоритма (не являющееся строгим):

***Алгоритмом*** *в алфавите $A$ называется эффективно вычислимая функция, областью определения которой служит какое-либо подмножество множества всех слов в алфавите $A$ и значениями которой также являются слова в алфавите $A$.*

В алгоритмах Маркова в качестве элементарного шага алгоритма принимается подстановка одного слова вместо другого. Пусть в алфавите $A$ построено исходное слово $P$, которое содержит подслово $P_r$ (в общем случае таких подслов в исходном слове может быть несколько), а также имеется некоторое слово $P_k$ в том же алфавите.

***Подстановкой*** *называется замена первого по порядку подслова $P_r$ исходного слова $P$ на слово $P_k$. Обозначается подстановка $P_r \to P_k$.*

Алгоритм в данной форме представления задается *системой подстановок*, которая представляет собой последовательность (список) подстановок. Если в этом списке имеются подстановки с левыми частями, которые входят в $P$, то *первая из них* применяется к $P$, в результате чего оно переходит в другое слово $P_1$. К нему вновь применяется схема подстановок и т. д. Процесс прекращается в двух случаях: либо в списке не нашлось подстановки с левой частью, входящей в $P_n$, либо при получении $P_n$ была применена последняя подстановка.

**Пример 9.10.**

Пусть задан алфавит $A = \{*, 1\}$ и единственная подстановка: $*1 \to 1$; Найти результат обработки, если исходным является слово $P = 11 * 111 * 1$.

Применение нормального алгоритма с указанной подстановкой к данному слову дает последовательность (подчеркиванием выделяется преобразуемая комбинация):

$$11\underline{*1}11 * 1 \to 11111\underline{*1} \to 111111,$$

т. е. алгоритм находит количество единиц в исходном слове (суммирует числа в унарной системе счисления).

**Пример 9.11.**

Алфавит содержит символы русского языка: $A = \{\text{а}, \text{б}, \ldots, \text{я}\}$. Найти систему подстановок, обеспечивающих преобразования: *путь* $\to$ *мутъ*, *поло* $\to$ *мала*. Найти результат применения такого алгоритма к исходным словам *папа*, *пузо*.

Система подстановок достаточно очевидна: п $\to$ м, о $\to$ а.

Применение алгоритма:

$$\underline{п}апа \to ма\underline{п}а \to мама\underline{п}узо \to музо\underline{} \to муза$$

**Пример 9.12.**

Составить нормальный алгоритм, обеспечивающий выполнение операции сложения в троичной системе счисления.

Алфавит будет содержать символы: $A = \{0, 1, 2, +\}$; система подстановок: $0 + 1 \to 1$, $1 + 1 \to 2$, $2 + 1 \to +10$, $+1 \to 1$. Применим алгоритм для различных исходных слов:

$$11\underline{2 + 1} \to 1\underline{1 + 10} \to 120$$

$$2\underline{2 + 1} \to \underline{2 + 10} \to \underline{+}100 \to 100$$

Различные нормальные алгоритмы отличаются друг от друга алфавитами и системами допустимых подстановок. Нормальный алгоритм Маркова можно рассматривать как стандартную форму для задания любого алгоритма. Данная форма представления алгоритма важна не только с точки зрения проведения исследований в теории алгоритмов, но она послужила основой специализированного языка символьных преобразований в системах искусственного интеллекта.

## 9.5. Сопоставление алгоритмических моделей

Вернемся к формулировке проблемы, решение которой мы обсуждали. Некоторые теоретические проблемы (например, проблема алгоритмической разрешимости) и потребности практики (например, необходимость формулировки принципов работы устройств, производящих автоматизированную обработку информации) потребовали построения строгого определения алгоритма. Различные варианты решения проблемы привели к построению так называемых *абстрактных алгоритмических систем* (их называют также *алгоритмическими моделями*). Мы рассмотрели лишь некоторые из них; их полный перечень может быть

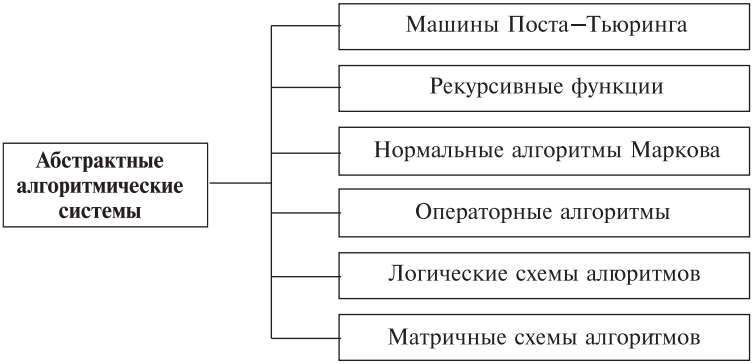



**Рис. 9.4.** Класс абстрактных алгоритмических систем (моделей)

проиллюстрирован схемой на рис. 9.4. Попробуем выяснить, какова взаимосвязь отдельных моделей.

Все алгоритмические задачи принято делить на два больших класса: *первый* — это задачи, связанные с вычислением значения функции; *второй* — задачи, связанные с распознаванием принадлежности объекта заданному множеству (что равносильно получению ответа на вопрос: обладает ли объект заданным свойством?). В первом случае алгоритм $Q$ начинает работать с исходными данными — словом $P$, составленным на основе алфавита $A$, и за конечное число шагов (преобразований) он должен остановиться и выдать результат $P_k = f_Q(P)$. Результат зависит (является функцией) от исходного слова, а также последовательности обработки, т. е. самого алгоритма. При этом *вычисление* понимается в широком смысле — как алфавитное преобразование.

В задачах, отнесенных ко второму классу, в результате выполнения алгоритма получается ответ на вопрос: «*Истинно ли высказывание, что* $x \in M$»? или, что то же самое, проверяется истинность предиката $x \in M$ и выдается один из двух возможных результатов: ИСТИНА или ЛОЖЬ. Этот класс можно считать разновидностью первого, поскольку предикат — это функция, принимающая два значения в зависимости от условия. Тем не менее, разделение этих классов задач полезно, так как приводит к двум важным понятиям теории алгоритмов — *вычислимая* функция и *разрешимое множество*.

> *Функция называется* **вычислимой**, *если имеется алгоритм, вычисляющий ее значение.*

> *Множество называется* ***разрешимым****, если имеется алгоритм, который для любого объекта позволяет определить, принадлежит он данному множеству или нет.*

Как уже говорилось ранее, данные определения не являются формально строгими, т. е. не позволяют предсказать, окажется ли некоторая функция вычислимой или нет (или, что тоже самое: как по характеру функции определить, можно ли построить алгоритм для ее вычисления?). По этой причине весьма важным для построения теории алгоритмов был тезис Черча, который утверждал, что всякая частично рекурсивная функция является вычислимой. Другими словами, если функцию удалось построить с помощью суперпозиции, рекурсии и минимизации из простейших арифметических действий, то существует алгоритм, ее вычисляющий. Далее последовал тезис Тьюринга, утверждавший, что для всякой вычислимой функции можно построить машину Тьюринга, которая ее вычисляет. Можно доказать, что алгоритмы Поста также сводятся к алгоритмам, реализуемых с помощью частично рекурсивных функций. Справедливо и обратное утверждение. В частности задачи, решенные для машины Поста в п. 9.3.2, являются примером реализации одной из простейших рекурсивных функций — функции непосредственного следования. Позднее была доказана теорема о сводимости одной алгоритмической модели к другой, следствием которой явились утверждения типа: «*любую рекурсивную функцию можно вычислить с помощью соответствующей машины Тьюринга*» или «*для любой задачи, решаемой с помощью машины Тьюринга, существует решающий ее нормальный алгоритм Маркова*». Таким образом, все модели оказываются *эквивалентными*, в чем виден глубокий смысл, ибо результат обработки информации, безусловно, определяется характером функции (алгоритмом) и входными данными, но не зависит от вида алгоритмической модели.

## 9.6. Проблема алгоритмической разрешимости

Всякому алгоритму соответствует задача, для решения которой он был построен. Обратное утверждение в общем случае является неверным по двум причинам: во-первых, одна и та же задача может решаться различными алгоритмами; во-вторых, можно предположить (пока), что имеются задачи, для которых алгоритм вообще не может быть построен.

Как уже отмечалось, термин «*алгоритм*» появился в математике достаточно давно и использовался долго — под ним понималась процедура, позволявшая путем выполнения последовательности определенных элементарных шагов получать однозначный результат, не зависящий от того, кто эти шаги выполнял. Таким образом, само проведенное решение служило доказательством существования алгоритма. Однако был известен целый ряд математических задач, разрешить которые в общем виде не удавалось. Примерами могут послужить три древние геометрические задачи: о трисекции угла, о квадратуре круга и об удвоении куба — ни одна из них не имеет общего способа решения с помощью циркуля и линейки без делений.

Интерес математиков к задачам подобного рода привел к постановке вопроса: *можно ли, не решая задачи, доказать, что она алгоритмически неразрешима, т. е. что нельзя построить алгоритм, который обеспечил бы ее общее решение*? Ответ на это вопрос важен, в том числе, и с практической точки зрения, например бессмысленно пытаться решать задачу на компьютере и разрабатывать для нее программу, если доказано, что она алгоритмически неразрешима. Именно для ответа на данный вопрос и понадобилось сначала дать строгое определение алгоритма, без чего обсуждение его существования просто не имело смысла. Построение такого определения, как мы знаем, привело к появлению формальных алгоритмических систем, что дало возможность *математического доказательства* неразрешимости ряда проблем. Оно сводится к доказательству невозможности построения рекурсивной функции, решающей задачу, либо (что эквивалентно) к невозможности построения машины Тьюринга для нее, либо несостоятельности любой (какой-либо) другой модели из представленных на рис. 9.3. То есть *задача считается алгоритмически неразрешимой, если не существует машины Тьюринга (или рекурсивной функции, или нормального алгоритма Маркова), которая ее решает.*

Первые доказательства алгоритмической неразрешимости касались некоторых вопросов логики и самой теории алгоритмов. Оказалось, например, что неразрешима задача установления истинности произвольной формулы исчисления предикатов (т. е. исчисление предикатов неразрешимо) — эта теорема была доказана в 1936 г. Черчем. В 1946–47 гг. А.А. Марков и Э. Пост независимо друг от друга доказали отсутствие алгоритма для распознавания эквивалентности слов в любом ассоциативном исчислении.

В теории алгоритмов к алгоритмически неразрешимой относится «*проблема остановки*»: можно ли по описанию алгоритма $Q$ и входным данным $x$ установить, завершится ли выполнение алгоритма результативной остановкой? Эта проблема имеет прозрачную программистскую интерпретацию. Часто ошибки разработки программы приводят к *зацикливанию* — ситуации, когда цикл не завершается и не происходит завершения работы программы и остановки. Неразрешимость проблемы остановки означает, что нельзя создать общий (т. е. пригодный для любой программы) алгоритм отладки программ. Неразрешимой оказывается и проблема распознавания эквивалентности алгоритмов: нельзя построить алгоритм, который для любых двух алгоритмов (программ) выяснял бы, всегда ли они приводят к одному и тому же результату или нет.

Важность доказательства алгоритмической неразрешимости в том, что если такое доказательство получено, оно имеет смысл закона-запрета, позволяющего не тратить усилия на поиск решения, подобно тому, как законы сохранения в физике делают бессмысленными попытки построения вечного двигателя. Вместе с этим необходимо сознавать, что алгоритмическая неразрешимость какой-либо задачи в общей постановке не исключает возможности того, что разрешимы какие-то ее частные случаи. Справедливо и обратное утверждение: решение частного случая задачи еще не дает повода считать возможным ее решения в самом общем случае, т. е. не свидетельствует о ее общей алгоритмической разрешимости. Роль абстрактных алгоритмических систем в том, что именно они позволяют оценить возможность нахождения полного (общего) решения некоторого класса задач. Для специалиста в области информатики важно сознавать, что наличие алгоритмически неразрешимых проблем приводит к тому, что оказывается невозможным построить универсальный алгоритм, пригодный для решения любой задачи (и, следовательно, бессмысленно тратить силы на его создание). К подобным проблемам приводят и попытки алгоритмизировать сложную интеллектуальную деятельность человека, например обучение других людей, сочинение стихов и пр.

## 9.7. Сложность алгоритма

Обсудим еще одну характеристику алгоритма — его сложность. Развитие и совершенствование компьютерной техники по-

влекло за собой создание разнообразных алгоритмов, обеспечивающих решение многочисленных прикладных задач, причем даже для однотипных задач создавалось (и создается) много алгоритмов (программ). Подобные алгоритмы ранее были названы эквивалентными. Однако для практики недостаточно знать, что решение некоторой задачи на компьютере в принципе возможно (т. е. проблема алгоритмически разрешима). Исполнение любого алгоритма требует определенного объема памяти компьютера для размещения данных и программы, а также времени центрального процессора по обработке этих данных — эти ресурсы ограничены и, следовательно, правомочен вопрос об эффективности их использования. Таким образом, в самом широком смысле понятие эффективности связано со всеми вычислительными ресурсами, необходимыми для работы алгоритма. Однако обычно под «*самым эффективным*» понимается алгоритм, обеспечивающий наиболее быстрое получение результата, поскольку в практических ситуациях именно ограничения по времени часто являются доминирующим фактором, определяющим пригодность того или иного алгоритма. По этой причине далее будет обсуждаться именно временна́я сложность алгоритмов.

Время работы алгоритма удобно выражать в виде функции от одной переменной, характеризующей «*размер*» конкретной задачи, т. е. объем входных данных, необходимых для ее решения. Такой подход удобен, поскольку сравнительная сложность задач может оцениваться через ее размер. Поскольку описание задачи, предназначенной для решения посредством вычислительного устройства, можно рассматривать в виде слова конечной длины, представленной символами конечного алфавита, в качестве формальной характеристики размера задачи можно принять длину входного слова. Например, если стоит задача определения максимального числа в некоторой последовательности из $n$ элементов, то и размер задачи будет $n$, поскольку любой вариант входной последовательности можно задать словом из $n$ символов.

> ***Временна́я сложность алгоритма*** *— это функция, которая каждой входной длине слова n ставит в соответствие* ***максимальное*** *(для всех конкретных однотипных задач длиной n) время, затрачиваемое алгоритмом на ее решение.*

При этом, безусловно, предполагается, что во всех задачах используется одинаковая схема кодирования входных слов.

Таблица 9.2

| Функция временно́й сложности | Размер задачи $n$ | | | | | |
|---|---|---|---|---|---|---|
| | 10 | 20 | 30 | 40 | 50 | 60 |
| $n$ | 0,00001 с | 0,00002 с | 0,00003 с | 0,00004 с | 0,00005 с | 0,00006 с |
| $n^2$ | 0,0001 с | 0,0004 с | 0,0009 с | 0,0009 с | 0,0025 с | 0,0036 с |
| $n^3$ | 0,001 с | 0,008 с | 0,027 с | 0,064 с | 0,125 с | 0,216 с |
| $n^5$ | 0,1 с | 3,2 с | 24,3 с | 1,7 мин | 5,2 мин | 13,0 мин |
| $2^n$ | 0,001 с | 1,0 с | 17,9 мин | 12,7 дней | 35,7 лет | 366 веков |
| $3^n$ | 0,059 с | 58 мин | 6,5 лет | 3855 веков | $2 \cdot 10^{10}$ лет | $1{,}3 \cdot 10^{15}$ лет |

Различные алгоритмы имеют различную временную сложность и выяснение того, какие из них окажутся *достаточно эффективны*, а какие нет, определяется многими факторами. Однако теоретики, занимающиеся разработкой и анализом алгоритмов, для сравнения эффективности алгоритмов предложили простой подход, позволяющий прояснить ситуацию. Речь идет о различии между *полиномиальными* и *экспоненциальными* алгоритмами.

> ***Полиномиальным** называется алгоритм, временна́я сложность которого выражается некоторой полиномиальной функцией размера задачи n.*

Алгоритмы, временна́я сложность которых не поддается подобной оценке, называются *экспоненциальными*.

Различие между указанными двумя типами алгоритмов становятся особенно заметными при решении задач большого размера. Для сопоставления в табл. 9.2 приведены данные о времени решения задач различной сложности (данные взяты из книги М. Гэри, Д. Джонсона [17, с. 20] и соответствуют состоянию развития вычислительной техники приблизительно тридцатилетней давности — это сказывается на абсолютных значениях времени обработки, но относительные показатели при этом не изменятся).

Из приведенных данных видно, что, во-первых, время обработки экспоненциальных алгоритмов при одинаковых размерах задач (превышающих 20) намного выше, чем у полиномиальных; во-вторых, скорость нарастания времени обработки с увеличением размера задачи у экспоненциальных алгоритмов значительно выше, чем у полиномиальных.

Различие между обоими типами алгоритмов проявляются еще более убедительно, если проанализировать влияние увеличения быстродействия компьютера на время исполнения алгорит-

Таблица 9.3

| Функция временно́й сложности | Быстродействие компьютера (усл. ед.) | | |
|---|---|---|---|
| | 1 | 100 | 1000 |
| n | $N_1$ | $100N_1$ | $1000N_1$ |
| $n^2$ | $N_2$ | $10N_2$ | $31{,}6N_2$ |
| $n^3$ | $N_3$ | $4{,}64N_3$ | $10N_3$ |
| $n^5$ | $N_4$ | $2{,}5N_4$ | $3{,}98N_4$ |
| $2^n$ | $N_5$ | $N_5 + 6{,}64$ | $N_5 + 9{,}97$ |
| $3^n$ | $N_6$ | $N_6 + 4{,}19$ | $N_6 + 6{,}29$ |

ма. В табл. 9.3 показано, насколько возрастают размеры наибольшей задачи, решаемой за единицу машинного времени, если быстродействие компьютера вырастет в 100 и 1000 раз.

Из табл. 9.3 видно, что, например, для экспоненциального алгоритма с функцией сложности $f(n) = 2^n$ рост скорости вычислений в 1000 раз приводит лишь к тому, что размер наибольшей задачи возрастает всего на 10 единиц, в то время как для функции $f(n) = n^5$ она возрастает почти в 4 раза.

Приведенные примеры призваны показать, что подобно тому, как существуют алгоритмически неразрешимые задачи, существуют и задачи объективно сложные, т. е. такие, трудоемкость которых невозможно уменьшить совершенствованием компьютера. Задача считается *труднорешаемой*, если для нее не удается построить полиномиального алгоритма. Это утверждение не является категорическим, поскольку известны задачи, в которых достаточно эффективно работают и экспоненциальные алгоритмы. Примером может служить симплекс-метод, который используется при решении задач линейного программирования, имея функцию сложности $f(n) = 2^n$. Однако подобных примеров не очень много, и общей следует признать ситуацию, что эффективно исполняемыми можно считать полиномиальные алгоритмы с функциями сложности $n, n^2$ или $n^3$. Например, при решении задачи поиска нужного данного из $n$ имеющихся в худшем варианте сложность равна $n$; если же оценить среднюю трудоемкость (продолжительность поиска), то она составит $(n + 1)/2$ — в обоих случаях функция сложности оказывается линейной $n$. Задача ранжирования, т. е. расстановки в заданном порядке $n$ однотипных данных приводит к полиному 2-й степени; сложность задачи вычисления определителя системы $n$ линейных уравнений с $n$ неизвестными характеризуется полиномом 3-й степени. Повышение

быстродействия элементов компьютера уменьшает время исполнения алгоритма, но не уменьшает степень полинома сложности. Следовательно, решению практической задачи на компьютере должна предшествовать оценка ее сложности и доказательство того, что задача решаема за приемлемое время.

Сложность алгоритма является ключевым фактором при разработке криптографических систем (см. гл. 7).

## Контрольные вопросы и задания к гл. 9

1. С чем связана необходимость точного определения понятия «алгоритм»?

2. Почему приведенное в п. 9.1 определение алгоритма названо «нестрогим»?

3. Можно ли считать алгоритмом: (a) правила правописания; (b) законы физики; (c) математические формулы; (d) статьи уголовного кодекса. Ответы обоснуйте.

4. На какие свойства алгоритма окажет влияние выбор того или иного исполнителя для решения одной и той же задачи?

5. Можно ли считать исполнителем алгоритма: (a) человека, ведущего запись текста под диктовку; (b) компьютер; (c) компьютерную программу; (d) дрессированное животное. Ответы обоснуйте.

6. Доказать, что примитивно-рекурсивными являются функции: (a) $xy$; (b) $x^y$; (c) $n!$.

7. Каким образом связаны свойства алгоритма и особенности устройства алгоритмической машины?

8. Какие действия алгоритмической машины следует считать *элементарными*?

9. Решите следующие задачи, используя алгоритмическую машину Поста; во всех задачах в исходном состоянии обозревается крайняя левая ячейка:

a) на ленте находятся два числа $N$ и $Q$, разделенные одной пустой ячейкой. Напишите программу нахождения суммы $N + Q$.

b) решите предыдущую задачу при условии, что исходные числа разделены произвольным числом пустых ячеек.

c) на ленте находятся два числа $N$ и $Q$ $(N > Q)$, разделенные одной пустой ячейкой. Напишите программу нахождения разности $N - - - Q$.

d) на ленте $N$ меток. Построить такое же количество меток справа от имеющихся через одну пустую.

e) на ленте находятся два числа $N$ и $Q$, разделенные одной пустой ячейкой. Напишите программу нахождения произведения $NQ$.

10. На каком-либо языке программирования высокого уровня разработайте программу эмуляции работы машины Поста.

11. Решите следующие задачи, используя алгоритмическую машину Тьюринга; во всех задачах в исходном состоянии обозревается крайняя левая ячейка:

a) сложение двух чисел в унарной системе счисления (например, 1111+11);

b) Дано слово из знаков $a$ и $b$ произвольной длины (например, *abbbab*), причем, заранее не известно, какой знак первый ($a$ или $b$). Необходимо первый знак переместить в конец слова.

c) Добавление 1 к числу в произвольной заданной системе счисления.

d) Перевод целого числа из одной системы счисления в другую.

12. На каком-либо языке программирования высокого уровня разработайте программу эмуляции работы машины Тьюринга.

13. Найти значение функции $S^2(S^1, S^1)$ (т. е. результат подстановки функции непосредственного следования самой в себя).

14. Нормальный алгоритм имеет алфавит $A = \{a, b, c\}$ и систему подстановок: $ac \to aa$, $aab \to bc$, $bc \to cab$. Найти результат применения алгоритма к исходным словам: (1) *cbcbba*; (2) *abccba*; (3) *accca*.

15. На каком-либо языке программирования высокого уровня разработайте программу, обеспечивающую задание и выполнение нормальных алгоритмов Маркова.

16. Разработайте нормальные алгоритмы, обеспечивающие:

a) выполнение операции вычитания единицы из числа в троичной системе счисления;

b) выполнение операции добавления единицы к числу в двоичной системе счисления;

c) инверсию числа в двоичном алфавите;

d) преобразование *док*$\to$ *тоска*. Применить его к словам *ток*, *дот*.

# 10 Формализация представления алгоритмов

Абстрактные алгоритмические модели, как уже неоднократно отмечалось, используются лишь при построении теории и доказательстве общих свойств алгоритмов. Для практических целей такое представление алгоритмов чаще всего неудобно, поскольку, во-первых, не всегда практически реализуема форма представления (например, машина Тьюринга или Поста); во-вторых, элементарные шаги, выделяемые в моделях, оказываются слишком «мелкими» для современных технических устройств, которые выступают в качестве исполнителей алгоритмов. В связи с этим встает вопрос о том, как может быть описан алгоритм, предназначенный для решения практической задачи с помощью реального технического устройства? Обсуждению вариантов ответа на него и посвящена данная глава.

## 10.1. Формальные языки

### 10.1.1. Формальная грамматика

Алгоритм был ранее определен как алфавитный оператор с конечной системой правил преобразования. Для записи входных, промежуточных и выходных слов используется некоторый алфавит. Каким-то образом должны быть описаны и правила преобразования. Очевидно, для этого требуется некоторый *язык*. Пригоден ли для описания алгоритма обычный разговорный язык?

Любой естественный язык возникал как средство общения людей. Именно по этой причине ему присущи такие особенности, как:

- изменчивость, которая состоит в непостоянстве словарного состава языка;
- неоднозначность трактовки фраз различными людьми;
- избыточность, о чем шла речь в разделе «Кодирование информации».

Перечисленные особенности не позволяют применить естественный язык для записи алгоритма, поскольку одним из свойств алгоритма является его детерминированность, т. е. однозначность выполнения шагов любым исполнителем. Наиболее простой путь преодоления нежелательных свойств естественных языков — построение языков искусственных со строгим синтаксисом и полной смысловой определенностью — такие языки получили название *формальных*.

В любом языке — естественном или искусственном — можно выделить две составляющие: *синтаксис* и *семантику*. *Синтаксис* (грамматика языка) — это совокупность правил, согласно которым строятся допустимые в данном языке конструкции. *Семантика* — смысловая сторона языка — она соотносит единицы и конструкции языка с некоторым внешним миром, для описания которого язык используется.

Для описания формального языка необходим другой язык, с помощью которого будут создаваться языковые конструкции. Описываемый формальный язык называется *языком-объектом*, а язык, средствами которого выполняется описание, — *метаязыком*. Метаязык должен обеспечивать как описание структурных единиц языка и правил объединения их в допустимые предложения, так и содержательную (смысловую) сторону языковых конструкций.

Любая грамматика начинается с указания алфавита, т. е. набора символов, посредством которого строятся конструкции языка.

Синтаксис формального языка задается некоторой системой правил (*порождающей системой*), которая из небольшого набора исходных конструкций порождает все допустимые их комбинации, т. е. язык образуется как множество разрешенных правилами сочетаний исходных конструкций. Кроме этого, синтаксис содержит формулировку условия, которое выполняется для законченных конструкций языка и не выполняется в противном случае.

Помимо синтаксиса устанавливается система правил, позволяющих конструкциям языка придать смысл — эти правила образуют *семантику языка*.

> ***Формальная грамматика*** *— система правил, описывающая множество допустимых конечных последовательностей символов формального алфавита.*

*Конечные цепочки символов, построенных в соответствии с формальной грамматикой, называются* ***предложениями формального языка,*** *а само множество цепочек —* ***языком,*** *описываемым данной грамматикой.*

Набор синтаксических правил формального языка аналогичен системе подстановок, используемых в нормальных алгоритмах Маркова. Вывод в данной порождающей грамматике есть последовательность цепочек, в которой любая, начиная со второй, получается из предыдущей применением какого-либо правила вывода.

Формальная грамматика задается упорядоченной четверкой $\{T, N, S, P\}$, где $T$ и $N$ — непересекающиеся конечные множества, образующие алфавит или словарь порождаемого формального языка; $T$ называется множеством (словарем) *терминальных символов*; $N$ — множеством (словарем) *нетерминальных* (вспомогательных) *символов*; $S$ — начальный (выделенный) вспомогательный символ из множества $N$; $P$ — набор правил вывода конструкций языка (подстановок) из выделенного вспомогательного символа, имеющие вид $g \to h$, где $g$ и $h$ — цепочки, состоящие как из терминальных, так и нетерминальных символов.

Подстановки работают следующим образом: если в преобразуемой цепочке есть слово $g$, то оно заменяется словом $h$. Единственное ограничение на вид подстановок состоит в том, что в слово $g$ не может состоять только из терминальных символов. Это означает, что получение на некотором шаге цепочки, состоящей *только из терминальных символов*, свидетельствует о прекращении процесса порождения — эта цепочка является правильной, завершенной конструкцией порождаемого языка. Подстановки $P$ могут применяться к трансформируемой цепочке в произвольном порядке.

**Пример 10.1.**

Пусть формальная грамматика задается следующим образом: $T = \{a, b\}$ (т. е. множество терминальных символов — алфавит языка — состоит из двух символов — $a$ и $b$); $N = \{S\}$, т. е. множество нетерминальных символов состоит из единственного символа $S$ — он, естественно, оказывается выделенным; система подстановок пусть имеет следующий вид: $S \to aSa$, $S \to bSb$, $S \to a$, $S \to b$.

Описанная грамматика порождает язык, состоящий из всех «*слов-перевертышей*» в алфавите $\{a, b\}$, имеющих нечетную длину, т. е. слов, которые слева направо читаются также, как справа налево, например, *aba*, *abababa*, *bbbbb*, *baaaaaab* и т. д. Легко видеть, что применение первых двух

правил (в любом числе и любой последовательности) порождает цепочки (слова) типа $\alpha S\alpha^{-1}$, где $\alpha^{-1}$ означает слово $\alpha$, записанное справа налево; применение третьего и четвертого правил завершает процесс порождения слова и формируют слова типа $\alpha a\alpha^{-1}$ или $\alpha b\alpha^{-1}$.

**Пример 10.2.**

Рассмотрим формальную грамматику, порождающую фрагмент естественного языка. Пусть $T = \{\text{а}, \text{б}, \ldots, \text{я}, \text{А}, \text{Б}, \ldots, \text{Я}\}$ — множество терминальных символов — букв русского алфавита. Нетерминальный алфавит строится из символов $N = \{Q, R, S\}$, где $Q = \{q_1, \ldots, q_n\}$ — множество имен людей в русском алфавите, $R = \{r_1, \ldots, r_m\}$ — множество глаголов, стоящих в третьем лице единственного числа настоящего времени; $r_i$ и $q_j$ записываются с помощью терминальных символов. Пусть система подстановок имеет вид:

$$S \to QR$$
$$Q \to q_1; Q \to q_2; \ldots; Q \to q_m;$$
$$R \to r_1; R \to r_2; \ldots; R \to r_m.$$

Очевидно, эта грамматика порождает язык, состоящий из фраз типа: «*такой-то делает то-то*», например, «*Маша читает*», «*Вася спит*» и т. п. Работает грамматика следующим образом: на первом шаге определяется тип фразы; второй шаг порождает конкретное имя, а третий шаг — конкретное действие (глагол).

Из данного примера виден содержательный смысл нетерминальных символов — они могут обозначать различные классы конкретных слов, в частности традиционные грамматические классы — части речи, члены предложения и пр.

Хотя мы подошли к рассмотрению формальных грамматик в связи с необходимостью построения строго (однозначно понимаемого) описания алгоритма, на самом деле области их применения в информатике гораздо обширнее. На основе формальных грамматик создаются языки программирования и трансляторы к ним. При решении задач искусственного интеллекта они используются в системах машинного перевода, а также для генерации синтаксически правильных предложений в ответах экспертных систем на запросы пользователей. Формальные грамматики могут быть применяться в учебных и иных программах (например, Microsoft Word), где требуется проверка правильности вводимого текста и поиск в нем ошибок.

### 10.1.2. Способы описания формальных языков

Как уже было сказано, для описания языка-объекта должен применяться метаязык. Но метаязык также должен обладать некоторыми свойствами формального языка, чтобы однозначно оп-

ределять конструкции языка-объекта. Следовательно, метаязык должен быть сначала описан сам, для чего также нужен язык, — естественно, может сложиться впечатление, что такой процесс никогда не закончится. Однако доказано, что *для описания любого метаязыка можно использовать язык естественный.*

Таким образом, для построения формального языка необходимо средствами естественного языка описать метаязык, а затем посредством метаязыка описать формальный язык (рис. 10.1).

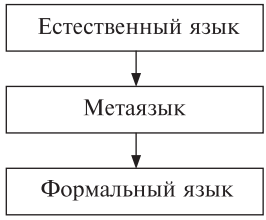



**Рис. 10.1.** Построение формального языка

Последовательность построения формального языка следующая: сначала определяется алфавит языка (в широком смысле *как полная упорядоченная совокупность знаков, с помощью которой записываются любые конструкции языка*). Далее из знаков принятого алфавита строятся стандартные комбинации — их называют *служебными словами.* Служебные слова и алфавит образуют множество терминальных символов языка. Для описания нетерминальных символов на начальных этапах построения языка используется метаязык, основой которого служит язык естественный. Однако в ходе описания допустимых конструкций формального языка все нетерминальные символы должны быть выражены (определены) через символы терминальные. Другими словами, нетерминальные символы выступают в качестве «строительных лесов», которые убираются после завершения построения всех возможных конструкций формального языка.

Рассмотрим два варианта описания метаязыков.

Один из широко распространенных метаязыков известен как *нотации Бекуса–Наура.* Для формирования предложений в форме Бекуса–Наура используются универсальные *метасимволы*: $\{<, >, ::=, |\}$. Первые два метасимвола называют «*угловыми скобками*» — они служат для обрамления нетерминального символа. Символ «::=» читается «*по определению есть*»; символ «|» — «*или*». В предложениях, записанных в форме Бекуса–Наура, нетерминальный символ, стоящий в угловых скобках, играет роль определяемой конструкции языка-объекта. В формулах Бекуса–Наура могут использоваться терминальные символы из алфавита языка-объекта, отличные от универсальных метасимволов.

Описание формального языка строится из последовательности формул, каждая из которых в левой части содержит один метасимвол, обозначающий некоторую конструкцию языка-объекта. Правая часть такой формулы содержит либо перечисление метасимволов и терминальных символов языка-объекта (никаких разделителей при этом не ставится), либо совокупности перечислений, разделенных символом «|». Правая и левая части объединяются в единую формулу знаком «::=».

Язык-объект можно считать полностью определенным в форме Бекуса–Наура, если любой нетерминальный символ можно представить последовательностью терминальных символов.

В качестве примера можно рассмотреть определение нетерминального символа «*идентификатор*», который используется во многих языках программирования. На естественном языке определение звучит следующим образом: «*Идентификатор — это любая последовательность букв и цифр, начинающаяся с буквы*». В форме Бекуса–Наура оно будет выглядеть следующим образом: сначала определяются нетерминальные символы «буква» и «цифра» (сами символы, как видно, записаны посредством естественного языка):

$$< \text{буква} >::= a|b|c|d|e\ldots; \quad < \text{цифра} >::= 0|1|2|3|\ldots|9$$

Теперь, считая эти два символа определенными (они выражены через терминальные), можно строить определение нетерминального символа «идентификатор»:

$$< \text{идентификатор} >::=< \text{буква} > | < \text{идентификатор} > < \text{буква} > | < \text{идентификатор} >< \text{цифра} > |$$

Элементарным оказывается идентификатор из одной буквы. Далее, подставляя вместо нетерминальных символов их возможные терминальные значения, можно получить в правой части последовательность только терминальных символов — на этом построение конкретного идентификатора будет завершено. Приведенная же форма выражает *обобщенное правило* построения любого идентификатора. По этой причине в определении присутствует рекурсивность — новый идентификатор может быть построен из имеющегося добавлением к нему справа букв или цифр.

Достоинство нотаций Бекуса–Наура в том, что они представляется в буквенном виде; неудобны нотации однообразностью способов построения предложений языка-объекта — запись оказывается громоздкой и плохо воспринимаемой.

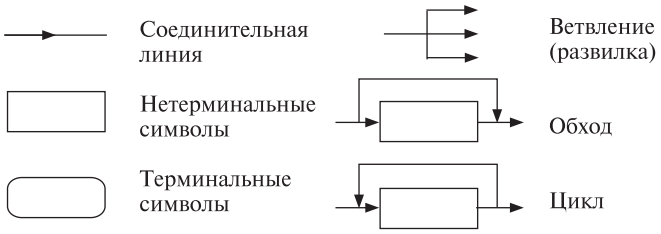



**Рис. 10.2.** Элементы синтаксических диаграмм

Гораздо более наглядной следует считать другой способ описания формального языка, который был предложен Никласом Виртом — создателем языка программирования PASCAL — и который получил название «*синтаксические диаграммы*». Синтаксическая диаграмма — это схема (графическое представление) описания какого-либо нетерминального символа языка-объекта. Схема всегда имеет один вход и один выход. Элементами схемы могут служить терминальные символы языка-объекта, заключенные в окружность (или овал) или нетерминальные символы (понятия) языка-объекта, заключенные в прямоугольник. Элементы соединяются между собой направленными линиями, указывающие порядок следования объектов в определяемом нетерминальном символе. Приняты обозначения, указанные на рис. 10.1.

Структура синтаксических диаграмм идентична структурам языков программирования, что позволило широко использовать диаграммы для написания трансляторов различных языков. Первым языком, описанным с помощью синтаксических диаграмм, был язык PASCAL.

Чтение диаграммы выполняется в направлении стрелок; в точке ветвления может выбираться любой маршрут. В качестве метаязыка может использоваться естественный русский язык; языки программирования строятся на англоязычной основе. Терминальные символы переписываются в конструкции формального языка дословно. Нетерминальные символы могут выражаться через терминальные или другие нетерминальные — в этом случае для них строятся уточняющие диаграммы; в конечном счете, все нетерминальные символы должны быть выражены через терминальные. При использовании синтаксических диаграмм принимается условие, что среди терминальных символов языка-объекта не должно быть одинаковых, а также ни один из терминальных символов не может служить началом другого. При нарушении данного условия возможно неоднозначное чтение ди-

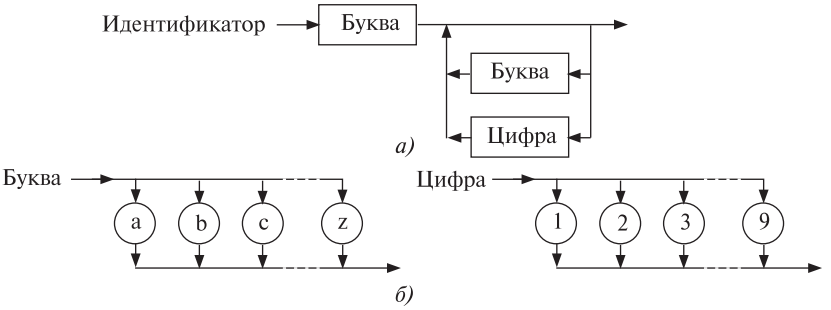



**Рис. 10.3.** Диаграммы понятия «идентификатор»

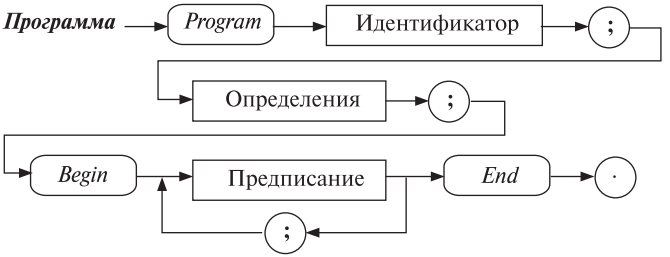



**Рис. 10.4.** Диаграмма программы на языке PASCAL

аграммы и, как следствие, построение или распознавание неверной конструкции языка.

Рассмотрим ряд примеров построения синтаксических диаграмм, первым из которых будет определение понятия «*идентификатор*» для сопоставления с приведенной выше нотацией Бекуса–Наура (рис. 10.3,*а*). При этом необходимы уточняющие диаграммы (рис. 10.3,*б*). Примеры построения англоязычных идентификаторов в соответствии с этой диаграммой: q, a123, identificator, e2e4.

Диаграмма, задающая общий вид программы на языке PASCAL, приведена на рис. 10.4. Терминальными здесь являются символы «*Program*», «*Begin*», «*End*», «;» и «.». Нетерминальный символ «Идентификатор» уже определен выше. Значит, для дальнейшей конкретизации и построения конечных грамматических конструкций требуется определить нетерминальные символы «Определения» и «Предписание».


**Рис. 10.5.** Виды предписаний

Далее определяются каждый из видов предписаний (рис. 10.5). В качестве приме-

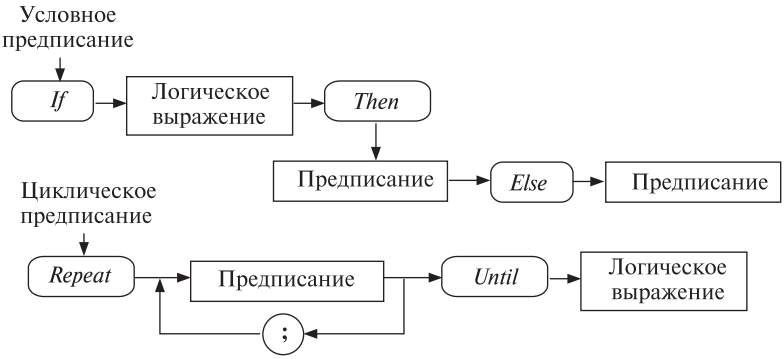



**Рис. 10.6.** Диаграммы условного и циклического предписаний

ров предписаний рассмотрим по одному виду предписаний «Условное» и «Циклическое», используя в качестве терминальных символов служебные слова языка PASCAL (рис. 10.6).

В соответствии с этими и подобными диаграммами строятся допустимые синтаксические конструкции языка.

Итак, нотации Бекуса–Наура и синтаксические диаграммы — это два альтернативных способа описания конструкций метаязыка, с помощью которого строится формальный язык. После того как построена формальная грамматика и ею порожден язык, он может быть использован для решения прикладных задач — коммуникации, хранения и обработки информации. Последний класс задач приводит к необходимости формулировки с помощью языков последовательностей обработки информации, т. е. алгоритмов, и их представлению в форме, доступной для понимания и исполнения лицом или техническим устройством, которые обработку производят.

## 10.2. Способы представления алгоритмов

Предыдущие рассуждения позволяют заключить, что свойство определенности алгоритма с неизбежности требует использования для его описания формальных языков. Однако степень формализации, т. е. то, насколько строгим должен быть синтаксис языка, а также возможные способы представления алгоритмов определяются тем, кто (или что) предполагается в качестве его исполнителя. В информатике сложились вполне определенные традиции в представлении алгоритмов, рассчитанных на различных исполнителей. Если алгоритм предназначен для исполнителя-человека, то запись его может быть не полностью

формализована; существенными в представлении оказываются *понятность* и *наглядность* — по этим причинам для записи алгоритма может быть использован естественный язык или язык графический. В представлении алгоритмов, предназначенных для исполнения техническими устройствами, важнейшими качествами оказываются *однозначность понимания и исполнения*, а также *ограниченность допустимых грамматических конструкций*, что требует использования строго формализованных языков. Помимо этого, особенности устройств ввода и возможности интерпретации входной информации техническим устройством допускают запись алгоритма только в словесной форме.

Далее более подробно рассматриваются различные формы представления алгоритмов. Однако сначала уточним понятие «*исполнитель алгоритма*», введенное в п. 9.1.

### 10.2.1. Исполнитель алгоритма

При построении алгоритмической теории понятие исполнителя алгоритма в явном виде не вводится. Механизм исполнения предлагается лишь в моделях Тьюринга и Поста, поскольку с ним связана суть модели. В остальных моделях обсуждаются лишь элементарные шаги алгоритма и правила построения сложных действий из элементарных. За рамками обсуждения остается вопрос: кем (или чем) данный алгоритм будет выполняться. Точнее, принимается по умолчанию, что для выполнения алгоритма в той или иной модели необходимо уметь понимать принятую форму записи и осуществлять необходимые действия.

Введем понятие *формального исполнителя*:

> ***Формальный исполнитель*** *— субъект или устройство, способные воспринимать и анализировать указания алгоритма, изменять в соответствии с ним свое состояние, а также обладающие механизмом исполнения, способным выполнять пошаговую обработку информации.*

Исполнитель алгоритма считается заданным, если для него установлены:

- система команд (элементарных действий алгоритма, которые способен выполнить исполнитель);
- формы представления входной и выходной информации;
- система допустимых внутренних состояний;
- язык представления алгоритма.

Таким образом, в решении задач практики первичными оказываются не особенности алгоритма, а возможности исполнителя. В частности, элементарность шагов определяется не тем, какая модель использована для представления алгоритма, а системой команд конкретного исполнителя. Форма представления исходных (входных) данных для любого алгоритма также должны быть ориентированы на конкретного исполнителя. Наконец, никакая логическая структура алгоритма не должна переводить исполнителя в запрещенное состояние (т. е. выводить за рамки допустимых состояний).

Помимо непосредственного выполнения действий конкретный исполнитель осуществляет и контроль правильности разработки алгоритма. Причинами невыполнения алгоритма при некотором наборе исходных данных (т. е. не достигается результативного окончания его работы) могут быть:

- ошибки синтаксиса, т. е. нарушение формальных правил записи алгоритма;
- выход начальных данных за пределы допустимого множества;
- несоответствие алгоритма возможностям исполнителя.

Если в роли исполнителя выступает компьютер, а алгоритм представляется в виде программы, синтаксический контроль осуществляется на этапе ее компиляции, т. е. до того, как начнется исполнение программы. В случае, когда ошибки имеют смысловой (семантический) характер, для их локализации и исправления прибегают к *тестированию* программы. Тестирование состоит в проверке работоспособности алгоритма (программы) при таких значениях исходных данных, которые охватили бы все возможные пути обработки информации. На практике, однако, осуществить такую проверку для сложных алгоритмов весьма затруднительно — слишком велико оказывается число возможных вариантов. Обычно делается попытка обработки предельных (больших и малых) входных значений, обработки недопустимых значений (их ввод не должен приводить к нерезультативной остановке исполнителя; точнее результатом должно быть сообщение исполнителя о невозможности выполнения действий или просто отсутствие действия). Поскольку перебрать все сочетания входных данных чаще всего невозможно, следует сознавать, что тестирование может обнаружить ошибку, но не доказывает их полное отсутствие.

Из приведенных рассуждений может сложиться впечатление, что на практике всегда формулировка и способ представления алгоритма оказываются зависящими от возможностей исполнителя и, следовательно, должны быть ориентированы на учет этих возможностей. Это справедливо, если мы рассматриваем какой-то конкретный алгоритм. Однако если имеется некоторое множество алгоритмов, то их единообразное представление становится обязательным только в том случае, когда исполнитель единственный — именно такой была ситуация на начальных этапах развития вычислительной техники — решение любой задачи требовало представления алгоритма в виде программы на входном языке компьютера. Ситуация изменялась по мере развития техники и специализированного программного обеспечения; в настоящее время можно считать, что компьютер через свое программное обеспечение предоставляет пользователю множество исполнителей, из которых следует выбрать *оптимальный*, т. е. наиболее соответствующий задаче и алгоритму.

### 10.2.2. Строчная словесная запись алгоритма

В соответствии с рассмотренными выше способами описания формальных языков в представлении алгоритмов можно выделить две основные формы: символьную (словесную) и графическую.

Строчная запись, как ясно из названия, представляет собой последовательность строк, каждая из которых содержит описание одного или нескольких элементарных действий. Эти строки могут иметь метки в виде букв или порядковых числовых номеров. Логика алгоритма, т. е. порядок выполнения действий, задается либо в явном виде путем указания метки последующей строки, либо в неявном — по умолчанию управление передается строке, следующей за выполненной. «Элементарность» действия определяется возможностями исполнителя.

Данный способ представления алгоритма следует считать *основным*, поскольку последовательностью строк может быть записана алгоритмическая нотация для любого исполнителя — как человека, так и технического устройства. Для человека строчная алгоритмическая запись может быть максимально приближена к естественному языку. Для технического устройства запись выполняется на специализированном формализованном языке.

Недостатком строчной формы представления алгоритма является неудобство целостного восприятия его логической структуры. Указание об изменении естественной последовательности

действий в строчной записи выглядит в виде ссылки на строку с меткой. В случае достаточно сложного алгоритма такие ссылки затрудняют уяснение логики и последовательности действий, что необходимо при проверке и отладке алгоритма.

Рассмотрим несколько примеров строчной записи алгоритмов для различных исполнителей.

***Пошагово-словесная*** форма *представляет собой пронумерованную последовательность строк, каждая из которых содержит описания конкретных действий на естественном языке.* Данная форма применяется в том случае, если исполнителем является человек. Обычно в качестве примеров алгоритмов, представленных в такой форме, приводят кулинарные рецепты, инструкции по использованию каких-либо устройств, указания по посадке деревьев и т. п. Однако примеры эти нельзя признать корректными, поскольку в них идет речь не об обработке информации, а об управлении действиями, направленными на получение некоторого материального результата. Примерами данной формы представления могут служить алгоритмы математических вычислений над конечными числами. Рассмотрим хорошо известный со школы алгоритм Евклида нахождения наибольшего общего делителя двух натуральных чисел ($a$ и $b$); его пошагово-словесное описание выглядит следующим образом:

- Если $a = b$, результатом считать $a$; закончить вычисления.
- Если $a > b$, найти разность $a - b$; новым значением $a$ считать значение разности; перейти к п. 1;
- Если $b > a$, найти разность $b - a$; новым значением $b$ считать значение разности; перейти к п. 1.

Удобство пошагово-словесной формы — в ее универсальности (по отношению к классам описываемых алгоритмов), использовании естественного национального языка для записи конструкций, отсутствии строгой формализации. Форма широко используется для представления различных учебных алгоритмов.

***Формула*** — *строчная запись действий, обеспечивающих обработку числовых, символьных или логических данных.* Формулы, предназначенные для исполнителя «*человек*», не обязательно могут быть строчными — это приводит к некоторой неоднозначности порядка действий, не сказывающейся, однако, на результат вычислений благодаря дистрибутивному и сочетательному законам. Пример:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}.$$

Если же формула записывается для вычисления компьютером, она представляется строго в строчной форме, а для однозначности выполнения действий устанавливаются *приоритеты операций*. Например, в языке PASCAL для математических и логических операций приняты следующие приоритеты:

- операции в скобках;
- возведение в степень, вычисление значения стандартных функций и процедур-функций;
- логическое отрицание NOT;
- умножение, деление, целочисленное деление (div), остаток от целочисленного деления (mod), логическое И (AND);
- сложение, вычитание, логическое ИЛИ (OR);
- операции отношения ($>$, $<$, $=$, $>=$, $<=$, $<>$).

Помимо указанных приоритетов принимается дополнительное правило:

- при наличии операций равного приоритета они выполняются в порядке слева направо.

Приведенная выше формула, записанная в соответствии с приоритетами и правилом, будет выглядеть следующим образом:

$$\mathrm{x1} := (-\mathrm{b} + \mathrm{sqrt}(\mathrm{b} * \mathrm{b} - 4 * \mathrm{a} * \mathrm{c}))/(2 * \mathrm{a}).$$

Обработка символьной информации выполняется предназначенными для этого функциями и процедурами.

**Псевдокод** — *ориентированный на исполнителя «человек» частично формализованный язык, позволяющий записывать алгоритмы в форме, весьма близкой к алголоподобным языкам программирования*. Термин «*частично формализованный*» в данном случае означает, что в псевдокоде строго определены только правила записи управляющих структур, а описание самих действий остается естественным. Псевдокод имеет русскоязычную основу и используется, в основном, при обучении азам программирования.

Алгоритм, представленный с помощью автокода, представляет собой последовательность строк, в каждой из которых содержится описание действий либо по обработке данных, либо по управлению процессом обработки. Для записи управляющих структур приняты следующие обозначения:

- внешнее оформление: АЛГ — начало алгоритма, ПРОЦ — начало процедуры, КНЦ; — конец процедуры, КНЦ. — конец алгоритма;

- ветвление: ЕСЛИ... ТО... ИНАЧЕ... ВСЕ; после ЕСЛИ ставится описание логического условия, по которому происходит ветвление, после ТО — описание действий (их может быть несколько), которые исполняются при значении условия TRUE, если ветвление полное — после ИНАЧЕ описываются альтернативные действия, в любом случае в конце ставится слово ВСЕ, которое служит признаком окончания данной конструкции;
- цикл: ПОКА... ПОВТОРЯТЬ... КЦ; после ПОКА ставится описание логического условия выполнения команд цикла, после ПОВТОРЯТЬ — описание действий (тела цикла), КЦ — признак конца циклической конструкции.

Часто при записи алгоритма отдельные действия заканчиваются разделителем (например, «;») — это позволяет избежать ошибок в случае, если описание действия занимает не одну строку. Помимо этого, для удобочитаемости и наглядности применяется так называемая «*структурная запись*», при которой отдельные элементы структур записываются не с начала строки, а с отступом, показывающим вложенность и подчиненность этих элементов.

В качестве примера алгоритма, записанного с помощью псевдокода, приведем рассмотренный выше алгоритм Евклида нахождения НОД двух целых чисел ($a$ и $b$).

```
АЛГ Евклид;
  ПОКА a ≠ b
    ПОВТОРЯТЬ
      ЕСЛИ a > b ТО a := a − b
      ИНАЧЕ b := b − a
    ВСЕ
  КЦ;
  ПЕЧАТЬ a
КНЦ.
```

Удобство использования псевдокода — в сочетании относительной строгости синтаксических конструкций и их русскоязычной основы. Близость конструкций псевдокода языкам программирования позволяет легко перейти от одного к другому. Однако отсутствие формальных правил записи действий не позволяет использовать псевдокод для составления алгоритмов, исполнителями которых являются технические устройства.

> ***Язык программирования*** *— искусственный формализованный язык, предназначенный для записи алгоритма для исполнителя «компьютер», метаязыком которого является естественный язык.*

Язык программирования строго фиксирует (т. е. определяет) и изображение управляющих структур, и описание допустимых действий, и синтаксические правила построения сложных структур.

Различают языки *низкого уровня* (машинно-ориентированные) и *высокого уровня* (машинно-независимые). К языкам первого типа относятся:

- *машинный язык* (язык машинных кодов) — совокупность команд, интерпретируемых и исполняемых компьютером; каждый оператор программы на этом языке является машинной командой, а все данные отыскиваются по абсолютным значениям адресов, по которым они располагаются в ОЗУ;
- *ассемблер* (*макроассемблер*) — язык символического кодирования — операторами языка являются машинные команды, которым приписываются мнемонические обозначения, а в качестве операндов используются не конкретные адреса в ОЗУ, а их символические имена.

Пример команд ассемблера:

*CLA* — очистить один из регистров сумматора (аккумулятор);

*ADD* — сложение содержимого ячейки, номер которой написан после команды, с содержимым аккумулятора; результат остается в аккумуляторе;

*MOV* — содержимое аккумулятора пересылается в ячейку с номером, записанным вслед за командой;

*HLT* — стоп.

Безусловно, ассемблеры содержат и другие команды.

Рассмотрим простой пример сложения чисел $a, b$ и $c$. Результат должен присваиваться переменной $d$.

| Код | Комментарий |
|---|---|
| CLA | Очистка аккумулятора |
| ADD a | В аккумуляторе $a$ |
| ADD b | В аккумуляторе $a + b$ |
| ADD c | В аккумуляторе $a + b + c$ |
| MOV 200 | Перемещение результата в ячейку номер 200 |
| HLT | Прекращение действий |

Алгоритм записывается в текстовом редакторе с ASC-кодировкой. Ясно, что для преобразования текста в последовательность машинных команд необходима еще одна промежуточная программа — она называется *компилятор*. На этапе компиляции выполняется также распределение данных в ОЗУ; при этом вместо имен переменных подставляются относительные адреса ячеек, в которых данные располагаются. Абсолютные адреса данным присваивает операционная система при размещении программы в ОЗУ компьютера перед ее использованием.

Ассемблер является машинно-зависимым языком, т. е. записанная на нем программа может исполняться лишь на той технике, точнее, тем типом процессора, ассемблер которого был использован.

Для всех языков *высокого уровня* общим является то, что ориентированы они не на систему команд той или иной машины, а на систему операторов, характерных для записи определенного класса алгоритмов. Примерами таких операторов, содержащихся во всех языках программирования, являются операторы присваивания, условные операторы и операторы циклов — причина этого будет обсуждаться ниже; помимо этого обязательно имеются операторы ввода-вывода для организации взаимодействия программы с пользователем.

По функциональному назначению языки программирования высокого уровня различают на *проблемно-ориентированные* и *универсальные*. Из названий классов ясно, что первые предназначены для решения каких-то специфических задач из некоторой отрасли знаний. Примерами являются язык FORTRAN (FORmula TRANslator) — язык решения сложных научных и инженерных задач (кстати, это был первый язык программирования высокого уровня); COBOL (Common Business Oriented Language) — язык для решения экономических и коммерческих задач; LISP (List Processing Language) — язык, используемый в решении задач искусственного интеллекта. К универсальным языкам относятся PASCAL (Philips Automatic Sequence CALculator), BASIC (Beginner ALL-purpose Symbolic Instruction Code), C (C++), Jawa, а также современные среды визуального программирования DELPHI, VISUAL BASIC и др. Эти языки позволяют решить, вообще говоря, любую задачу, хотя трудоемкость решения конкретной задачи в разных языках будет различаться.

Другой признак, в соответствии с которым возможна классификация языков программирования, является *парадигма (кон-*

Таблица 10.1

| Парадигма программирования | Представление программ и данных | Исполнение программы | Связь частей программы между собой |
|---|---|---|---|
| Процедурное | Программа и данные представляют собой отдельные, не связанные друг с другом элементы | Последовательное выполнение операторов | Возможна только через совместно обрабатываемые данные |
| Объектно-ориентированное | Данные и методы их обработки инкаспулированы в рамках единого объекта | Последовательность событий и реакций объектов на эти события | Отдельные части программы могут наследовать методы и элементы данных друг у друга |
| Логическое | Данные и правила их обработки объединены в рамках единого логического структурного образования | Преобразование логического образования в соответствии с логическими правилами | Разбиение программы на отдельные независимые части затруднительно |

*цепция) программирования*, т. е. *совокупность основополагающих идей и подходов, определяющих модель представления данных и их обработки, а также методологии программирования*. Основные различия в парадигмах представлены в табл. 10.1.

В настоящее время в распоряжении программиста имеется весьма обширный спектр языков-инструментов, из которых для любой конкретной задачи можно выбрать тот, что позволит решить ее оптимальным путем. Мы снова приходим к высказанной ранее мысли: компьютер по отношению к пользователю оказывается не единственным исполнителем, а предоставляется множеством исполнителей, из которых следует выбрать наиболее подходящий для задачи.

Следует выделить особый класс языков программирования — языки, встроенные в прикладные программы, например язык VBA в MS Office или внутренний язык пакета MathCAD. С одной стороны, их следует отнести к языкам высокого уровня, поскольку они имеют собственные формальные грамматики и интерпретаторы, которые встроены в соответствующие пакеты. С другой стороны, программы, написанные на них, вообще говоря, не могут исполняться без «программы-хозяина» и, в отличие от «настоящих» языков высокого уровня, по их текстам нельзя получить

исполняемые модули на другом типе компьютера или в другой операционной системе.

### 10.2.3. Графическая форма представления алгоритма

Другое распространенное название данной формы — *блок-схема*. В данной форме для представления отдельных блоков алгоритма используются обусловленный набор геометрических фигур. Принятые обозначения показаны на рис. 10.7.

Графическая форма предназначена, безусловно, только для исполнителя «*человек*» — в этом ее основной недостаток. Главное достоинство такой формы представления — наглядность; блок-схема позволяет охватить весь алгоритм сразу, отследить различные варианты его выполнения. На стадии разработки в блоках можно делать записи как на естественном, так и на формальном языке. Именно по этой причине блок-схема считается весьма полезной формой при обучении алгоритмизации, а также при разработке сложных алгоритмов. Однако в блок-схеме, как правило, отсутствует подробное описание конкретных действий — их существование лишь обозначено.

По блок-схеме гораздо проще осуществляется запись алгоритма на каком-либо формальном языке. Правда, следует заметить, что синтаксическое богатство языков программирования выше языка блок-схем — по этой причине не все языковые конструкции имеют простое графическое представление — примером может служить конструкция цикла с параметром FOR ... DO (или FOR ... NEXT), не имеющая собственного представления в языке блок-схем.

В качестве примера приведем блок-схему обсуждавшегося выше алгоритма Евклида (рис. 10.8).

В заключении хотелось бы подчеркнуть различие между блок-схемами и синтаксическими диаграммами. *Синтаксиче-*

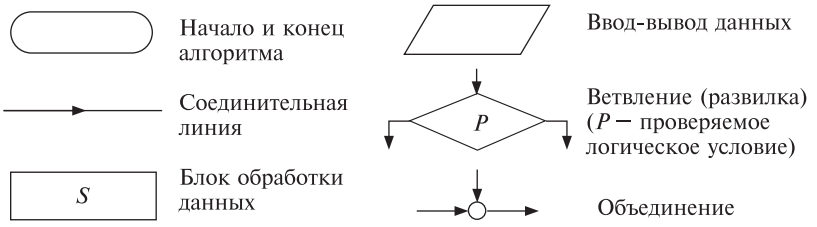



**Рис. 10.7.** Элементы блок-схем

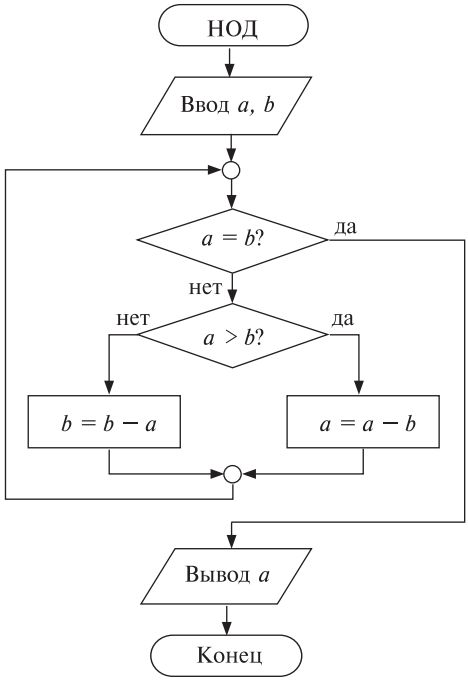



**Рис. 10.8.** Блок-схема алгоритма Евклида

*ские диаграммы являются средством описания и средством генерации конструкций формального языка.* При этом каждая диаграмма позволяет генерировать множество однотипных конструкций. Другими словами, синтаксическая диаграмма — это правило порождения некоторой конструкции языка. Язык программирования, построенный с помощью синтаксических диаграмм, является средством словесного описания алгоритма. Блок-схема же является *графической формой представления конкретного алгоритма*, в который отдельные конструкции входят в качестве составных элементов.

### 10.2.4. Классификация способов представления алгоритмов

Как следует из проведенного выше рассмотрения, по уровню формализации представление алгоритмов можно разделить на две группы: естественное и формальное. В группу естественного представления входят некоторые виды строчной записи и графическая форма. Группа формального представления включает алгоритмические модели и формальные языковые кон-


**Рис. 10.9.** Классификация способов представления алгоритмов

струкции. Все варианты представления алгоритмов могут быть объединены в единую классификационную схему, изображенную на рис. 10.9.

Как мы увидим ниже, на основании приведенной классификации строится общий порядок алгоритмического решения задачи. В частности, будет показано, что разработка компьютерной программы для решения некоторой прикладной задачи требует умения пользоваться как формальными, так и естественными способами представления алгоритмов.

## 10.3. Структурная теорема

После того как были рассмотрены возможные способы записи алгоритмов, вполне закономерным представляется вопрос о технологии их разработки. До середины 60-х годов теории разработки алгоритмов не существовало — процесс разработки целиком определялся опытом и искусством программиста. Однако по мере роста сложности программ возникла необходимость создания методологии разработки программ, и она появилась в виде *структурного программирования*. Идеи структурного программирования были высказаны в 1965 г. Э. Дейкстрой, но сведены в некую законченную систему правил они не были. В том же году итальянские математики К. Бом и Д. Джакопини сформулировали теорему о структурности. Прежде, чем рассмотреть ее суть, необходимо ввести некоторые понятия.

Поскольку алгоритм определяет порядок обработки информации, он должен содержать, с одной стороны, действия по обработке, а с другой стороны, порядок их следования, называемый *потоком управления*.

Рассмотренные выше элементы блок-схем, связанные с обработкой данных, делятся на *простые* и *условные*. Особенность простого действия в том, что оно имеет один вход и один выход, в отличии от условного, обладающим двумя выходами в зависимости от того, истинным ли окажется условие. Простое действие не означает, что оно единственное, — это может быть некоторая последовательность действий.

> *Часть алгоритма, организованная как простое действие, т. е. имеющая один вход (выполнение начинается всегда с одного и того же действия) и один выход (т. е. после завершения данного блока всегда начинает выполняться одно и то же действие), называется* ***функциональным блоком****.*

Из этого определения, в частности, следует, что каждое простое действие является функциональным блоком, а условное — нет.

Согласно положениям структурного программирования можно выделить *всего три* различных варианта организации потока управления действиями алгоритма. Поток управления может обладать следующими свойствами:

1) каждый блок выполняется не более одного раза;

2) выполняется каждый блок.

Поток управления, в котором выполняются оба эти свойства, называется *линейным* — в нем несколько функциональных блоков выполняются *последовательно*. Линейному потоку на языке блок-схем соответствует структура, показанная на рис. 10.10,*а*. Очевидно, несколько блоков, связанных линейным потоком управления, могут быть объединены в один функциональный блок (рис. 10.10,*б*).

Второй тип потока управления называется *ветвлением* — он организует выполнение одного из двух функциональных блоков в зависимости от значения проверяемого логического условия. Блок-схема структуры показанная на рис. 10.10,*в*. В этом типе выполняется свойство (1), свойство (2) — нет. Если структура содержит два функциональных блока ($S_1$ и $S_2$), ветвление называется *полным*; возможно существование *неполного* ветвления — при этом один из блоков пуст (обычно $S_2$, см. рис. 10.10,*г*).

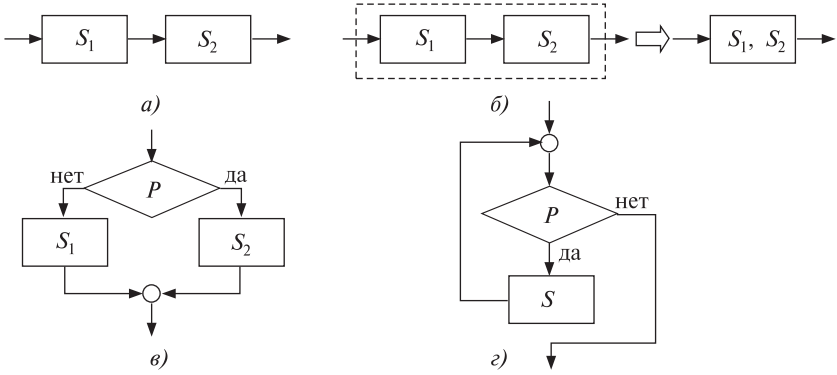



**Рис. 10.10.** Потоки управления

Третий тип потока управления называется *циклическим* — он организует многократное повторение функционального блока, пока логическое условие его выполнение является истинным. Для циклического потока выполняется свойство (2), но не выполняется (1). Его блок-схема показана на рисунке.

Поскольку ветвление и циклический типы управления имеют один вход и один выход, они в целом также подходят под определение функционального блока. Введем рекурсивным образом понятие *стандартного функционального блока*:

- каждое простое действие есть стандартный функциональный блок;
- каждая из описанных трех управляющих структур является стандартным функциональным блоком, если все входящие в них блоки являются стандартными функциональными;
- других стандартных функциональных блоков не существует.

Определим еще одно понятие:

> *Алгоритм называется* ***структурным****, если он может быть представлен стандартным функциональным блоком.*

Другими словами, структурный алгоритм представляет собой комбинацию трех рассмотренных выше структур (иногда они называются *базовыми алгоритмическими структурами*). Безусловно, не все алгоритмы являются структурными. Однако именно структурные алгоритмы обладают рядом замечательных преимуществ по сравнению с неструктурными:

- *понятность* и *простота* восприятия алгоритма (поскольку невелико число исходных структур, которыми он образован);

- *проверяемость* (для проверки любой из основных структур достаточно убедиться в правильности входящих в нее функциональных блоков);
- *модифицируемость* (она состоит в простоте изменения структуры алгоритма, поскольку составляющие блоки относительно независимы).

После введенных определений можно сформулировать структурную теорему Бома–Джакопини:

**Любой алгоритм может быть сведен к структурному.**

Или по-другому:

**Любому неструктурному алгоритму может быть построен эквивалентный ему структурный алгоритм.**

Значение структурной теоремы для практики программирования состоит в том, что на ее основе разработан и широко используется *структурный метод программирования*. Основой метода является использование принципа *модульности* построения сложных программ. При этом каждый программный модуль организуется в виде стандартного функционального блока (т. е. строится только из трех базовых структур) и выполняет лишь одну функцию по обработке данных. В идеале каждый такой модуль должен иметь один вход и один выход. Модули обладают определенной автономностью, что позволяет их отладку (т. е. поиск и устранение ошибок) вести независимо от остальной программы, а также обеспечивает относительно простую модифицируемость как отдельного модуля, так и программы в целом. Эффективность метода структурного программирования особенно заметна при создании сложных программ; модульный принцип позволяет разбить общую задачу на составные и относительно автономные части, каждая из которых может создаваться и отлаживаться независимо (и даже разными разработчиками); безусловно, такое разбиение требует согласования входных и выходных параметров модулей.

## Контрольные вопросы и задания к гл. 10

1. Опишите формальную грамматику, порождающую множество целых двоичных чисел.

2. Измените описание грамматики из примера в п. 10.1.2 таким образом, чтобы она описывала конструкции типа «Имя_1, Имя_2... Имя_N делают_то-то».

3. Что определяет следующая нотация Бекуса–Наура:

<формула> ::= <цифра> | <формула> <знак> <формула>
<знак> ::= + | − | *
<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

4. На каком-либо языке программирования напишите программу, функционирующую в соответствии с нотацией, приведенной в предыдущем задании.

5. С помощью синтаксических диаграмм опишите следующие конструкции языка PASCAL:

a) оператор цикла с предусловием WHILE... DO;

b) составной оператор;

c) оператор цикла с параметром FOR... DO;

d) оператор выбора CASE.

6. Можно ли считать формальным исполнителем алгоритма следующие устройства:

a) кодовый замок;

b) графический редактор;

c) телефон с памятью для записи номеров;

d) принтер?

7. Постройте блок-схемы следующих структурных алгоритмов:

a) вычисление $n!$ (ввод $n$);

b) суммирование цифр целого числа при произвольной его разрядности (ввод — целое число);

c) перевод целого числа в двоичную систему счисления (ввод — целое число);

d) вычисление значения функции $\sin x$ с заданной точностью $e$ суммированием ее разложения в ряд Тейлора (ввод — аргумент $x$, точность вычисления $e$).

8. Запишите с помощью псевдокода алгоритмы, приведенные в задании 7.

9. Напишите программы на каком-либо языке программирования для алгоритмов задания 7.

10. В чем смысл и значение структурной теоремы для практики разработки алгоритмов? Возможно ли существование неструктурных алгоритмов? Если «да» — приведите примеры.

# 11 Представления о конечном автомате

Мы продолжаем обсуждение вопросов, связанных с обработкой (преобразованием) дискретной информации. До этого речь велась об алгоритмах, т. е. последовательности действий по преобразованию информации безотносительно того, кто (или что) эти действия будет выполнять. В данной главе мы выясним, какими качествами должны обладать технические устройства, способные реализовать алгоритмическую обработку дискретной информации. При этом будут рассматриваться не физическая или техническая реализация конкретных устройств, а общие принципы их строения, существенные с точки зрения обработки и хранения информации.

## 11.1. Общие подходы к описанию устройств, предназначенных для автоматической обработки дискретной информации

Начнем с довольно очевидного определения:

> ***Автомат*** *— это устройство, выполняющее без непосредственного участия человека определенную последовательность операций, в результате которой происходит преобразование материальных объектов, энергии или информации.*

Когда говорится «без участия человека», то подразумевается отсутствие явного управления со стороны человека в ходе выполнения операций. Однако на самом деле управление осуществляется, но опосредованно, через программу, составленную предварительно человеком и заложенную в устройство. В дальнейшем мы ограничим себя обсуждением лишь тех устройств, которые предназначены для автоматического преобразования информации, представленной в дискретной форме.

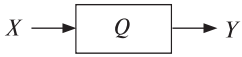



**Рис. 11.1.** Общая схема конечного автомата

Поскольку такое устройство обменивается информацией с внешней средой, очевидно, оно должно обладать *входным каналом*, по которому информация поступает в него, а также *выходным каналом*, через который выдается результат преобразования (рис. 11.1). Помимо этого, устройство может обладать *памятью*, в которой фиксируется его текущее *состояние*; результат обработки в общем случае определяется входными воздействиями и внутренним состоянием устройства. Будем считать, что для представления входной информации используется некоторый конечный (*входной*) алфавит $X = \{x\}$ , а для представления выходной — конечный (*выходной*) алфавит $Y = \{y\}$. По сути $X$ и $Y$ представляют собой множества входных и выходных сигналов устройства. Как отмечалось ранее, требование конечности этих множеств является следствием конечности времени обработки информации. Внутренние состояния устройства также образуют дискретный набор, который можно считать *внутренним алфавитом* — обозначим его $Q = \{q\}$; относительно количества различных внутренних состояний пока никаких предположений не выдвигаем.

Будем полагать также, что поступление входных символов и переключение состояний устройства происходят не непрерывно, а в определенные моменты времени, т. е. *дискретно*. Если последовательность моментов произвольна, то говорят об *асинхронной* организации работы элементов устройства, например набор номера телефона или кода замка.

Однако в сложных устройствах чаще используется *синхронная* организация, при которой моменты поступления и выхода сигналов, а также переключения внутренних состояний следуют друг за другом через один и тот же фиксированный промежуток времени $\Delta t = \text{const}$, называемый *тактом*. Эти моменты задаются с помощью специального устройства — *тактового генератора* (или генератора синхроимпульсов). Число тактовых импульсов за единицу времени называется *тактовой частотой* — она является одним из важнейших факторов, определяющих *быстродействие* устройства. Можно ввести моменты времени $t_0$, $t_1$, $t_2$,..., обозначающие границы тактов ($t_0$ соответствует моменту начала работы). При этом можно считать, что события, относящиеся к такту $i$ — поступление входного символа, изменение внутреннего состояния, формирование и выдача выходного символа — проистекают мгновенно в момент $t_i$.

*Устройства, у которых дискретны множества внутренних состояний, входных и выходных сигналов, а также множество моментов времени, в которые поступают входные сигналы, меняются внутренние состояния и выдаются выходные сигналы, называются* ***дискретными****.*

Примерами дискретных устройств являются наборный диск телефона, кодовый замок, калькулятор, электронные табло и, безусловно, компьютер. По назначению устройства можно разделить на три группы:

- *информационные* — справочные автоматы, светофоры, электронные табло, устройства аварийной сигнализации и т. п.;
- *управляющие* — кодовый замок, устройство управления лифтом, станки с программным управлением, микропроцессоры фотоаппарата, видео, стиральной машины и т. п.;
- *вычислительные* — микрокалькулятор, микропроцессоры компьютеров.

Существуют устройства, осуществляющие деятельность всех трех видов, например, компьютер, автопилот и др.

Для наших рассуждений существенным оказывается то обстоятельство, что сигналы на выходе и внутренние состояния такого устройства оказываются дискретными функциями входных сигналов и номеров тактов работы. Введем понятие *функции переходов* $\Psi$, которая будет связывать внутреннее состояние устройства на последующем такте с состоянием и входным сигналом на текущем такте:

$$q(t_{i+1}) = \Psi(q(t_i), x(t_i)). \tag{11.1}$$

Другими словами, функция переходов показывает, в какое состояние из всех возможных $Q$ перейдет дискретное устройство, если оно находилось в состоянии $q_i$, а на вход поступил сигнал $x_i$.

Подобным же образом введем *функцию выходов* $\Theta$, которая будет связывать внутреннее состояние устройства и входной сигнал на текущем такте с выходным сигналом на этом же такте:

$$y(t_i) = \Theta(q(t_i), x(t_i)). \tag{11.2}$$

Следовательно, функция выходов определяет, какой сигнал образуется на выходе, если на данном такте определен сигнал на входе и состояние устройства.

Говорят, что пятеркой компонентов $\langle X, Y, Q, \Psi, \Theta\rangle$ задается *автомат*, обеспечивающий преобразование по определенным

правилам последовательностей сигналов входного алфавита в выходную последовательность. Действительно, если принять начальное условие $q_1 = q_0$, то рекуррентные соотношения (11.1) и (11.2) определят порядок преобразования конечной последовательности входных сигналов $x_0, x_1, \ldots, x_n$ в некоторую цепочку выходных сигналов той же длины $y_0, y_1, \ldots, y_n$; в процессе будет возникать определенная последовательность внутренних состояний $q_1, q_2, \ldots$. В этом и заключается функционирование дискретного устройства. Выходной сигнал, вырабатываемый им на некотором такте $i$, зависит не только от входного сигнала, воспринятого на этом же такте, но и от сигналов, поступивших ранее — они фиксируются в дискретном устройстве путем изменения его внутреннего состояния. По этой причине множество внутренних состояний устройства является его *внутренней памятью*. В зависимости от объема этой памяти выделяются следующие типы дискретных устройств:

- без памяти;
- с конечной памятью;
- с бесконечной памятью.

Забегая несколько вперед, скажем, что *дискретное устройство с конечной памятью называется конечным автоматом*. Отметим также, что функции переходов и выходов имеют обобщающее название *автоматные функции*.

Дискретные устройства без памяти автоматами по сути не являются, поскольку при любых комбинациях входных сигналов они вырабатывают выходные всегда по одному и тому же набору правил, содержащихся в функциях выхода; функции переходов, позволяющие менять поведение устройства, отсутствуют. К этому типу дискретных устройств относятся *комбинационные схемы* — они будут рассмотрены ниже.

Что касается дискретных устройств с бесконечной памятью — это чисто модельное теоретическое представление, поскольку никакие реальные устройства бесконечной памятью обладать не могут. Примером автомата с бесконечной памятью может служить машина Тьюринга, в которой, как указывалось в п. 9.3.3, роль памяти выполняет бесконечная (или при необходимости наращиваемая) в обе стороны бумажная лента. Таким образом, можно считать, что *автоматом с бесконечной памятью является алгоритм, представленный в форме машины Тьюринга*. Поскольку вопросы, связанные с функционированием машин Тьюринга, уже рассматривались ранее, в данном разделе они обсуж-

даться не будут. Однако имеет смысл подчеркнуть, что комбинационные схемы и машину Тьюринга можно рассматривать в качестве двух предельных случаев автомата с конечной памятью.

В реальных дискретных устройствах сигналы и их преобразователи могут иметь различную физическую природу (электрическую, механическую, пневматическую и пр.). Однако от этой природы можно отвлечься и изучать только общие законы построения автоматов и правила, определяющие преобразование информации ими.

> *Автомат, рассматриваемый только как преобразователь входных последовательностей сигналов в выходные без конкретизации его устройства, называется* ***абстрактным****.*

По сути абстрактный конечный автомат является формой представления алгоритма с конечной памятью (роль памяти играют внутренние состояния)

В теории абстрактных автоматов решаются несколько групп задач:

*во-первых*, выясняется, какие преобразования возможны в автомате, как их описать, как выполняемые преобразования связаны с числом состояний (т. е. сложностью) автомата, существуют ли неразрешимые автоматом задачи;

*во-вторых*, исследуются эквивалентные преобразования автоматов; задача эквивалентного преобразования состоит в построении автомата эквивалентного данному, но имеющего иное количество внутренних состояний (обычно, меньшее);

*в-третьих*, рассматривается круг вопросов, в которых определяется строение автомата по характеру и соотношению входных и выходных сигналов (автомат — «*черный ящик*») — это важная прикладная задача технической диагностики устройств, без которой невозможна их практическая эксплуатация и ремонт;

*в-четвертых*, выделяются структурные элементы автоматов и определяются правила построения из них сложных устройств (задача *синтеза автоматов*) — с этим связана разработка новых автоматов, например компьютеров.

Важным для практики частным случаем являются автоматы, в которых вся информация представлена с помощью *двоичного кода*, т. е. алфавиты $X$, $Y$ и $Q$ являются двоичными — такие автоматы называют *двоичными* или *логическими*. Последнее название обусловлено тем, что, как показывает теория, при двоичном кодировании любой конечный автомат можно представить

в виде комбинации некоторого числа элементов, реализующих логические операции *И, ИЛИ, НЕ,* а также элементов памяти (задержка и триггер). Объединение элементов называется *логической схемой*. Важными представляются два обстоятельства: во-первых, работу логических схем можно описать законами и правилами математической логики (т. е. результат действия логической схемы сводится к вычислению значения логического выражения); во-вторых, из данного небольшого набора элементов можно построить любой конечный автомат.

## 11.2. Комбинационные схемы

Введенное в предыдущем разделе понятие автомата является достаточно общим. Накладывая ограничения на компоненты $X, Y, Q, \Psi, \Theta$, можно получить частные случаи автоматов. Одним из них являются *автоматы без памяти*, т. е. устройства, в которых не происходит фиксации внутреннего состояния. Очевидно, в этом случае из общего описания должны быть исключены компоненты $Q$ и $\Psi$; автомат без памяти задается тройкой компонентов $\langle X, Y, \Theta\rangle$. Соотношение (11.2) принимает вид

$$y(t_i) = \Theta(x(t_i)),$$

т. е. выходной сигнал на данном такте определяется только входным сигналом и не зависит от ранее поступивших сигналов. Следовательно, каждый автомат без памяти реализует единственный преобразователь (*оператор*), который осуществляет перевод входных последовательностей сигналов в выходные.

Пусть имеется дискретное устройство, имеющее $n$ входов $x_1, \ldots, x_n$ и $m$ выходов $y_1, \ldots, y_m$ (рис. 11.2). Если данное устройство не обладает памятью, то преобразование входных сигналов в выходные описывается системой уравнений:

$$\begin{cases} y_1 = \Theta_1(x_1, \ldots, x_n); \\ \ldots \\ y_m = \Theta_m(x_1, \ldots, x_n). \end{cases}$$

Поскольку входные и выходные сигналы описываются дискретными алфавитами, значения функций в данной системе образуют дискретный ряд.

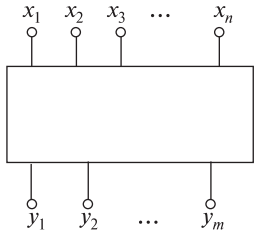



**Рис. 11.2.** Дискретное устройство с несколькими входами-выходами

Если входной и выходной алфавит являются двоичными, то представленная система оказывается системой логических функций, для решения которой можно привлечь аппарат математической логики. Именно такие устройства будем рассматривать в дальнейшем.

Подобные устройства могут быть построены из нескольких элементарных компонентов (*элементов*). Такие элементы образуют конечный набор, называемый *базисом*, а сами элементы — *базисными*. Базис определяет совокупность простейших действий, о которых шла речь в теории алгоритмов, а базисный элемент можно рассматривать в качестве устройства, выполняющего это простейшее действие. Если речь идет о двоичных дискретных устройствах, то базис строятся из элементов, которые реализуют элементарные логические функции. Напомним, что к таким функциям относятся *конъюнкция* (логическое И — $\wedge$), *дизъюнкция* (логическое ИЛИ — $\vee$), *импликация* ($\rightarrow$), *сумма по модулю 2* ($\oplus$), *эквивалентности* ($\sim$) и *отрицания* (логическое НЕ — $\neg$). Однако между элементарными функциями имеются *соотношения эквивалентности* (см. приложение В), позволяющие выразить одни функции через другие. В результате оказывается, что нет необходимости включать в базис все элементы, реализующие простейшие логические функции — достаточно выбрать некоторое минимальное их подмножество. Этому условию удовлетворяет базис, построенных на элементах И, ИЛИ и НЕ — он называется *простейшим*, а входящие в него элементы — *логическими элементами* или (*логическими вентилями*). Схемы элементов приведены на рис. 11.3. Рассмотрим каждый из них по отдельности.

Элемент И ($\wedge$) имеет два входа, на которые *независимо* могут подаваться сигналы $x_1$ и $x_2$; каждый из сигналов является двоичным, т. е. может принимать одно из двух значений — 1 или 0; элемент формирует единственный выходной сигнал $y$, значения которого определяются только входными сигналами. Функция выходов $\mathbf{\Theta}(x_1, x_2)$ согласно табл. В.1 (см. приложение В)

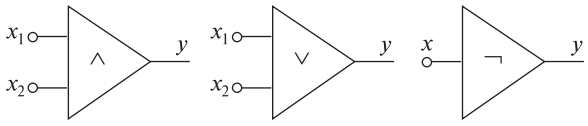



**Рис. 11.3.** Базисные логические элементы

принимает следующие значения:

$$\Theta_\wedge(0,0)=0;\quad \Theta_\wedge(1,0)=0;$$
$$\Theta_\wedge(0,1)=0;\quad \Theta_\wedge(1,1)=1;$$

Функция выходов элемента ИЛИ ($\vee$) принимает следующие значения:

$$\Theta_\vee(0,0)=0;\quad \Theta_\vee(1,0)=1;$$
$$\Theta_\vee(0,1)=1;\quad \Theta_\vee(1,1)=1.$$

Логический элемент НЕ ($\neg$) обеспечивает *одноместное* преобразование (с одним входным сигналом) в соответствии с правилами:

$$\Theta_\neg(0)=1;\quad \Theta_\neg(1)=0.$$

Комбинируя базисные элементы по определенным правилам, можно построить их сложные объединения — *схемы* (над данным базисом), способные совершать преобразования, соответствующие, вообще говоря, любым логическим функциям. Таким образом:

> ***Схема** есть комбинация базисных элементов, в которой выходы одних элементов присоединяются к входам других.*

Если в таких объединениях элементов отсутствуют замкнутые контуры (т. е. нет подачи сигнала с выхода элемента на один из его же входов), то возникает класс схем, которые называются *комбинационными*. Говорят, что базис обладает свойством *полноты*, если схемами над ним может быть реализована любая логическая функция (и, следовательно, любая система логических функций). В частности, таким свойством обладает простейший базис.

Со схемами связано решение двух классов задач — *анализа* и *синтеза*. В задачах анализа выявляется логическая функция, реализуемая заданной схемой. Синтез — это, наоборот, построение схемы по заданной конечной логической функции.

**Пример 11.1.**

Пример задачи анализа: по заданной схеме (рис. 11.4) установить, какую обработку информации она производит.

Решение будем вести последовательно, находя значение функции в точках схемы, отмеченных цифрами. Используем упрощенную запись логических операций и соотношения из приложения В.

1) $x_1 \cdot x_2$, т. е. на выход $b$ поступает результат логического умножения сигналов на входе ($b = x_1x_2$);

2) $x_1 + x_2$;

3) $\overline{(x_1 \cdot x_2)} = \bar{x}_1 + \bar{x}_2$ — закон де Моргана:

$$y = (x_1+x_2)(\bar{x}_1+\bar{x}_2) = x_1\cdot\bar{x}_1+x_2\cdot\bar{x}_1+x_1\cdot\bar{x}_2+x_2\cdot\bar{x}_2 = x_1\cdot\bar{x}_2+x_2\cdot\bar{x}_1 = x_1\oplus x_2.$$

Таким образом, установлены две функции выхода:

$$y = x_1 \oplus x_2 \quad \text{и} \quad b = x_1 \wedge x_2.$$

Результаты обработки различных комбинаций входных сигналов могут быть представлены в виде таблицы:

| $x_1$ | $x_2$ | $y$ | $b$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 |

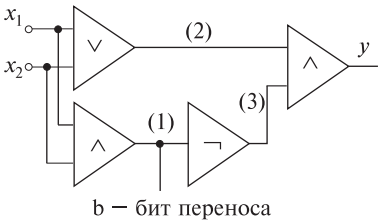



**Рис. 11.4.** Комбинационная схема полусумматора

Поскольку перечисленные преобразования являются правилами двоичного суммирования, рассмотренная комбинация логических элементов получила название *полусумматора*. На ее вход подаются два сигнала: $x_1$ и $x_2$; выходной сигнал $y$ дает результат сложения в том же двоичном разряде, в котором стоят складываемые числа; в случае суммирования 1+1 происходит перенос в старший разряд — для него имеется еще один выход $b$.

В арифметико-логическом устройстве компьютера полусумматор обрабатывает лишь младшие разряды регистров. Для всех остальных разрядов помимо двух складываемых значений необходимо учитывать бит переноса, образовавшийся в результате сложения предыдущих разрядов. Следовательно, комбинационная схема, обеспечивающая выполнение данной операции, должна иметь три входа ($x_1, x_2$ и $b_{i-1}$) и формировать два выходных значения ($y$и$b_i$) — эта схема называется *двоичным сумматором*. Логические функции такой схемы для каждого из разрядов $i$ ($i = 2, \ldots, n$, считая, что младшим является разряд 1):

$$y = x_1 \oplus x_2 \oplus b_{i-1}; \quad b_i = (x_1 \wedge x_2) \vee (x_1 \wedge b_{i-1}) \vee (x_2 \wedge b_{i-1}).$$

Схема, реализующая такие логические функции, называется *последовательным двоичным сумматором* и содержит 15 логических элементов (9 И, 4 ИЛИ и 2 НЕ). Схема двоичного сумматора представлена на рис. 11.5. Такой сумматор обеспечивает выполнение операций в одном из разрядов процессора. Следовательно, 32-разрядный процессор будет содержать 31 схему

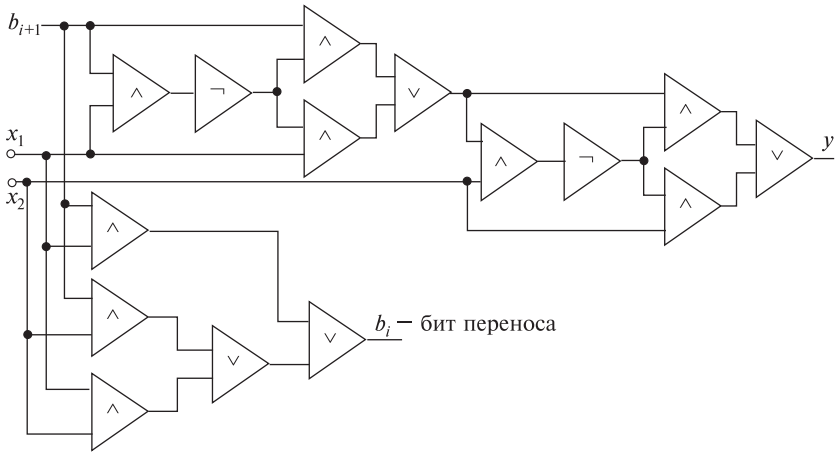



**Рис. 11.5.** Комбинационная схема двоичного сумматора

сумматора и 1 полусумматор (для младшего разряда), которые соединены друг с другом, и вместе образуют *сумматор*.

Подобным образом, вообще говоря, можно построить комбинационную схему для любого конечного множества задач, решение которых (т. е. выходные сигналы) *однозначно* определяются их условием (т. е. входными сигналами). В частности, если ограничиться некоторой фиксированной точностью представления числа, то можно построить комбинационную схему, которая вычисляет значение любой функции $y = f(x_1, \ldots x_n)$ (безусловно, в двоичных кодах), например $\sin x$ и др. Однако на практике получается, что при разрядности 32 и выше даже схема *умножителя*, вычисляющего произведение $x_1 x_2$, становится столь сложной, что оказывается проще реализовать умножение иным путем, который можно назвать *алгоритмическим* и который позволяет представить умножение в виде последовательности сложений и сдвигов, о чем шла речь ранее. Точно также и иные вычисления сводятся к цепочкам элементарных операций: сложение, сдвиг, инверсия и пр.

**Пример 11.2.**

Пример задачи синтеза: построить схему с двумя входами и двумя выходами, которая ни при каких входных комбинациях не давала бы на выходе 11.

Решение начинается с построения таблицы значений:

| $x_1$ | $x_2$ | $y_1$ | $y_2$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

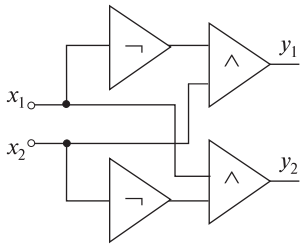



**Рис. 11.6.** Схема к примеру 11.2

То есть при подаче на входы одинаковых сигналов (0,0) или (1,1) на выходе схемы будет (0,0). Наиболее трудным (и даже творческим!) этапом решения является установление явного вида функций выхода по имеющейся таблице значений. В рассматриваемом примере они угадываются (подбираются) достаточно легко:

$$y_1 = x_1 \cdot \bar{x}_2; \quad y_2 = \bar{x}_1 \cdot x_2,$$

Соответствующая схема приведена на рис. 11.6.

Попутно следует заметить, что подобные схемы реально включаются перед элементами памяти — триггерами (см. п. 11.3.2).

## 11.3. Конечные автоматы

### 11.3.1. Способы описания конечного автомата

Комбинационные схемы, хотя и позволяют реализовать любые *фиксированные* зависимости между входными и выходными сигналами, но не могут изменять характера своего поведения (т. е. последовательности обработки данных) — любое такое изменение требует изменения схемы, т. е. по сути переходу к другой схеме. Решить проблему перестройки работы без изменения структуры схемы можно, если ввести в нее *элементы памяти*, которые позволяли бы фиксировать и сохранять промежуточные состояния устройства, — в этом случае выходной сигнал будет зависеть не только от входного сигнала, но и от состояния схемы. Если количество таких элементов конечно, то, как указывалось выше, дискретное устройство будет называться *конечным автоматом*.

> ***Конечным автоматом*** *называется система* $\langle X, Y, Q, \Psi, \Theta \rangle$, *в которой* $X$ *и* $Y$ *являются конечными входным и выходным алфавитами,* $Q$ *— конечным множеством внутренних состояний,* $\Psi(x, q)$ *— функцией переходов и* $\Theta(x, q)$ *— функцией выходов.*

Ранее отмечалось, что $\Psi(x, q)$ задает порядок преобразования входных сигналов и состояния автомата на предыдущем такте в состояние на последующем, а $\Theta(x, q)$ — преобразования входных сигналов и состояния автомата на текущем такте в выходные сигналы. Если $q_0$ — начальное состояние автомата, а $i$ — номер такта, то его работа описывается системой:

$$\begin{cases} q(t_1) = q_0; \\ q(t_{i+1}) = \Psi(q(t_i), x(t_i)); \\ y(t_i) = \Theta(q(t_i), x(t_i)). \end{cases} \tag{11.3}$$

Данные соотношения получили название *системы канонических уравнений* конечного автомата. Пользуясь ими, можно, начиная с $q_0$, последовательно находить все последующие состояния автомата и выходные сигналы.

Выделяются два типа автоматов — *инициальные* и *неинициальные*. В инициальных автоматах начальное состояние фиксировано (т. е. они всегда начинают работать из одного и того же состояния $q_0$). В неинициальных автоматах в качестве начального состояния может быть выбрано любое из множества $Q$; этим выбором определяется дальнейшее поведение автомата.

Представление конкретного конечного автомата фактически сводится к описанию задающих его автоматных функций. Из системы (11.3) следует, что при конечном числе возможных внутренних состояний количество возможных значений автоматных функций также оказывается конечным. Их описание возможно различными способами, наиболее распространенными из которых являются *табличный* и с помощью *диаграмм*.

В **табличном способе** значения автоматных функций задаются двумя таблицами, именуемыми соответственно *матрицей переходов* и *матрицей выходов*. В этих таблицах строки обозначаются знаками входного алфавита, а столбцы — знаками внутреннего алфавита (знаками, кодирующими внутреннее состояние автомата). В матрице переходов на пересечении строки $(x_k)$ и столбца $(q_r)$ помещаются значения функции $\Psi(q_r, x_k)$, а в матрице выходов — значения функции $\Theta(q_r, x_k)$.

**Пример 11.3.**

По заданному табличному представлению автомата требуется построить систему его команд.

Пусть конечный автомат имеет алфавиты: $X = \{a, b\}$, $Y = \{\alpha, \beta, \gamma\}$, $Q = \{q_1, q_2, q_3\}$, а автоматные функции задаются таблицами:

$\Theta(q, x)$

| $x$ | $q$ | | |
|---|---|---|---|
| | $q_1$ | $q_2$ | $q_3$ |
| $a$ | $q_3$ | $q_3$ | $q_1$ |
| $b$ | $q_2$ | $q_3$ | $q_3$ |

$\Psi(q, x)$

| $x$ | $q$ | | |
|---|---|---|---|
| | $q_1$ | $q_2$ | $q_3$ |
| $a$ | $\beta$ | $\alpha$ | $\alpha$ |
| $b$ | $\beta$ | $\gamma$ | $\gamma$ |

Представленные две таблицы можно объединить в одну, условившись в каждую клетку на первую позицию ставить значение $\Psi(q_r, x_k)$, а через запятую на вторую позицию помещать значение $\Theta(q_r, x_k)$. В результате получится следующая «сводная» таблица:

$\Psi(q, x)\Theta(q, x)$

| $x$ | $q$ | | |
|---|---|---|---|
| | $q_1$ | $q_2$ | $q_3$ |
| $a$ | $q_3, \beta$ | $q_3, \alpha$ | $q_1, \alpha$ |
| $b$ | $q_2, \beta$ | $q_3, \gamma$ | $q_3, \gamma$ |

Видно, что структура команды, а также функциональная таблица весьма напоминают те, с которой мы сталкивались при описании машины Тьюринга (см. п. 9.3.3): $q_j x_k \to q_i y_r$ (отличие только в отсутствии команды перемещения ленты). Из таблицы, в частности, легко просматриваются команды преобразования, осуществляемые данным автоматом:

$$1)\ q_1 a \to q_3\beta; \quad 3)\ q_2 a \to q_3\alpha; \quad 5)\ q_3 a \to q_1\alpha;$$
$$2)\ q_1 b \to q_2\beta; \quad 4)\ q_2 b \to q_3\gamma; \quad 6)\ q_3 b \to q_3\gamma.$$

Пусть на начальном такте автомат находится в состоянии $q_0 = q_1$ и на его вход в последующие такты подается последовательность $abab$. Пользуясь перечнем команд, можно установить, как автомат преобразует эту последовательность:

| | Такты | | | | | |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 |
| Вход $x$ | $-$ | $a$ | $b$ | $a$ | $b$ | $-$ |
| | | $\downarrow$ | $\downarrow$ | $\downarrow$ | $\downarrow$ | |
| Состояние $q$ | $q_1 \to$ | $\underbrace{q_1 a}$ | $\to$ | $\underbrace{q_3 b}$ | $\to$ | $\underbrace{q_3 a}$ | $\to$ | $\underbrace{q_1 b}$ | $\to q_2$ |
| Выход $y$ | $-$ | $\beta$ | | $\gamma$ | | $\alpha$ | | $\beta$ | $-$ |

Видно, что на выходе получится $\beta\gamma\alpha\beta$ и после этого автомат окажется в состоянии $q_2$. Стоит обратить внимание на то обстоятельство, что хотя на вход два раза подавался один и тот же сигнал, например $a$, выходные сигналы оказывались различными ($\beta$ и $\alpha$) — как указывалось выше, выходной сигнал автомата определяется не только сигналом на входе, но и внутренним состоянием.

Другой вариант описания автоматных функций — графический. Он обладает большей наглядностью, чем табличный. Состояния автомата $\langle X, Y, Q, \Psi, \Theta \rangle$ задается посредством ориентированного графа, который называется *диаграммой* (точнее, *диаграммой Мура*). Вершины графа помечены символами из алфавита состояний автомата $Q$, а каждой команде $q_i x_r \to q_j y_s$ соответствует ребро, идущее из вершины $q_i$ в вершину $q_j$; при этом ребру приписывается метка $x_r y_s$. Таким образом, ребро возникает в том случае, если автомат, находящийся в состоянии $q_i$, посредством некоторого входного сигнала $x_r$ может быть переведен в состояние $q_j$. Если такой перевод возможен при нескольких входных воздействиях $(x_r)^{(1)}, \ldots, (x_r)^{(n)}$ и при этом формируются выходные сигналы $(y_s)^{(1)}, \ldots, (y_s)^{(n)}$, то ребру приписывается выражение $((x_r)^{(1)}, (y_s)^{(1)}) \vee ((x_r)^{(2)}, (y_s)^{(2)}) \vee \ldots \vee ((x_r)^{(n)}, (y_s)^{(n)})$.

Для диаграмм, представляющих конечный автомат, справедливы следующие утверждения:

- из одной вершины не может выходить двух ребер с одинаковым входным символом (*условие однозначности*);
- если при работе автомата реализуется входная комбинация $q_i x_r$, то обязательно существует ребро, идущее из вершины $q_i$, помеченное символом $x_r$ (*условие полноты*);
- количество вершин и ребер диаграммы является конечным.

**Пример 11.4.**

Построить диаграмму Мура для описанного в 11.3 конечного автомата.

Если на начальном такте автомат находился в состоянии $q_0 = q_1$ и на его вход в последующие такты подавались сигналы $abab$, то, пользуясь диаграммой, можно проследить последовательность преобразований: $q_1 a \to q_3 b \to q_3 a \to q_1 b \to q_2$ — выходные сигналы будут появляться в порядке $\beta\gamma\alpha\beta$, что совпадает с результатом в примере 11.3.

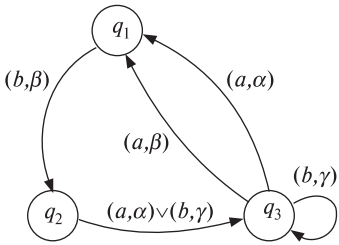



**Рис. 11.7.** Диаграмма Мура

## 11.3.2. Схемы из логических элементов и задержек

При создании конечных автоматов используется несколько типов элементов памяти, из которых основными следует считать *элемент задержки*, *двоичный триггер* и *двоичный счетчик*.

*Элемент задержки* (рис. 11.8,$a$) имеет один вход и один выход; функция перехода: $q(t_i) = x(t_i)$; функция выхода — $y(t_i) =$

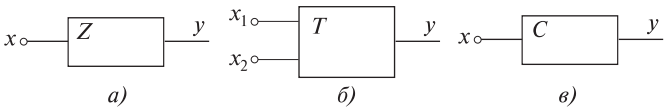



**Рис. 11.8.** Элементы памяти

$= q(t_{i-1})$. Из них, в частности видно, что $y(t_{i+1}) = q(t_i) = x(t_i)$, т. е. элемент осуществляет задержку входного сигнала на один такт (поданный на некотором такте сигнал поступит на выход лишь на следующем такте). Задержка используется в линиях обратной связи, когда, например, сигнал подается с выхода схемы на ее вход.

*Двоичный триггер* (рис. 11.8,*б*) имеет два входа $x_1$ и $x_2$. Функция переходов не зависит от внутреннего состояния элемента и определяется только сигналами на входе:

$$q(t_i) = \Psi(x_1(t_i), x_2(t_i)).$$

Значения функции устанавливаются следующим образом: $\Psi(1,0) = 1$, т. е. при подаче 1 на вход $x_1$ и 0 на вход $x_2$ триггер переходит в состояние 1 и сохраняет это состояние до тех пор, пока оно не будет заменено иным; $\Psi(0,1) = 0$, т. е. 0 на $x_1$ и 1 на $x_2$ переводят триггер в состояние 0; $\Psi_i(0,0) = \Psi_{i-1}$, т. е. при обоих нулевых сигналах состояние триггера не изменяется. Одновременная подача единичных сигналов на оба входа триггера исключается конструкцией схемы (см. пример 11.2). Выходная функция триггера при любых состояниях $y(t_i) = q(t_i)$, т. е. на выход подается текущее состояние триггера, но оно от этого не изменяется. Таким образом, триггер сохраняет сколь угодно долго записанный в него бит информации. Объединения подобных элементов образуют регистры и ячейки памяти, из которых, в свою очередь, строится вся память устройства. Память эта по количеству двоичных элементов, разумеется, *конечна*.

*Двоичный счетчик* (рис. 11.8,*в*) имеет один вход и один выход; он меняет свое состояние на противоположное, если на входе 1, и сохраняет его, если на входе 0. Очевидно, $q(t_i) = \Psi(x(t_i), q(t_{i-1}))$, причем $\Psi(1,1) = 0$, $\Psi(1,0) = 1$, $\Psi(0,1) = 1$, $\Psi(0,0) = 0$; $y(t_i) = q(t_i)$.

Добавлением элементов памяти к базису из логических элементов, использовавшихся при построении комбинационных схем, образует новый базис, который обладает *свойством полноты*.

> *Под* ***полнотой*** *понимается возможность построения на основе данного базиса схемы, задающей любое отображение последовательностей входных сигналов на последовательности выходных, которое вообще может быть задано дискретным двоичным преобразователем информации с конечной памятью.*

Другими словами, возможно построение любых схем, поведение которых определяется как входными сигналами, так и их состоянием на предыдущем такте.

Далее будем рассматривать схемы с единственным видом памяти — задержкой (на схемах будет обозначаться $Z$). Схемы строятся по определенным правилам. Исходными понятиями являются: элемент (логический или задержка) и *вход в схему* (он называется *полюсом*). Можно дать следующее *индуктивное* определение схемы (попутно определяется понятие *вершины* схемы):

- совокупность полюсов, соответствующая некоторому набору входных переменных $x_1, \ldots, x_n$, является схемой; вершинами схемы $y_1, \ldots, y_n$ будут ее полюсы (рис. 11.9,*а*);
- результат присоединения к полюсам всех входов некоторого базисного элемента есть схема; полюсами будут все исходные полюсы, а вершинами — все выходы элемента; в общем случае количество полюсов и вершин различно ($n \neq m$) (рис. 11.9,*б*);
- результат присоединения к вершинам схемы входов некоторого логического элемента есть схема; полюсами будут все полюсы исходной схемы, а вершинами — все свободные вершины исходной схемы и выходы присоединенного элемента (рис. 11.9,*в*);
- результат присоединения вершины схемы $y_j$ через задержку к некоторому полюсу $x_k$ есть схема; ее полюсами будут все полюсы исходной схемы, за исключением $x_k$, а вершинами — все вершины исходной схемы, кроме $y_j$ (рис. 11.9,*г*); при этом $y_j = \Theta_j(x_1, \ldots, x_n)$; в задержке $q_j = y_j$; после задержки и на $x_k = q_j(t_{i-1})$.

В соответствии с правилами построения схем каждой ее вершине может быть сопоставлена функция, указывающая значение сигнала на некотором такте $t_i$, а каждой задержке $Z_j$ — дополнительная функция, указывающая ее состояние на данном такте $q_j(t_i)$. Значения функций определяются по следующим правилам:

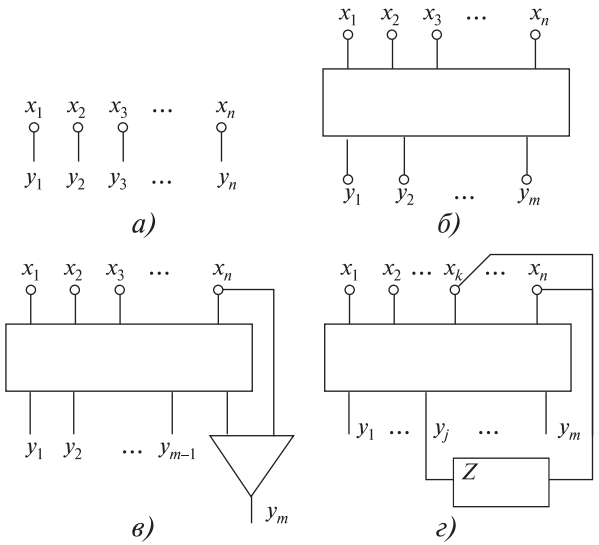



**Рис. 11.9.** Схемы из логических элементов

- полюсу $x_r$ приписывается дискретная функция $x_r(t)$, значения которой зависят от $t$ в такты 0, 1, 2,...;
- вершине, соответствующей выходу логического элемента (рис. 11.9,*в*), приписывается значение, сформированное этим элементом согласно сигналам на его входе;
- если элемент задержки$Z_j$ присоединен к вершине, которой приписана некоторая функция $f(t)$, то согласно значениям автоматных функций задержки для состояния задержки принимается $q_j(t_i) = f(t_i)$, а выходу задержки приписывается $q_j(t_{i-1})$; если задержка присоединяется к полюсу (рис. 11.9,*г*, то в функциях, приписываемых вершинам и состояниям задержек, функция $x_k(t_i)$ везде заменяется на $q_j(t_{i-1})$.

Любая правильно построенная схема из логических элементов и задержек обладает следующими двумя свойствами:

- каждый вход всякого элемента присоединен либо к полюсу, либо к выходу другого элемента;
- в любой циклической цепочке элементов присутствует, по крайней мере, один элемент задержки.

В отношении схем, содержащих элементы памяти, решаются задачи *анализа* и *синтеза*. Задача анализа состоит в том, чтобы по заданной схеме найти реализуемый ею автомат, т. е. установить все алфавиты, а также явный вид системы канонических уравнений. Задача синтеза является обратной по отношению к задаче

анализа: по заданному конечному автомату построить реализующую его схему над выбранным базисом (обычно это логические элементы и элементы памяти).

В основе анализа схем лежит *метод устранения задержек*. Его идея состоит в том, что из схемы можно удалить имеющиеся задержки, заменив их дополнительными полюсами, на которые поданы сигналы, соответствующие значениям функции, приписываемые задержке. В результате получается новая схема только из логических элементов с большим (по отношению к исходной задаче) числом полюсов и расширенным входным алфавитом. Для этой схемы строятся автоматные функции и тем самым устанавливается вид конечного автомата.

**Пример 11.5.**

Пусть задана схема (рис. 11.10,*а*), содержащая по одному логическому элементу И, ИЛИ, НЕ и два элемента задержки (на схеме обозначены $Z_1$ и $Z_2$), на вход которой подается сигнал $x$. Требуется описать работу автомата и получить последовательность сигналов на выходе, если на вход подается 10110.

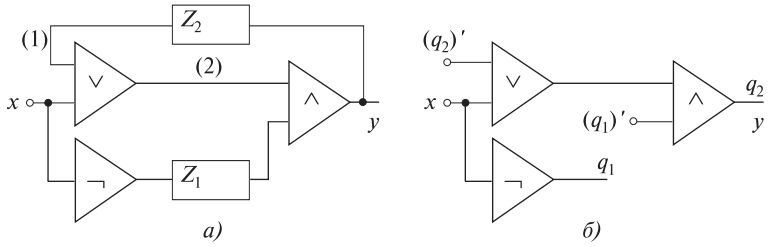



**Рис. 11.10.** Схемы к примеру 11.5

Обозначим внутреннее состояние задержек $q_1$ и $q_2$. Тогда согласно сформулированным выше правилами для выделенных вершин (1, 2 и 3) схемы можно записать:

полюс (1) $= q_2(t_{i-1})$;

вершина (2) $= x(t_i) + q_2(t_{i-1})$;

вершина схемы (3) $= y = q_1(t_{i-1}) \cdot (x(t_i) + q_2(t_{i-1}))$.

После удаления из исходной схемы задержек мы перейдем к схеме, представленной на рис. 11.10,*б*, которая теперь содержит два новых полюса (обозначим их $(q_1)' = q_1(t_{i-1})$ и $(q_2)' = q_2(t_{i-1})$) и одну новую вершину $(q_1)$; вершина $q_2$, образовавшаяся при удалении задержки $Z_2$, как видно из схемы, совпадает с $y$.

Схема реализует следующие автоматные функции:

$$y = q_2 = (q_1)' \cdot (x + (q_2)'); \quad q_1 = \bar{x},$$

поэтому канонические уравнения автомата будут иметь вид

$$\begin{cases} y(t_i) = q_1(t_{i-1}) \cdot (x(t_i) + q_2(t_{i-1})); \\ q_1(t_i) = \bar{x}(t_i); \\ q_2(t_i) = q_1(t_{i-1}) \cdot (x(t_i) + q_2(t_{i-1})). \end{cases}$$

Теперь можно построить таблицу значений для этой системы уравнений, считая все три полюса независимыми и задавая на них различные значения входных сигналов; очевидно, возможны 8 разных их сочетаний:

| Вход $x$ | Предыдущее состояние $Q$ | | Новое состояние $Q$ | | Выход $y$ |
|---|---|---|---|---|---|
| $x(t_i)$ | $q_1(t_{i-1})$ | $q_2(t_{i-1})$ | $q_1(t_i)$ | $q_2(t_i)$ | $y(t_i)$ |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 |

Далее строится таблица команд автомата подобно рассмотренной в примере 11.3. Ее строки и столбцы соответствуют входным сигналам $x(t_i)$ и состояниям на предыдущем такте $(q_1(t_{i-1}), q_2(t_{i-1}))$. В клетках размещаются состояния текущего такта $(q_1(t_i), q_2(t_i))$ и значение выходного сигнала $y(t_i)$. Введем сразу новые обозначения внутренних состояний автомата $(q^{(p)} = (q_1, q_2))$: $q^{(0)} = (00)$, $q^{(1)} = (01)$, $q^{(2)} = (10)$, $q^{(3)} = (11)$. То есть будем считать, что внутренние состояния автомата описываются двузначными комбинациями. Тогда окончательно получаем таблицу автомата:

| $x$ | $q$ | | | |
|---|---|---|---|---|
| | 00 | 01 | 10 | 11 |
| 0 | 10,0 | 10,0 | 10,0 | 11,1 |
| 1 | 00,0 | 00,0 | 01,1 | 01,1 |

$\Rightarrow$

| $x$ | $q$ | | | |
|---|---|---|---|---|
| | $q^{(0)}$ | $q^{(1)}$ | $q^{(2)}$ | $q^{(3)}$ |
| 0 | $q^{(2)}, 0$ | $q^{(2)}, 0$ | $q^{(2)}, 0$ | $q^{(3)}, 1$ |
| 1 | $q^{(0)}, 0$ | $q^{(0)}, 0$ | $q^{(1)}, 1$ | $q^{(1)}, 1$ |

Если начальным считать состояние (00) и подавать на вход $x$ последовательность 10110:

| | Такты | | | | | | |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Вход $x$ | – | 1 | 0 | 1 | 1 | 0 | – |
| Состояние $Q$ | 00 | 00 | 00 | 10 | 01 | 00 | 10 |
| Выход $y$ | – | 0 | 0 | 1 | 0 | 0 | – |

то, как видно, на выходе $y$ получается последовательность 00100.

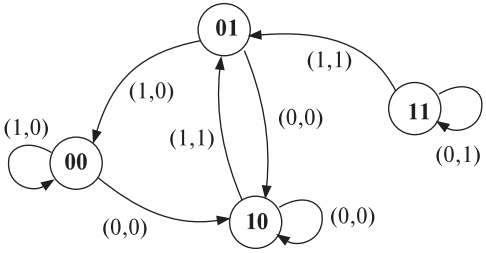



**Рис. 11.11.** Диаграмма Мура для примера 11.5

Диаграмма Мура приведена на рис. 11.11.

Задача синтеза автомата из логических элементов и элементов задержки решается в обратном порядке:

- заполняется таблица значений функций автомата (по принципу: «при условии подачи на вход таких-то комбинаций на выходе должно быть то-то»); другой вариант — представляются автоматные функции в виде таблицы или диаграммы;
- по таблице значений (или таблице автомата) строится система канонических уравнений автомата;
- из уравнений выявляется система булевских функций, описывающих работу автомата;
- по функциям определяется набор логических элементов и связей между ними;
- строится схема без задержек с промежуточными полюсами и вершинами;
- вводятся элементы задержки, устраняются промежуточные полюса и вершины.

**Пример 11.6.**

Построить автомат с единственным входом — «электронный замок», который выдавал бы на выходе 1 только тогда, когда на вход подается последовательность 101, а во всех остальных случаях на выходе 0.

*Анализ условия*: поскольку вход единственный, за один такт по нему может поступить только один сигнал; комбинация входных сигналов может формироваться только в последовательности тактов; чтобы отслеживать цепочку из трех сигналов, автомат должен сохранять два входных сигнала, предшествовавших текущему. Это означает, что схема должна включать две задержки — $Z_1$ и $Z_2$. Пусть для определенности $Z_1$ сохраняет сигнал на входе в предыдущем такте, а $Z_2$ — в предпредыдущем. Значит изначально схема будет иметь три независимых полюса: $x$, $q_1'$ и $q_2'$; новые внутренние состояния определяются из следующих соображений: на каждом следующем такте происходит перестановка: $q_1 = x$; $q_2 = q_1'$.

Таблица значений, очевидно, будет выглядеть следующим образом:

| $x(t_i)$ | $q_1'$ | $q_2'$ | $q_1$ | $q_2$ | $y(t_i)$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 |

Соответствующая таблица автоматных функций:

| $x$ | $q$ | | | |
|---|---|---|---|---|
| | 00 | 01 | 10 | 11 |
| 0 | 00,0 | 00,0 | 01,0 | 01,0 |
| 1 | 10,0 | 10,1 | 11,0 | 11,0 |

Появление на выходе 1 должно происходить только при значениях $x = 1$, $(q_1)' = 0$, $(q_2)' = 1$. Ясно, что должно использоваться логическое умножение, однако элемент И может обрабатывать только два сигнала, поэтому понадобятся два таких элемента. Кроме того, сигнал $(q_1)'$ нужно инвертировать, чтобы при верной комбинации на вход И подавалась 1.

Система канонических уравнений автомата будет выглядеть следующим образом:

$$(q_1)' = x(t_{i-1}); \quad (q_2)' = x(t_{i-2}); \quad y = (x \cdot q_2') \cdot \bar{q}_1'.$$

Можно записать по-другому:

$$q_1 = x; \quad (q_1)' = q_2; \quad y = (x \cdot q_2') \cdot \bar{q}_1'.$$

Схема без задержек с промежуточными полюсами приведена на рис. 11.12,*а*, а окончательная схема с задержками — на рис. 11.12,*б*. Диаграмма Мура приведена на рис. 11.13.

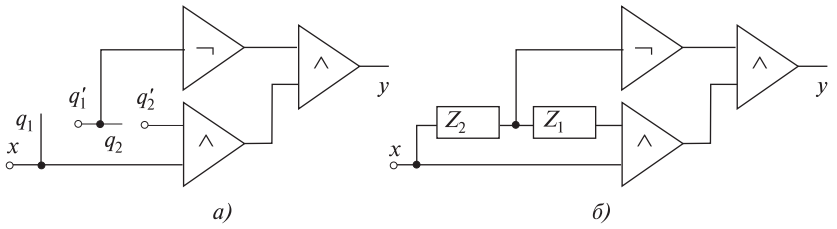



**Рис. 11.12.** Схемы к примеру 11.6

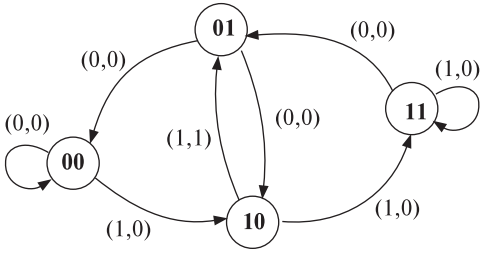



**Рис. 11.13.** Диаграмма Мура для примера 11.6

Таким образом, теория автоматов позволяет разработать алгоритмические методы перехода от этапа описания характера преобразований, которые должен осуществлять конечный автомат, к конкретным схемным решениям, основанным на использовании рассмотренного выше набора элементов. Это, в свою очередь, дает возможность задачу проектирования новых конечных автоматов формализовать для решения посредством другого конечного автомата, в частности, компьютера. Такая технология действительно существует и широко используется в практике создания новых устройств.

### 11.3.3. Эквивалентные автоматы

Автоматы являются устройствами для переработки дискретной информации. При этом характером перерабатываемой информации определяется входной и выходной алфавиты ($X$ и $Y$); алфавит внутренних состояний $Q$ определяется строением автомата и, вообще говоря, может различаться у разных автоматов с одинаковыми входными и выходными алфавитами. Следовательно, одно и то же преобразование информации может быть осуществлено автоматами с разным числом состояний и, следовательно, посредством различного числа команд. Введем ряд определений:

> *Состояния $q$ автомата $M$ и $q'$ автомата $M'$ считаются* ***эквивалентными,*** *если оба автомата, получив одну и ту же (любую) входную последовательность символов, перерабатывают ее в одинаковую выходную последовательность.*

> *Автоматы $M$ и $M'$ называются* ***эквивалентными,*** *если для каждого состояния автомата $M$ существует эквивалентное ему состояние автомата $M'$ и наоборот.*

Другими словами, эквивалентные автоматы реализуют одинаковые преобразования, но могут иметь различное число внутренних состояний.

Понятие эквивалентности состояний применимо и к одному автомату (формально можно считать, что $M$ и $M'$ совпадают). Для одного автомата эквивалентными будут различные состояния, через которые одна и та же входная последовательность символов преобразуется в одинаковую выходную.

В связи с синтезом схем практический интерес представляет задача построения автомата, выполняющего заданный набор преобразований при минимальном числе внутренних состояний.

*Автомат, эквивалентный заданному и имеющий наименьшее из всех возможных число состояний называется* ***минимальным.***

Задача построения минимального автомата называется *задачей минимизации автомата.* Ее решение осуществляется в два этапа: сначала устанавливаются все неэквивалентные состояния исходного автомата, а затем по ним стоится минимальный автомат. В свою очередь, для определения неэквивалентных состояний необходимо выявить классы эквивалентных состояний.

Пусть имеется автомат с множеством внутренних состояний $Q$, среди которых могут быть эквивалентные. Отношения эквивалентности состояний обладают обычными свойствами эквивалентности, т. е. рефлексивностью, симметрией и транзитивностью. Поэтому множество $Q$ может быть разбито на *классы эквивалентности.* Процедуру рассмотрим на примере.

**Пример 11.7.**

Пусть имеется конечный автомат, заданный таблицей:

| $x$ | $q$ | | | | | |
|---|---|---|---|---|---|---|
| $xq$ | $q_1$ | $q_2$ | $q_3$ | $q_4$ | $q_5$ | $q_6$ |
| 0 | $q_4$,1 | $q_2$,0 | $q_4$,1 | $q_4$,1 | $q_6$,0 | $q_5$,0 |
| 1 | $q_5$,0 | $q_2$,0 | $q_2$,0 | $q_3$,0 | $q_6$,0 | $q_6$,0 |
| 2 | $q_1$,0 | $q_6$,1 | $q_3$,0 | $q_3$,0 | $q_5$,1 | $q_2$,1 |

На основе ее составим другую таблицу (рис. 11.14), клетки которой будут соответствовать всем различным парам $q_iq_j$, $i \neq j$, заполнив ее согласно следующим правилам:

- если два состояния $q_i$ и $q_j$ неэквивалентны (т. е. для какого-либо значения входного символа $x$ значения на выходе различаются), то соответствующая клетка зачеркивается;
- если в состояниях $q_i$ и $q_j$ для каждого $x$ значения на выходе одинаковы, то в клетки записываются все пары состояний $q_vq_w$ $(v \neq w)$, отличные

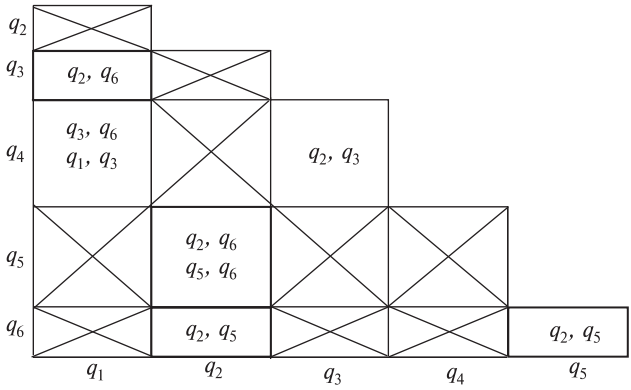



**Рис. 11.14.** Построение таблицы состояний автомата

от $q_iq_j$, в которые автомат может перейти из $q_i$ и $q_j$ при подаче одного и того же входного символа.

Согласно первому правилу зачеркнутой оказалась, например, клетка, соответствующая паре $q_1q_2$, поскольку при $x = 0$ на выходе выдаются разные значения (1 и 0). По второму правилу, например, для пары $q_5q_6$ следует записать пару $q_2q_5$ из следующих соображений: у $q_5$ и $q_6$ все выходные символы одинаковы при одинаковых входных; $x = 0$ приводит к исходной паре $q_5q_6$; $x = 1$ приводит к паре одинаковых состояний $q_6q_6$. Подобным образом анализируются остальные сочетания.

Наконец, последующими преобразованиями вычеркиваются клетки, в которых находятся пары, соответствующие уже зачеркнутым ранее клеткам. Например, следует зачеркнуть клетку для пары $q_1q_4$, поскольку в ней содержится $q_3q_6$, а также $q_3q_4$, так как в ней есть $q_2q_3$. Затем снова нужно зачеркнуть все клетки, которые содержат пары, соответствующие вычеркнутым клеткам. Процедура должна продолжаться до тех пор, пока не сформируется таблица, в которой нельзя вычеркнуть ни одной из оставшихся клеток. Для рассматриваемого примера эти клетки выделены жирной рамкой. Можно доказать, что *оставшиеся не вычеркнутыми клетки соответствуют всем парам эквивалентных состояний.* Это $q_1q_3, q_2q_5, q_2q_6$ и $q_5q_6$. Классы эквивалентности образуются состояниями, которые попарно эквивалентны. В нашем случае это $\{q_1, q_3\}$ и $\{q_2, q_5, q_6\}$. Состояния, не вошедшие в эти классы, эквивалентны лишь себе и образуют собственные классы эквивалентности; в рассматриваемом примере это $\{q_4\}$. Таким образом, классы эквивалентности оказались выделенными.

После выделения классов эквивалентности состояний для автомата $M$ можно построить эквивалентный ему автомат $M'$. В качестве входного и выходного алфавитов для $M'$ возьмем соответствующие алфавиты $M$, а каждому классу эквивалентных состояний $M$ сопоставим одно состояние

$M'$. Для рассмотренного выше примера можно принять $(q_1)' \Rightarrow \{q_1, q_3\}$, $(q_2)' \Rightarrow \{q_2, q_5, q_6\}$, $(q_3)' \Rightarrow \{q_4\}$. Окончательно получаем таблицу нового автомата $M'$.

$\Psi(q,x)\Theta(q,x)$

| $x$ | $q$ | | |
|---|---|---|---|
| | $(q_1)'$ | $(q_2)'$ | $(q_3)'$ |
| 0 | $(q_3)'$,1 | $(q_2)'$,0 | $(q_3)'$,1 |
| 1 | $(q_2)'$,0 | $(q_2)'$,0 | $(q_1)'$,0 |
| 2 | $(q_1)'$,0 | $(q_2)'$,1 | $(q_1)'$,0 |

Можно доказать следующую теорему*:

**Для всякого конечного автомата существует единственный эквивалентный ему минимальный автомат.**

**Существует алгоритм, который по любому заданному конечному автомату строит соответствующий ему минимальный.**

Можно доказать также, что рассмотренная нами процедура и является тем самым алгоритмом, позволяющим построить минимальный автомат эквивалентный заданному.

## Контрольные вопросы и задания к гл. 11

1. Почему для описания конечного автомата требуется задание двух автоматных функций? Возможны ли ситуации, когда функция преобразования будет единственной?

2. Почему рассматриваемые в теории автоматов устройства называются *дискретными*?

3. Постройте таблицу значений для схемы двоичного сумматора, изображенной на рис. 11.3.

4. Постройте комбинационные схемы, реализующие следующие логические функции, а также таблицы значений для них:

a) $y_1 = \neg(x_1 \wedge x_2)$; $y_2 = \neg x_1 \wedge x_2$;

b) $y = \neg((x_1 \vee x_2) \wedge (x_3 \vee x_4))$;

c) $y_1 = (\neg x_1) \wedge (x_2 \vee x_3)$; $y_2 = \neg x_1 \wedge x_3$;

d) $y = x_1 \oplus (\neg x_2)$;

e) $x_1 \sim x_2$.

5. Постройте комбинационную схему с тремя входами, которая при подаче на вход комбинации 110 выдавала бы на выходе 1, а при всех остальных входных комбинациях на выходе был бы 0.

---

* Тех, кого интересует доказательство, можно адресовать к книге Шоломова Л.А. [49, с. 125–126].

6. Постройте комбинационную схему с двумя входами и двумя выходами, которая ни при каких входных комбинациях не давала бы на выходе 00.

7. В чем состоит отличие комбинационных схем и конечных автоматов?

8. Можно ли считать конечным автоматом: (a) электронный будильник; (b) телевизор с пультом управления; (c) автомат для продажи газированной воды; (d) телефонный аппарат с наборным диском; (e) кнопочный замок, срабатывающий при одновременном нажатии комбинации клавиш; (f) табло с бегущей рекламой.

9. Задан конечный автомат с алфавитами $X = \{a_1, a_2\}$, $Y = \{b_1, b_2\}$, $Q = \{q_1, q_2\}$; автоматные функции определены в виде таблицы:

| | $q_1$ | $q_2$ |
|---|---|---|
| $a_1$ | $q_1, b_1$ | $q_2, b_2$ |
| $a_2$ | $q_2, b_1$ | $q_1, b_2$ |

Постройте систему команд автомата, а также представьте его диаграммой. На вход подано слово $a_1a_2a_2a_1$. Определите выходное слово.

10. Постройте таблицы автоматных функций, таблицы значений и диаграммы Мура для элемента задержки, триггера, двоичного счетчика.

11. Пользуясь методом устранения задержек, напишите систему канонических уравнений для автомата, схема которого приведена на рис. 11.15; постройте таблицу значений и таблицу автоматных функций, а также постройте диаграмму автомата. Какова будет выходная последовательность, если на вход подано 101110?

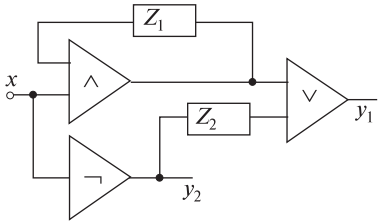



**Рис. 11.15.** Схема к заданию 11

12. Действие конечного автомата описывается таблицей:

| $x$ | $q_{i-1}$ | $q_i$ | $y_i$ |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 |

Постройте систему канонических уравнений, таблицу автоматных функций, диаграмму, а по ним — схему конечного автомата.

13. Постройте схему автомата аналогичного описанному в примере 11.6 при условии, что 1 на выходе должна появляться при последовательности входных сигналов 110. Постройте диаграмму автомата.

14. Почему описанная в данном разделе система элементов названа *полной*?

15. Какие автоматы называются *эквивалентными*? Какой автомат из эквивалентных является *минимальным*? Всегда ли возможно построение минимального автомата?

16. Постройте автомат, эквивалентный описанному в примере 11.5.

# 12 Модели и системы

Рассмотренные в предыдущей главе вопросы, связанные с формализацией представления алгоритмов, безусловно, имеют не только теоретическое значение. Выбор формы представления алгоритма является одним из этапов решения любой задачи, если решение предполагается осуществлять посредством какого-либо исполнителя. Однако начинается решение не с этого, а с *постановки* (формулировки) *задачи* на естественном языке. Примерами таких постановок могут быть: описать поведение тела, двигающегося в среде с сопротивлением; описать последствия ядерной войны; построить оптимальный вариант транспортных перевозок; спрогнозировать последствия сброса промышленных отходов в водоем и т. п. Будем считать, что такая постановка осуществлена; нас будут интересовать этапы решения задачи, следующие за постановкой, но предшествующие формальному представлению алгоритма. Другими словами, мы ходит проследить всю последовательность решения задачи с помощью исполнителя и, в частности, компьютера.

Как мы увидим, интересующие нас этапы связаны с *выделением систем* и *построением моделей*. Эти шаги тесно связаны друг с другом, поскольку выделение системы есть моделирование, а модель, в свою очередь, является системой. Тем не менее, прежде чем обсуждать связи между понятиями «*система*» и «*модель*», их взаимопроникновение и взаимообусловленность, рассмотрим их по отдельности.

## 12.1. Понятие модели

Переходим к обсуждению одного из наиболее популярных понятий, используемых буквально во всех научных дисциплинах и при решении многих прикладных задач. Широта применения термина приводит к тому, что оказывается затруднительным дать общее и универсальное для всех ситуаций его определение.

Поэтому мы поступим следующим образом: сначала наметим общие подходы, отражающие смысл модели и моделирования в любом приложении, а затем дадим более строгие определения математической модели и формальной системы.

### 12.1.1. Общая идея моделирования

Предположим, перед нами стоит задача описать поведение тела, падающего на землю. Для ее точного решения пришлось бы учитывать сопротивление воздуха при падении, неоднородности плотности земной поверхности, несферичность Земли, притяжение к Луне и другим планетам, суточное вращение Земли и т. д. Как и в примерах, приведенных выше, точное описание процесса падения требует рассмотрения поведения *сложного объединения* множества тел и учета различных факторов, которых, вообще говоря, может оказаться так много, что выделить и учесть их все просто не представится возможным и, следовательно, задача не будет решена. Если же мы некоторые факторы отбросим и тем самым упростим задачу, мы сможем ее решить, однако это будет уже не та исходная задача, которая ставилась изначально, а какая-то иная. И решение, полученное нами, будет лишь в большей или меньшей степени относиться к исходной задаче, но не являться ее точным решением. Другими словами, убедившись в сложности исходной задачи, мы оказываемся перед выбором: либо признать невозможность ее точного решения и отказаться от него, либо переформулировать задачу в упрощенном варианте и найти решение, заведомо сознавая, что для первоначальной задачи оно не обеспечит точного результата. Вполне естественно, что в подавляющем большинстве выбирается второй путь. Тем более что исходные данные, используемые при постановке и решении задачи, как правило, не являются абсолютно точными и содержат погрешности в силу ограниченной точности измерительных приборов, недостоверности сведений и пр., т. е. в принципе не могут обеспечить точного решения. Наконец, часть факторов, влияющих на результат решения, могут быть неизвестны или неизвестны их закономерности. Все это приводит к тому, что возникает практический интерес и необходимость нахождения даже неточного, приблизительного результата.

С другой стороны, всегда ли необходимо добиваться абсолютно точного решения? Безусловно, нет. Нам не требуется прогноз погоды с точностью, скажем, 0,01 °C — вполне приемлемым оказывается разброс ±2...3 °C. Нет нужды с секундной точностью

прогнозировать срок службы автомобиля или с точностью до одного человека предсказывать численность народонаселения страны через 10 лет. То есть постановка задачи должна содержать в качестве одного из параметров *точность*, с которой требуется или желательно иметь ее решение.

Примем следующие определения, которые будем рассматривать как предварительные (т. е. в дальнейшем они будут уточнены):

> *Сложное объединение сущностей и связей между ними, описание которого необходимо построить, будем называть* ***прототипом****.*

> ***Моделирование*** *— построение упрощенного варианта прототипа, обеспечивающего приемлемую для данной задачи точность описания его строения или поведения.*

Каким образом осуществляется моделирование? Оно основывается на том, что не все из многочисленных составляющих прототипа, взаимосвязей между ними и влияний внешних факторов оказываются одинаково значимыми в рамках поставленной задачи. Можно выделить (на уровне обоснования или просто догадки) *существенные* и *несущественные* для данной задачи составляющие сложного объединения, а затем последние отбросить или описать приблизительно. Точно также из взаимодействий (связей) как внутренних, так внешних, можно выделить те, которые играют в задаче основную роль, и отбросить второстепенные, малозначительные. Тем самым мы получим *новое объединение*, которое не будет полностью передавать свойства прототипа, но будет *проще* его — назовем его *моделью*.

> ***Модель*** *— это объединение составных частей (элементов) и связей между ними, отражающее существенные для данной задачи свойства прототипа.*

Комментарии к определениям:

1. Моделирование — это всегда упрощение. В этом смысл моделирования — замена чего-то сложного более простым. При этом описание становится неточным, зато появляется сама возможность описания и, следовательно, предсказания поведения системы — это очень важно с практической точки зрения.

2. Термином «*модель*» обозначают не любое объединение каких-то частей (элементов). Модель есть упрощенный вариант некого оригинала и без него построена быть не может. Нельзя создать модель чего-то несуществующего или неизвестного, на-

пример новой конструкции или внешности инопланетянина, — подробные построения являются проявлением творческой фантазии автора, но не моделями. Другими словами, модель всегда обладает некоторым прототипом.

3. Построение моделей является основным приемом изучения сложных объединений во всех естественных и общественных науках. Всегда при описании изучаемых наукой природных или социальных структур, явлений и процессов приходится прибегать к каким-то упрощениям, т. е. строить модели*. Таким образом, оказывается, что науки устанавливают законы не для реальных систем, а для их моделей. Изучение науки есть не что иное, как освоение ее моделей и правил их построения. Моделирование является единственным способом описания реального мира. Следовательно, моделирование — это один из основных методологических приемов научного познания.

4. Моделирование, т. е. упрощение, оказывается обязательным и неизбежным этапом решения любой научной или прикладной задачи, поскольку решение должно быть достигнуто за конечное время, что возможно лишь при использовании конечного числа данных, в то время как природные системы порождают, вообще говоря, неограниченное количество данных.

5. С построением моделей мы сталкиваемся постоянно и в повседневной жизни, как только пытаемся чего-либо объяснить кому-либо, например что увидели на улице, каков характер вашего друга или что поняли из объяснений преподавателя. Таким образом, люди живут в мире моделей; мы постоянно сталкиваемся с необходимостью их создавать и использовать.

6. Один прототип в общем случае может иметь множество моделей, поскольку по-разному можно выделить его существенные и несущественные стороны. Например, если прототипом является человек, то его моделями, очевидно, будут фотография, манекен, характеристика, словесное описание и пр. Другим примером могут служить модели, принятые в физике для описания движения — механика Ньютона, механика Лагранжа, механика Гамильтона–Якоби, релятивистская механика Эйнштейна, наконец, квантовая механика.

---

* Исключение составляет лишь те науки (разделы наук), предмет исследования которых не имеет прямого отношения к материальному миру, например математика.

7. В связи со множественностью моделей встает вопрос о том, какая из них лучше? Сравнение допустимо только для тех из них, которые могут быть сопоставлены с реальным оригиналом (ниже такие модели будут названы *проверяемыми*); обычно считается, что из таких моделей лучше та, которая ближе к оригиналу, т. е. описывает его поведение с меньшей погрешностью. Однако на практике не всегда оказывается, что наиболее точная модель будет наилучшей в данных условиях. Например, часто при решении задач управления в условиях реального времени оказывается гораздо важнее, чтобы модель позволяла получить результат быстро, хотя и не очень точно (поскольку более точные расчеты, как правило, требуют больше времени) с тем, чтобы в дальнейшем произвести коррекцию. Для моделей непроверяемых постановка вопроса о том, какая из них лучше, просто неприменима (см. п. 12.1.2).

8. Для построения моделей требуются какие-то средства, называемые инструментальными (или инструментами). Например, при создании макета здания используются ножницы, клей, бумага; фотоаппарат является инструментом для создания модели внешности человека; человеческий язык служит средством построения словесных моделей, которые мы используем при общении и объяснении.

Прежде, чем обсуждать аппарат, посредством которого модель может быть представлена (описана), рассмотрим основные классы моделей.

### 12.1.2. Классификация моделей

По ходу освоения данного учебника мы неоднократно занимались объединением каких-либо сходных по свойствам сущностей (пока будем называть их *объектами*) в различные группы (классы) в зависимости от того, обладают они или нет какими-то признаками или свойствами (например, выделяя типы сложных данных или методов описания алгоритмов). Такой прием используется весьма часто во всех без исключения сферах человеческой деятельности и называется *классификацией*.

> ***Классификация** — это распределение однотипных объектов в соответствии с выделенными свойствами (признаками, категориями, классами).*

Примерами известных научных классификаций являются классификации растений и животных, которую построили Линней и Дарвин; таблица Менделеева, классификация видов дви-

жения Энгельса. По-видимому, справедливо утверждение, что *классификация — один из универсальных и широко распространенных приемов, используемых при обработке информации в разных научных и прикладных дисциплинах.*

Может оказаться, что у объектов много (несколько) общих свойств, *независимых* друг от друга, — тогда распределять их можно по разным классификационным признакам. Например, студентов в группе можно разделить на две категории — юноши и девушки; можно по росту на категории: 150...159 см, 160...169 см, 170...179 см, меньше 150 см, больше 180 см; можно разделить по цвету волос на блондинов, шатенов и брюнетов и т. п. Следствием этого является *множественность классификаций* для одной и той же группы объектов.

Любая классификация начинается с выделения *общих* свойств (признаков) и, возможно, определения их значений, по которым объекты будут распределяться в различные группы (классы). Например, ручки, используемые для письма, по типу пишущего узла можно разделить на *шариковые, перьевые, гелиевые*; в свою очередь, перьевые можно разделить на *ручки с неметаллическим пером* (например, гусиным) и *с металлическим пером*, а среди них можно выделить *простые* (которые приходилось макать в чернильницу) и *автоматические*.

Представлена классификация может быть в графической форме (в виде графа) и в текстовой форме (в виде таблиц или списков).

Если описанную выше классификацию ручек представить в виде графа, она будет выглядеть так, как показано на рис. 12.1. Представление классификации в графической форме удобно в тех случаях, когда имеется один классификационный признак или одни свойства объектов оказываются следствием других.

Примером *табличной* классификации может служить распределение студентов по успеваемости (табл. 12.1).

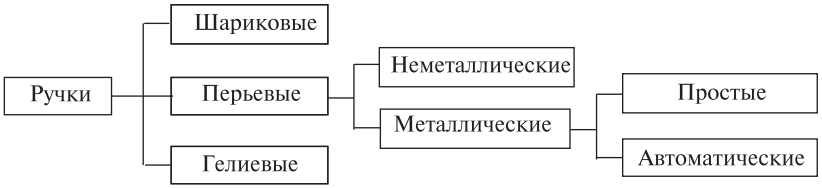



**Рис. 12.1.** Классификация ручек

Таблица 12.1

| Отличники | Хорошисты | Троечники | Двоечники |
|---|---|---|---|
| Примеров П.<br>Отличнова О.<br>Блесков Б. | Нормалева Н.<br>Хорошевская Х. | Таксебеев Т.<br>Удочкин У.<br>Середняков С. | Букин Б.<br>Бякин Н. |

Пример классификации в форме *списков*:

Гласные: а, е, ё, и, о, у, ы, э, ю, я.

Согласные: б, в, г, д, ж, з, й, к, л, м, н, п, р, с, т, ф, х, ц, ч, ш, щ, ь, ъ.

Используем рассмотренные подходы для классификации моделей с тем, чтобы выделить то их подмножество, которое изучается в информатике. Для этого необходимо установить одинаковые для больших групп моделей признаки и уже на основании этих признаков выделить те модели, которыми занимается информатика. Признаков этих несколько, поэтому предложить единую и универсальную классификацию невозможно и, следовательно, необходимо рассмотреть несколько классификаций по различным признакам.

**Модели структурные и функциональные.** *Состояние* прототипа — это совокупность свойств его составных частей, а также его собственных. Состояние — «моментальная» фотография прототипа для выбранного момента времени. С течением времени состояние может изменяться — тогда говорят о существовании *процесса*. В соответствии со сказанным возможно построение *модели состояния* и *модели процессов*. Модели первого типа называются *структурными моделями*, второго типа — *функциональными моделями*. Примерами структурных моделей являются чертеж какого-либо устройства, схема компьютера, блок-схема алгоритма и пр. Примерами функциональных моделей являются макет, демонстрирующий работу чего-либо; протез. Важнейшим классом функциональных моделей являются модели *имитационные*.

> ***Имитационное моделирование** — метод исследования, основанный на том, что изучаемый прототип заменяется его имитатором — натурной или информационной моделью — с которым и проводятся эксперименты с целью получения информации об особенностях прототипа.*

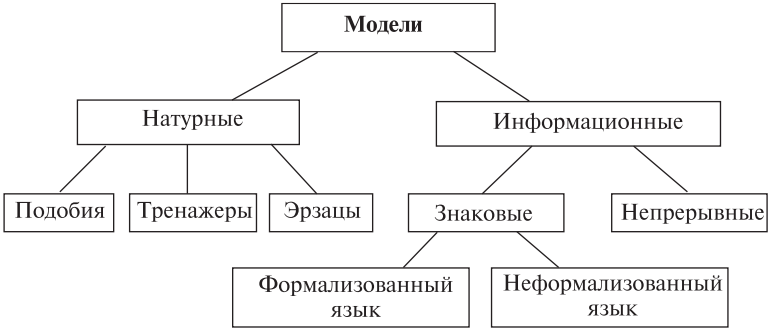



**Рис. 12.2.** Вариант классификации моделей

Примером натурной имитационной установки может служить аэродинамическая труба, позволяющая исследовать воздушные потоки, обтекающие транспортное средство (самолет, автомобиль, тепловоз и пр.) при движении. В качестве имитаторов могут выступать и математические модели, реализованные на компьютере. В настоящее время именно имитационное моделирование оказывается важнейшим методом исследования и прогнозирования в науке, экономике и др. Примерами является моделирование последствий ядерной войны, прогноз погоды, экономические прогнозы и пр.

**Модели натурные и информационные.** Модели можно отнести к одной из двух групп: *натурные* (материальные) и *информационные* (нематериальные). Классификацию по этому признаку удобно представить в графической форме (рис. 12.2).

Примерами натурных *моделей подобия* являются игрушка, манекен, фотография и т. п.

К натурным *моделям-тренажерам* следует отнести различные устройства, применяемые при подготовке летчиков, водителей и пр., которые имитируют некоторую ситуацию, требующую принятия решений и действий, и позволяют отрабатывать методы ее разрешения.

*Модели-эрзацы* — это, в первую очередь, протезы, заменяющие и частично выполняющие функции настоящих органов.

Примером *непрерывной информационной модели* может служить математическая функция и ее график.

*Знаковые информационные модели* можно было бы назвать также *дискретными*, подчеркивая то обстоятельство, что информация в них представлена в дискретной форме, т. е. посредством некоторого алфавита и языка. Язык может быть обычным

разговорным — с *неформализованным синтаксисом*. Примерами моделей, построенных посредством такого языка, являются различные описания, характеристики, мнения, толкования и пр.

Формализованные языки, как обсуждалось выше, имеют фиксированный набор лексических единиц (слов) и жесткий синтаксис фраз. Именно этим обеспечивается однозначность понимания смысла фраз и исполнения содержащихся в них указаний. Примером моделей, представленных посредством формализованных языков, может служить математическое описание существующего в природе или человеческом обществе явления, процесса; запись шахматной партии; нотная запись услышанных звуков и пр. Построение знаковой модели является обязательным этапом решения практической задачи с помощью компьютера; в дальнейшем, говоря о моделях в информатике, мы будем подразумевать именно *информационные знаковые модели*.

**Модели проверяемые и непроверяемые.** Ранее шла речь о возможности построения множества моделей для одного и того же прототипа. Всегда ли возможно соотнесения модели с прототипом и качества моделей между собой?

> ***Проверяемыми** являются те модели, у которых результат их использования может быть соотнесен (сравнен) с соответствующими параметрами прототипа.*

Например, построив математическую модель падения тела на землю и рассчитав в соответствии с ней время полета, можно провести эксперимент (или наблюдение) и сравнить с тем, что есть «на самом деле». Для таких моделей можно определить понятие *точности* как количественной меры, отражающей соотношение результатов моделирования и реальности. Ясно, что чем ближе будут данные, полученные в результате моделирования, к данным реального явления (процесса), тем точнее модель. Именно точность соответствия прототипу позволяют сопоставлять проверяемые модели между собой.

*Непроверяемые* модели сопоставить с реальным прототипом нельзя вообще или отсутствуют объективные (т. е. общие для всех) критерии такого сопоставления. Примером модели, которую невозможно соотнести с прототипом, является религия, если рассматривать ее как модель происхождения и устройства мира. Моделями с отсутствием объективных критериев сопоставления являются: характеристика человека как модель его качеств; трактовка замысла художника как модель произведения; описание увиденного; суждение о чем-либо и пр. Для одного прототи-

па допустимо построение множества непроверяемых моделей (например, множество религий или множество характеристик человека), однако невозможность их сравнения с оригиналом не позволяет также сравнивать их между собой и отдавать какой-либо из них предпочтение. Подобные модели должны приниматься как *равноправные*.

**Модели по назначению.** Различным может быть *назначение* моделей; можно выделить модели:

- *иллюстрационно-описательные* — для демонстрации строения или функционирования прототипа;
- *имитационные* — для исследования свойств прототипа и процессов в нем;
- *управленческие* — для осуществления управления прототипом.

В связи с приведенной классификацией возникает вопрос: к какому типу моделей следует отнести литературное произведение, картину художника, музыкальное произведение, макет нового здания или новую математическую теорию? Ответ зависит от того, имеется ли прототип у продукта творчества. Если этот продукт является вымыслом автора и прототип у него отсутствует, то такой продукт *нельзя считать моделью,* поскольку по определению модель — есть упрощение каких-то существующих реалий. При дальнейшем анализе художественного произведения (например, критиками или изучающими его студентами) может строиться его модель, однако само произведение моделью не является. Моделью событий (исторических, политических, уголовных и пр.) может служить только описание фактов без домыслов автора (именно такой подход принят в юриспруденции). Другими словами, летопись — это модель событий, а их художественное представление — нет (например, описание Бородинской битвы у Лермонтова или Толстого). Точно также нельзя называть моделями монтажную схему нового компьютера, блок-схему разрабатываемой программы или план создаваемого литературного или научного произведения — правильнее было бы использовать какой-то иной термин, например *проект*. Если же продукт творчества обладает прототипом (например, это портрет или пейзаж с натуры), то он может считаться моделью, причем это могут быть как натурные, так и информационные модели.

Несколько слов относительно термина «*математическая модель*», который в настоящее время используется не только в информатике, но и во множестве других научных и прикладных

дисциплин. В целом, математику следует считать наукой *самодостаточной*, поскольку ее понятия и теории строятся, в отличие от естественных наук, из исходной аксиоматики и законов внутренней логики, а не из необходимости интерпретации чего-то реального. Математическая модель может не иметь прототипа, так же как понятие «*функция*» не связывается с какой-то реальной зависимостью между величинами, а понятие «*квадрат*» с реальными площадями. Поэтому математической моделью (в широком толковании термина) считается любое описание задачи с использованием формализма математики и логики, безотносительно существования прототипа. К таким моделям следует отнести математическое описание процессов в атмосфере, в экономической системе или при управлении полетом ракеты, т. е. тех, что существуют реально. Но математическое описание лежит и в основе компьютерных игр, графических редакторов, систем проектирования и пр. С точки зрения создания и дальнейшей эксплуатации компьютерной программы между первой и второй группой задач различий нет — программа строится по математической модели независимо от наличия прототипа; обращение к прототипу при необходимости выполняется лишь при анализе результатов использования программы.

Рассмотренная классификация, как и любая иная, не исчерпывает всего многообразия существующих моделей и отдельных их особенностей. Однако для нас важным должно быть понимание того, что решение практической задачи на компьютере с неизбежностью требует построения информационной знаковой проверяемой модели.

### 12.1.3. Понятие математической модели

Обсудим частное определение модели, принятое в математике; в дальнейшем будем называть его *математической моделью в узком значении термина*:

> ***Математическая модель** — это множество элементов произвольной природы, на которых определено конечное множество отношений.*

Будем обозначать $M = \{m_1, m_2, \ldots\}$ — множество элементов (его называют *несущим множеством*), $R = \{R_1, R_2 \ldots R_n\}$ — множество отношений между ними.

Определение, безусловно, нуждается в комментариях.

1. В математической модели не конкретизируется, каков характер (природа) элементов множества $M$ — он может быть лю-

бым. Кроме того, отсутствует требование дискретности этого множества, т. е. в общем случае оно может содержать бесконечное количество элементов; примером может служить множество вещественных чисел, для которых определены отношения $a > b$ или $a^2 = b$.

2. На одном и том же множестве могут быть построены различные модели, если будут выделены различные группы отношений. Например, учебную группу можно рассматривать как объединение субъектов с их межличностными отношениями, но можно выделить информационные отношения, имущественные, родственные и пр. Соответственно, будут строиться и описываться различные математические модели.

3. Характер отношений между элементами множества определяется свойствами, которыми элемент может обладать или, наоборот, не обладать. Например, для множества людей определено отношение «*быть другом*». Для каждого отдельного элемента (т. е. человека) существует большее или меньшее число элементов из того же множества, обладающие набором свойств (качеств), позволяющих установить указанное отношение; но имеются и те элементы, которые необходимым свойством не обладают, и данное отношение не устанавливается. Таким образом, отношения определяются атрибутами элементов множества: $R_k = R_k(a_1, \ldots, a_p)$. Важно, что *число отношений и количество атрибутов конечны*.

4. Отношения между элементами множества могут носить парный (бинарный) и непарный характер. Например, для элементов множества целых чисел определены парные отношения $x_i = x_{i-1} + 1$, $x_i = x_{i+1} - 1$. Они и множество целых чисел определяют одну из возможных математических моделей для данного множества. В качестве примера непарных отношений можно рассмотреть отношение $ax + b = 0$ для некоторых троек чисел $a, b, x$ (при $a \neq 0$) из множества рациональных чисел справедливо, следовательно, они также образуют математическую модель (в отличие от остальных троек, которые этому отношению не удовлетворяют и, таким образом, в данную модель не входят).

Для описания математических моделей используются языковые и графические средства. Язык описания может быть формализованным (например, язык математических формул) или естественным. Графическая форма, как всегда, обеспечивает удобство общего обзора модели, однако наглядность эта проявляется только в случае бинарных отношений; если отношения в модели

не являются бинарными, изображать модель в виде графа становится неудобно и для их представления используются языковые средства.

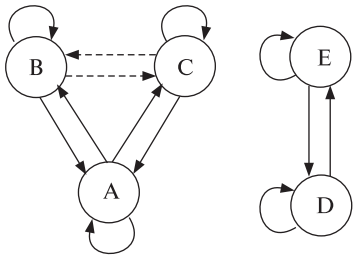



**Рис. 12.3.** Граф модели

Рассмотрим графическую форму модели, соответствующей следующему словесному описанию: «*A учится в одной группе с B и C, но не с D и E, которые учатся в другой группе*» (рис. 12.3). Здесь $M = \{A, B, C, D\}$, а отношением $R$ будет «*учиться в одной группе*». Вершинами графа являются элементы несущего множества, а его дугами — отношения.

Отношения между элементами множества $R_k$ могут обладать различными свойствами, но важнейшими из них являются три: *рефлексивность*, *симметричность* и *транзитивность*.

> *R обладает свойством* ***рефлексивности****, если любой элемент M, на котором R определено, вступает в отношение с самим собой.*

Отношение «*учиться в одной группе*» обладает свойством рефлексивности, поскольку каждый студент учится сам с собой в одной группе. На графе свидетельством рефлексивности являются дуги, начинающиеся и заканчивающиеся на одном и том же элементе. Другим примером рефлексивного отношения является «*равенство*» (если $a = b$, очевидно, $a = a$). Примером нерефлексивных отношений могут служить «*больше*» ($a > b$) или «*быть родителем*» (очевидно, что ничто не порождает самое себя).

> *R обладает свойством* ***симметричности****, если из того, что элемент $m_1$ множества M связан этим отношением с элементом $m_2$, следует, что обязательно и $m_2$ связан с $m_1$ тем же отношением.*

На графе симметричность отношения видна в том, что дуги, связывающие вершины, являются парными и противоположно направленными. Рассматриваемое в данном примере отношение симметрично, поскольку, если $A$ учится с $B$ в одной группе, то, очевидно, и $B$ учится с $A$ в одной группе. Примерами несимметричных отношений могут служить «*быть начальником*», «*быть родителем*», «*больше*». На графе несимметричное отношение изображается одинарной направленной дугой.

Наконец,

*Отношение $R$* ***транзитивно****, если из того, что этим отношением связаны $m_1$ и $m_2$, а также $m_1$ и $m_3$, следует, что между $m_2$ и $m_3$ имеется то же отношение.*

Очевидно, рассматриваемое отношение транзитивно, что отражено парными пунктирными дугами, связывающими $B$ и $C$. Примером нетранзитивного отношения является «*быть родителем*»: если справедливо, что $X$ является родителем $Y$, а также $X$ является родителем $Z$, то $Y$ и $Z$ таким отношением не связаны.

Если некоторое отношение $R$ обладает одновременно свойствами рефлексивности, симметричности и транзитивности, то говорят, что оно является отношением *эквивалентности*. Такие отношения разбивают множество $M$ на непересекающиеся *классы эквивалентности* — это видно из нашего примера: классами эквивалентности оказываются $A, B, C$ и $D, E$, поскольку они учатся в разных группах, но связаны одним типом отношений.

Помимо рассмотренных свойств отношений возможны противоположные им свойства — *антирефлексивность*, *антисимметричность* и *нетранзитивность*. Существование таких обратных свойств означает отсутствие прямого свойства в отношениях между *любой парой* элементов $M$.

Комбинацией свойств из приведенной шестерки (прямых и обратных) можно охарактеризовать различные отношения. Например, можно показать, что отношение «$\leqslant$» является рефлексивным, транзитивным и антисимметричным; отношение «$<$» — транзитивным, антисимметричным и антирефлексивным.

Математические модели в узком значении термина широко применяется в теории принятия решений, математической лингвистике, представлении знаний и ряде других разделов информатики. Однако, как уже указывалось, чаще термин «*математическая модель*» используется в широкой трактовке как *описание задачи с использованием формализма математики.*

## 12.2. Понятие системы

### 12.2.1. Определение объекта

Пожалуй, трудно указать сферу человеческой деятельности, в которой не применялся бы термин «*система*»: мы постоянно слышим о системах политических, экологических, банковских, геологических, космических; о системах образования, медицинского обслуживания, социального обеспечения и т. п. Можно сказать, что любая наука занята изучением «своих» систем: биоло-

гических, физических, социальных, информационных, инженерных, экономических и пр. и, на первый взгляд, они представляются совершенно различными. Это утверждение верно, если мы будем рассматривать конкретные свойства и особенности систем. Однако, несмотря на частные различия, выделяются некие общие признаки, по которым можно судить, имеем ли мы дело с системой или же нет. Существование этих признаков позволяет построить общее (хотя и не абсолютно строгое) определение системы, которое в дальнейшем мы уточним, формализовав его.

Понятие «*система*» может рассматриваться в качестве уточнения и конкретизации понятия «*сложное*». Альтернативой понятию «*сложное*» является понятие «*простое*». Он применяется к тем сущностям, в отношении которых *неприменимы* вопросы: «*Из чего состоит*?» или «*Как устроено*?». Простое не состоит из чего-либо и никак не устроено — это *конечный уровень проникновения вглубь строения*. Другими словами, простое — исходные «*кирпичи*», из которых складывается нечто сложное. Для обозначения этих исходных простейших сущностей будем использовать термин *объект*.

> ***Объект*** *— простейшая составляющая сложного объединения, обладающая следующими качествами:*
> • *в рамках данной задачи он не имеет внутреннего устройства и рассматривается как единое целое;*
> • *у него имеется набор свойств (атрибутов), которые изменяются в результате внешних воздействий;*
> • *он идентифицирован, т. е. имеет имя (название).*

Комментарии к определению:

1. «*. . . в рамках данной задачи. . .* » означает, что одна и та же сущность в одних задачах может рассматриваться в качестве простой (т. е. объекта), а в других — нет. Например, отдельное предприятие в рамках экономики государства может считаться простым элементом, т. е. объектом, с некоторым набором параметров, для государства существенных: характер и объем выпускаемой продукции, местонахождение, потребности в ресурсном обеспечении, количество работников и т. д. При этом не включается в рассмотрение структура производства, количество производственных помещений, личности руководителей, цвет знаний и пр. В другой же задаче, где требуется найти оптимальную схему производства конкретного предприятия, рассматривать его как простой элемент, безусловно, нельзя. Проникновение вглубь устройства чего-либо, вообще говоря, безгранично, поэтому всег-

да приходится останавливаться на некотором «уровне простоты», который приемлем для данной задачи. Таким образом, отнесение каких-то составляющих сложного объединения к объектам есть не что иное, как упрощение реальной ситуации, т. е. моделирование. Объект — модельное представление. Выделение объектов из составляющих сложного объединения выполняется на этапе построения модели при решении практической задачи.

2. Объекты, безусловно, могут обладать свойствами. Понятие свойства определим следующим образом:

> ***Свойством (атрибутом)*** *называется качество объекта, для которого установлена мера; сама мера называется* ***значением атрибута****.*

Наличие меры означает, что, во-первых, имеется качественная или количественная шкала, в соответствии с которой атрибуту приписывается значение (величина); во-вторых, определен порядок соотнесения атрибута с этой шкалой (порядок измерения). Например, цвет объекта мы определяем по качественной шкале с использованием градаций «*красный*», «*черный*», «*зеленый*» и пр., относя цвет к определенной части спектра; соотнесение со шкалой в данном случае реализуется по субъективному восприятию и, следовательно, не является строгим и однозначным, т. е. необъективно. Другим подобным примером является отметка, которую преподаватель выставляет учащемуся; отметку можно считать атрибутом знаний, отнесенным к принятой шкале («2», «3», «4», «5»); поскольку процедура измерения не определена однозначно, значение атрибута также необъективно. Примерами объективной (т. е. одинакового для всех) установки значения атрибута является определения меры нагретости тела посредством термометра или размеров тела посредством линейки — в обоих случаях имеется шкала, позволяющая однозначно охарактеризовать атрибут количественно. Даже из приведенных примеров видно, что свойства объекта могут определяться различными величинами: цвет задается словом, школьная отметка — целым числом, температура и длина — числом вещественным. Таким образом, мы приходим к необходимости использования различных *типов величин* (типов данных) для описания свойств объекта, о чем шла речь в гл. 8.

3. Набор свойств объекта будем называть *полем свойств*. В поле свойств выделяются две составляющие: свойства индивидуальные и общие. К индивидуальным свойствам относятся те, которые выделяют данный объект из множества ему подоб-

ных. Например, если в качестве объектов рассматривать автомобили одной марки, то индивидуальными атрибутами оказываются цвет, год выпуска, пробег. К общим свойствам относятся те, которые обеспечивают принадлежность к данной модели автомобиля — это оказывается общим свойством объектов.

> ***Класс** — это множество объектов, обладающих одним или несколькими одинаковыми атрибутами; эти атрибуты называются **полем свойств класса**.*

Среди атрибутов объекта всегда имеются те, которые определяют характер его связей (взаимодействия) с другими объектами и, следовательно, оказываются существенными для объединения объектов; и, наоборот, часть атрибутов объекта для объединения может быть несущественной. Например, учебная группа объединяет людей, поступивших в одно время в данное учебное заведение; несущественными оказываются рост, цвет глаз и волос и прочие индивидуальные качества.

4. Свойства объекта могут изменяться с течением времени. Изменение свойства — это процесс. Любой процесс имеет причину (*«движущую силу»*). Для объекта причины протекающих процессов могут быть только внешними по отношению к нему, поскольку согласно определению внутреннее устройство и, соответственно, какие-либо внутренние воздействия и причины у объекта отсутствуют. Внешние воздействия (причины) могут носить постоянный (непрерывный во времени) характер (например, притяжение к Земле) или быть дискретными — в этом случае их называют событиями (например, толчок тела или поступление порции информации). Реакцией объекта на внешнее воздействие является изменение его свойств.

5. Важной характеристикой процесса является его скорость протекания, т. е. изменение свойства за единицу времени. Изменяются, вообще говоря, все свойства объекта, однако скорости процессов, безусловно, различны. При этом, если относительное изменение некоторого свойства за время наблюдения незначительно, то говорят о *постоянстве свойства*, т. е. независимости его от времени. Например, цвет одежды в течение первых нескольких недель носки практически не изменяется. Однако в результате длительной эксплуатации и стирки цвет претерпевает изменение. Таким образом, постоянство (неизменность) свойства — это, безусловно, модельное представление, принимаемое в рамках данной задачи.

6. Описание любого объекта начинается с присвоения ему

идентификатора, т. е. имени — без него невозможно указать, какая сущность рассматривается. Имя (название) объекта (No) является его индивидуальным признаком, который, однако, нельзя считать свойством, поскольку оно не имеет меры. Имя (название) класса (Nc) является общим признаком для группы объектов. Например, в электронных схемах можно выделить классы с именами «резистор», «конденсатор», «микросхема» и др. Именем отдельного объекта будет «резистор 470 кОм». Имя объекта или класса не может изменяться с течением времени ($No \neq No(t)$, $Nc \neq Nc(t)$); изменение имени (переименование) следует рассматривать как прекращение существования одного объекта (класса) и возникновение другого.

### 12.2.2. Определение системы

Вернемся к соотнесению понятий «*простое*» — «*сложное*». Если нечто мы определяем как «*сложное*», то подразумеваем, что оно имеет какое-то строение, т. е. из чего-то состоит. В дальнейшем эту составную часть сложного будем называть *компонентом*. Очевидно, компоненты могут быть двух типов:

- те, которые в данной задаче можно считать простыми, т. е. *объекты*;
- *сложные*, т. е. те, которые в свою очередь состоят из чего-то еще.

Теперь можно попытаться определить понятие *система*.

> ***Система** — совокупность взаимодействующих компонентов, каждый из которых в отдельности не обладает свойствами системы в целом, но является ее неотъемлемой частью.*

Комментарии к определению:

1. Системой может называться не любая совокупность (объединение) неких сущностей, а только сущностей взаимодействующих, т. е. связанных друг с другом. Например, груду кирпичей или набор радиодеталей считать системами нельзя; если же эти кирпичи разместить в определенном порядке и связать раствором, а радиодетали нужным образом соединить между собой, то получатся системы — дом и телевизор. Следствием взаимодействия оказывается то, что компоненты системы определенным образом организованы, т. е. система имеет структуру, отражающую ее организацию (устройство). Взаимодействия (связи) могут быть различной природы: механические, физические, информационные и др. К способам описания структуры необходимо

отнести языковый (с использованием естественного или формализованного языка) и графический.

2. Любая система обладает двумя качествами: *системности* и *единства*:

- *системность* означает, что при объединении компонентов возникает некоторое новое качество — *системное свойство* — которым изначально не обладали отдельные компоненты; в рассмотренном выше примере с телевизором совершенно очевидно, что никакая его деталь (компонент) по отдельности не обладают свойством демонстрации изображения и звука, перенесенных радиоволнами;
- *единство* или, по-другому, *целостность* системы означает, что удаление из нее какого-либо компонента приводит фактически к ее уничтожению, поскольку меняется (или исчезает) системное свойство (в этом легко убедиться, если из телевизионной схемы убрать какую-либо деталь).

3. Уточним терминологию: предельно простые компоненты системы далее будем называть *объектами*; сложные, которые также состоят из связанных простых (и, следовательно, подпадают под определение системы), будем называть *подсистемами*. Например, двигатель является подсистемой автомобиля, а болт — объектом.

4. Понятия «*система*» и «*модель*» неразрывно связаны друг с другом. Выделение, изучение и описание каких-либо систем неизбежно сопровождается моделированием, т. е. упрощениями, причем, моделирование осуществляется на двух уровнях. На внешнем уровне выделяется сама система: поскольку любое реальное объединение (прототип системы) включает множество составляющих и связей между ними, на этапе постановки задачи приходится какие-то из них включать в систему и рассматривать далее, а какие-то отбрасывать как второстепенные. На внутреннем уровне моделирование состоит в том, что часть составляющих системы принимаются и рассматриваются в качестве объектов, что, как указывалось выше, также является упрощением. Кроме того, пренебрегается некоторыми внутренними взаимосвязями. Таким образом, в задачах, связанных с изучением и описанием сложных объединений, система — это модельное представление. Однако это утверждение не будет справедливым для задач, в которых системы создаются искусственно (т. е. человеком), — технические конструкции и механизмы, здания, художественные произведения, компьютерные программы и пр., —

порождаемые фантазией автора, они не имеют прототипов и, следовательно, не могут быть моделями, хотя подпадают под определение системы. С другой стороны, модель сложного прототипа также представляет собой объединение связанных составных частей, т. е. *модель является системой*. Однако модель объекта, очевидно, системой быть не может. Следовательно, несмотря на связь понятий «*система*» и «*модель*», их нельзя отождествлять; соотношение этих понятий определяется характером решаемой задачи.

5. Приведенное определение является инвариантным по отношению к области знаний или технологий, в которой система исследуется или создается. Другими словами, степень общности определения высока.

На практике необходимость выделения систем связаны с постановкой и решением следующих задач:

- *изучение прототипа системы*, т. е. выяснение строения природного или искусственного прототипа системы, особенностей связей между компонентами, влияния внешних и внутренних факторов на характер протекающих процессов;
- *описание системы*, т. е. представление системы языковыми или графическими средствами;
- *построение системы* — создание новой системы из компонентов;
- *использование системы* — решение с помощью системы каких-то проблем практики.

При решении перечисленных системных задач используются два метода — *анализ* и *синтез*.

> ***Анализ** — метод исследования, основанный на выделении отдельных компонентов системы и рассмотрении их свойств и связей.*

Анализ — это *декомпозиция* (расчленение) сложного объединения на составные части и рассмотрение их и связей между ними по отдельности. В информатике имеется раздел (это и самостоятельная научная дисциплина) — *системный анализ*, в котором изучаются способы выделения, описания и исследования систем. В то же время анализ является универсальным методом познания, применяемым во всех без исключения научных и прикладных дисциплинах. Его альтернативой и дополнением является *синтез*.

> ***Синтез*** *— (1) метод исследования (изучения) системы в целом (т. е. компонентов в их взаимосвязи), сведение в единое целое данных, полученных в результате анализа; (2) создание системы путем соединения отдельных компонентов на основании законов, определяющих их взаимосвязь.*

Синтез — это объединение составляющих для получения нового качества (системного свойства). Такое объединение возможно только после изучения свойств компонентов и закономерностей их взаимодействий, а также изучения влияния различных факторов на системные свойства. Синтез — целенаправленная деятельность человека, следовательно, его результатом будет *искусственная* система (в отличие от природных *естественных*). Система может создаваться с конечной целью изучения и описания ее прототипа — подобную систему, как было сказано выше, следует считать моделью. Примером может служить упоминавшаяся ранее имитационная модель процессов в атмосфере Земли, на основании которой прогнозируется погода. Другой целью создания (построения) системы может быть ее практическое использование для удовлетворения каких-либо потребностей человека, например сооружения, транспортные средства, электронные устройства. Эти системы нельзя считать моделями, поскольку отсутствуют их прототипы. Однако они сами являются прототипами для чертежей и схем, по которым создаются. К этой же категории искусственных систем необходимо отнести художественные произведения, компьютерные программы и другие построения, выполненные посредством некоторого языка (естественного или формализованного) и имеющие смысловую завершенность. Задачи анализа и синтеза мы решали, в частности, в отношении конечных автоматов (см. гл. 11).

*Использование* системы — это конечная цель ее изучения или создания. Часто использование связано с *управлением* системой; общие законы управления системами изучает *кибернетика*.

Прежде чем выделить различные классы систем, произведем ряд терминологических уточнений. Полный набор свойств системы — *поле свойств системы* — составляют поля свойств ее отдельных компонентов, а также системные свойства. В дальнейшем из индивидуальных свойств компонентов будем включать в поле свойств системы лишь те, которые оказываются *существенными* для системы, т. е. определяют характер связей (отно-

шений) с другими компонентами или внешними по отношению к системе телами. Таким образом, на данном этапе обсуждения мы можем каждой системе поставить в соответствие три множества: множество компонентов $\{A\}$, множество отношений между ними $\{R\}$, а также множество (поле) свойств системы $\{P\}$.

Рассмотрим некоторые признаки, которые могут быть положены в основу классификации систем.

**Системы статические и динамические.**

*Система называется* ***статической****, если множества* $\{A\}$, $\{R\}$ *и* $\{P\}$ *не меняются с течением времени.*

Неизменность $\{A\}$ и $\{P\}$ означает постоянство состава системы и поля ее свойств. Неизменность $\{R\}$ означает постоянство структуры системы.

Если любое из перечисленных множеств изменяется, то система будет *динамической*; изменение всегда сопровождается *процессом* (или несколькими процессами).

Статическую систему иногда рассматривают как *мгновенное состояние* системы динамической.

Примером статической системы может служить организационная структура учреждения; динамической — само предприятие в его развитии.

Частным случаем статических систем являются системы *равновесные*; их особенность в том, неизменность системы достигается за счет нескольких процессов, идущих в противоположных направлениях и уравновешивающих друг друга. Примерами могут служить система «*вода–насыщенный пар*», равновесие в которой достигается процессами испарения и конденсации; экологическая система с равновесием хищных и нехищных животных; система «*человек*» или «*животное*» с уравновешивающими друг друга процессами ассимиляции и диссимиляции; предприятие или целое государство, в которых сбалансированы доходы и расходы. Таким образом, статичность системы не тождественно отсутствию в ней процессов.

**Системы замкнутые и незамкнутые.** Совершенно очевидно, что помимо объектов и других компонентов, входящих в систему, имеются иные сущности, которые в систему не включены и являются внешними по отношению к ней. Компоненты системы могут взаимодействовать с внешним окружением или этого взаимодействия может не быть (в этом случае взаимодействие осуществляется только между компонентами системы).

*Система называется* ***замкнутой*** *(изолированной*), если ее компоненты не взаимодействуют с внешними сущностями, а также отсутствуют потоки вещества, энергии и информации из системы или в нее.*

Примером физической замкнутой системы может служить горячая вода и пар в термосе. В замкнутой системе количество вещества и энергии остается неизменным. Количество же информации может изменяться как в сторону уменьшения, так и увеличения, — в этом просматривается еще одна особенность информации как исходной категории мироздания. Замкнутая система является некоторой идеализацией (модельным представлением), поскольку полностью изолировать какую-то совокупность компонентов от внешних воздействий невозможно.

Построив отрицание приведенному выше определению, мы получим определение системы *незамкнутой*. Для нее должно быть выделено множество внешних воздействий $\{E\}$, оказывающих влияние (т. е. приводящих к изменениям) на $\{A\}$, $\{R\}$ и $\{P\}$. Следовательно, незамкнутость системы всегда связана с протеканием процессов в ней. Внешние воздействия могут осуществляться в форме каких-то силовых действий либо в форме *потоков* вещества, энергии или информации, которые могут поступать в систему или выходить из нее. Примером незамкнутой системы является какое-либо учреждение или предприятие, которые не могут существовать без материальных, энергетических и информационных поступлений. Очевидно, исследование незамкнутой системы должно включать изучение и описание влияния на нее внешних факторов, а при создании системы должна предусматриваться возможность появления этих факторов.

**Системы естественные и искусственные.** Различие осуществляется по тому, имеется ли у системы природный прототип или нет.

***Естественными*** *называются системы, имеющие прототип природного происхождения.*

***Искусственные*** *— это системы, созданные человеком.*

Выделение системы из природного образования неизбежно связано с принятием упрощающих и ограничивающих положений; по этой причине естественная система является моделью и

* В физике понятия замкнутой и изолированной систем не являются тождественными, однако для нашего рассмотрения их различия несущественны.

отражает свойства прототипа неточно. Искусственная система строится в соответствии с замыслом человека и может точно этому замыслу соответствовать.

### 12.2.3. Формальная система

Обсуждавшее выше понятие системы является общим и универсальным, т. е. может использоваться в различных отраслях человеческого знания. Наряду с ним в информатике и ряде других приложений используется понятие «*формальная система*»; оно отличается от общего понятия системы, подобно тому, как понятие математической модели отличалось от понятия модели вообще.

> ***Формальная система*** *— это математическая модель, задающая множество дискретных компонентов путем описания исходных объектов и правил построения новых компонентов из исходных и уже построенных.*

Уточнения к определению:

1. Компонентами формальной системы является *информационные представления* материальных объектов, состояний, отношений и пр. Представления могут быть знаковыми (символическими) или графическими, но обязательно информационными. Таким образом, *формализация* (или построение формальной системы) — *это замена реального прототипа его формальным описанием, т. е. его информационной моделью.*

2. Компоненты формальных систем могут представлять комбинацию конечного числа исходных объектов — неделимых (простейших) элементов с определенным набором свойств. Множество видов таких элементов называется *алфавитом системы.* Число экземпляров элементов каждого вида может быть любым (в том числе бесконечным).

3. Правила построения новых компонентов могут иметь вид «*условие — действие*» («*если имеющиеся объекты или компоненты удовлетворяют некоторым условиям, то для построения нового компонента необходимо выполнить такое-то действие*»). Другим видом правил является «*посылка — заключение*» («*если уже построены компоненты вида* $A_1$, $\ldots, A_{n-1}$, *то компонент* $A_n$ *также считается построенным*»). Новые компоненты называются *выводимыми объектами* (правильнее было бы их называть «*выводимыми компонентами*»).

4. Определение формальной системы во многом напоминают общее (интуитивное) определение алгоритма, данное в главе 8. Это подобие не случайно и будет обсуждаться ниже.

Рассмотрим несколько примеров формальных систем.

**Пример 12.1.**

Множество арифметических формул, которые могут содержать цифровые или буквенные выражения с целочисленными коэффициентами. Алфавит: цифры 0,. . . , 9; буквы a,. . . , z; знаки $+$, $-$, $\times$, $/$; скобки (,). Любой символ может считаться исходной формулой, любая комбинация цифр с первой ненулевой называется числом и считается формулой. Правила построения новых формул следующие:

- если $A$ и $B$ — числа и $A \neq 0$, то $AB$ также число (т. е. частный вид формул — числа — получаются приписыванием одних к другим таким образом, чтобы слева был не 0);
- если $F_1$ и $F_2$ — формулы, то $(F_1 + F_2)$, $(F_1 - F_2)$, $(F_1 \times F_2)$ и $(F_1/F_2)$ также являются формулами.

В описанной формальной системе выводимыми оказываются формулы типа:

$$3215; \quad (z - 15); \quad (((15 \times a) + (1 - c))/(d + 2)).$$

**Пример 12.2.**

Множество возможных шахматных позиций, которые могут быть получены в процессе игры. Алфавит системы — клетки шахматной доски (черные и белые, как свободные, так и занятые той или иной фигурой). Исходное состояние — начальная шахматная позиция. Формальными правилами вывода новых позиций являются правила шахматной игры.

**Пример 12.3.**

Множество высказываний в математической логике. Алфавитом служат буквы, обозначающие переменные, символы логических операций, скобки. Исходными компонентами являются аксиомы; правилами — правила исчисления высказываний. Множеством выводимых объектов оказываются все тождественно истинные высказывания.

**Пример 12.4.**

Машина Тьюринга — это формальная система, порождающая множество своих конфигураций. Исходным объектом является начальная конфигурация; правилами — функциональная таблица. Однако ранее (гл. 9) была доказана эквивалентность представления алгоритмов в различных алгоритмических моделях. Следовательно, если алгоритм в представлении машины Тьюринга является формальной системой, то и алгоритм в любом другом представлении также оказывается формальной системой. Вместе с тем, обратное утверждение («*любая формальная система является алгоритмом*») будет неверным — причина данного обстоятельства прояснится при обсуждении свойств формальных систем.

Приведенные примеры показывают, что формальные системы весьма разнообразны. Однако, как и в случае алгоритмов, можно выделить ряд общих свойств, которыми обладают все формальные системы.

1) *Дискретность.* Это свойство формальных систем проявляется, во-первых, в том, что компоненты системы состоят из неделимых элементов (объектов); во-вторых, в том, что число правил построения новых объектов из имеющихся конечно.

2) *Формальность.* Суть формального подхода состоит в выполнении (соблюдении) ряда принципов:

- *принцип педантизма: все, что делается — делается строго по правилам формальной системы.* Если формальная система не позволяет построить желаемые объекты в рамках имеющихся правил, то вести построение, нарушая правила, нельзя. Можно изменить имеющиеся правила или ввести новые, но это эквивалентно построению другой формальной системы;
- *принцип явного описания: все условия применения правил и действия, которые нужно совершить при их применении, формулируются явно.* В формулировках и описаниях действий не должны использоваться знания и опыт исполнителя или «действия по умолчанию», т. е. что-то не описанное явно, но подразумевающееся. На практике построить такое строгое описание если и удается, то оно оказывается весьма громоздким. Однако вести его строго и последовательно оказывается необязательным: если исполнителем является человек, можно необходимые знания сначала ему передать, а затем на них опираться; если исполнитель — устройство, детальность описания определяется имеющейся системой команд.
- *принцип синтаксичности: условия применения правил формулируются только на основе чисто внешних и четко различимых признаков тех объектов, к которым эти правила применяются* (именно поэтому важна дискретность — различия не должны быть малыми до неразличимости). Иначе говоря, чтобы применять правила, достаточно уметь различать только *внешние* свойства объектов: их элементный состав (символы, фигуры и пр.) и их взаимное расположение. Такие свойства в теории формальных систем называются *синтаксическими.* Поэтому принципу можно дать иную формулировку: *объекты считаются различ-*

*ными, если и только если они различны синтаксически*, т. е. имеют различный внешний вид. При таком подходе отпадает необходимость в понятиях «*существенные различия*» и «*несущественные различия*» — они либо есть — и тогда это разные объекты, либо их нет — тогда объекты идентичны. Отсутствует и понятие внутренних различий, не имеющих явных внешних проявлений «*злой — добрый*», «*полый — сплошной*», «*со сложным внутренним строением — с простым строением*». Именно принцип синтаксичности позволяет осуществлять синтаксический анализа предъявленного текста. Синтаксически правильный текст — это текст, выводимый в данной формальной грамматике, а любой текст, содержащий синтаксические ошибки, не выводим. Такой анализ осуществляется при трансляции компьютерной программы с языка программирования в машинные коды.

3) *Элементарность действий.* Речь идет о том, что хотя правила формальной системы могут иметь самый разнообразный вид, однако любое сложное правило может быть сведено к последовательности простых *комбинаторных* манипуляций с элементами системы, носящих число механический характер. В теории алгоритмов мы уже обсуждали эти манипуляции; в приложении к формальным системам это порождает следующий набор элементарных действий:

- замена пустого элемента (т. е. места, где согласно правилам элемент может быть, но отсутствует) непустым элементом, что эквивалентно появлению элемента рядом с имеющимися;
- замена одного элемента другим (или группой других);
- удаление элемента (т. е. замена его пустым элементом).

Любые действия из системы команд исполнителя в конечном счете могут быть сведены к указанным элементарным, что было показано в теории алгоритмов.

3) *Необязательность детерминизма.* Это свойство состоит в том, что на каждом шаге процесса построения новых объектов в общем случае применимо не одно правило, а несколько, и, следовательно, следующий шаг выбирается из нескольких возможных. В этом и состоит отличие формальных систем от алгоритмов, в которых любому порождаемому объекту применимо только одно правило (это свойство детерминированности алгоритма), что обеспечивает однозначность работы и результата алгоритма. Будет справедливым утверждение, что *формальная система с единственным правилом на каждом шаге функ-*

*ционирования является алгоритмом* и, следовательно, формальная система является более общим понятием, чем алгоритм, а алгоритм можно считать частным случаем формальной системы. В формальной системе на каждом шаге или некоторых шагах оказывается возможным применение различных правил (это видно из рассмотренных выше примеров — шахматной игры или построения формул), что обеспечивает порождение ни одного результата (как в алгоритме), а *множества результатов* (например, множество вариантов игры в шахматы или множество арифметических формул).

Относительно рассмотренных общих свойств формальных систем необходимо сделать еще ряд замечаний.

Формальная система функционирует строго по своим правилам и различает свои объекты только по синтаксическим признакам. С этой точки зрения она однозначно задает некоторую математическую модель. Но при использовании этой модели для решения какой-либо задачи, элементам формальной системы присваиваются конкретные свойства и признаки, т. е. происходит *интерпретация* системы. Подобный механизм широко применяется в программировании: разрабатывается процедура, обеспечивающая выполнение некоторой последовательности действий по обработке данных с формальными параметрами (т. е. задается схема, последовательность обработки), а затем при вызове процедуры вместо формальных подставляются фактические значения. Одна и та же формальная система может иметь различные интерпретации, зависящие от предметной области, в которой используется модель, либо от целей, для которых она создавалась. Например, для формальной системы, описывающей законы математической логики, имеется иная интерпретация — правила функционирования логических схем, о чем шла речь ранее.

Интерпретация является внешней процедурой по отношению к формальной системе и в рамках самой системы не обсуждается. Уровень интерпретации — его называют *метауровнем* — может быть как формализованным, так и неформальным. На метауровне система становится уже не средством решения задачи, а объектом изучения. Например, формальная система, описывающая правила игры в шахматы, позволяет отличить правильные шахматные ходы от неправильных, но в ее рамках невозможно отличить хорошие ходы от плохих. Вопрос об оценке качества хода решается не в системе, а на метауровне. Причем возможно неформальное решение — на основе опыта и мастерства игрока,

а также шахматной теории (которая является обобщением опыта многих шахматистов); возможна и формальная интерпретация оценки качества хода, которую создают и затем закладывают в шахматные компьютерные программы их разработчики. Точно также при компиляции компьютерных программ осуществляется их синтаксический контроль, т. е. проверка соответствия правилам формальной грамматики, однако компилятор не может отличить хорошую (в смысле эффективности исполнения) программу от плохой.

### 12.2.4. Значение формализации

Из рассмотренных выше свойств формальных систем наиболее важным, безусловно, следует считать формальность. Даже в нашей повседневной жизни многое организуются по принципам формальных систем. *Формализм — это порядок*; поэтому там, где имеется множество однотипных объектов (на складе, в библиотеке, в компьютерной базе данных), субъектов (работник предприятия, солдат, студент) или ситуаций (правонарушения, дорожное движение, начисление налогов, обучение в вузе), с которыми имеют дело различные люди, совершенно необходимы четкие правила и их одинаковое понимание всеми, кто ими пользуется. Только формальное следование установленным законам, кодексам, положениям обеспечивает порядок в государстве как системе в целом и отдельных его подсистемах (учреждениях, сообществах, организациях и пр.). Примерами могут служить действия пилотов, машинистов, военных, судей, рабочих на конвейере и т. п. И наоборот, чем меньше формализма в какой-либо деятельности, тем сложнее и неоднозначнее оценка ее результатов, что видно по работе государственных деятелей, учителей, писателей и др.

Формальный подход принципиально важен для точных наук. Для того чтобы знание одного человека стало доступным и однозначно понимаемым для всех, оно должно быть изложено с помощью точных понятий и терминов. По этой причине язык науки близок к формальному языку (в части определения понятий). Предельным и наиболее однозначным для понимания является язык математики и логики — именно этим объясняется расширение сферы применения математических методов по мере развития науки.

В настоящее время теоретическая или прикладная проблема считается поставленной точно и однозначно, если она формали-

зована, т. е. может быть представлена в виде формальной системы. Любые конечные состояния выводятся из исходных объектов посредством действий, представляющих совокупность правил формальной системы. При этом если соблюдается принцип синтаксичности, во-первых, может быть осуществлен синтаксический контроль правильности описания действий и, во-вторых, сами действия можно выполнять «не думая» и, в частности, посредством технического устройства. Именно поэтому для компьютеризации любой области знаний или деятельности необходима ее полная формализация.

## 12.3. Этапы решения задачи посредством компьютера

Переведем проблему в практическую плоскость и обсудим, можно ли указать некую общую для различных задач последовательность действий, если для их решение предполагается применение компьютера?

Ранее (см. п. 10.2 и рис. 10.1) нами были выделены две группы форм представления алгоритмов, различающихся уровнем формализации: естественное и формальное. В группу естественного представления были отнесены некоторые виды строчной записи и графическая форма, а в группу формального — теоретические алгоритмические модели и формальные языковые конструкции. На основании этой классификации, а также понятий, обсуждавшихся в данной главе, можно построить следующий *общий порядок* решения задачи с использованием компьютера:

1) выделение исходной системы. Постановка задачи в естественной форме;

2) формализация задачи; построение информационной модели (проекта);

3) доказательство возможности решения задачи;

4) структурирование алгоритма; представление его в естественной форме;

5) выбор инструментальной среды;

6) построение схемы обработки данных;

7) формальное представление схемы обработки на языке исполнителя;

8) тестирование модели, исправление ошибок;

9) эксплуатация модели.

Относительно приведенной последовательности необходимо сделать ряд замечаний:

**Этап 1** — *постановка задачи* — является неформализованным. Его начало — это замена прототипа, поведение которого предполагается изучать или описывать, некоторой системой, что само по себе является упрощением реальной ситуации, т. е. моделированием. Другими словами, при компьютерном решении задач моделирование начинается не с построения информационной модели на этапе 2, а с определения перечня тех компонентов, которые будут включены в систему и взаимодействие которых предстоит учесть. Любое подобное построение приводит к появлению различий между прототипом и представляющей его системой. Эти различия повлекли бы расхождения в результатах компьютерной обработки и естественными явлениями (процессами) даже в том случае, если все последующие этапы были бы выполнены абсолютно точно. Таким образом, «*исходная система*» — это модель; однако на данном этапе она может носить неформальный характер, например быть описанной на естественном языке.

**Этап 2** — *формализация задачи* — это по сути построение формальной системы, соответствующей исходной неформальной и представляющей собой ее информационную модель. Частным (распространенным, хотя не единственно возможным) вариантом описания такой модели является математическое. Из сказанного становится понятна важность формализации как этапа решения задачи на компьютере: именно он определяет принципиальную разрешимость задачи (существование алгоритма ее решения), эффективность использования ресурсов компьютера (объема памяти и времени работы центрального процессора), точность моделирования. Наряду с этим возможна и довольно широко распространена ситуация, когда компьютер используется в качестве инструментального средства для создания искусственной системы — документа, рисунка, базы данных и т. п. В этом случае говорить о модели не приходится — отсутствует прототип; однако все равно должен быть построен (с явным представлением или в воображении пользователя) формализованный проект конечного продукта.

**Этап 3** включает две позиции: во-первых, это доказательство принципиальной алгоритмической разрешимости задачи (см. п. 9.7); а во-вторых, оценка сложности алгоритма (см. п. 9.6) и доказательство того, что имеющихся технических ресурсов достаточно для решения задачи требуемого уровня сложности.

После того как произведена формализация и доказана разрешимость задачи, наступает этап *конкретизации алгоритма*: он

структурируется, т. е. общая задача разбивается на модули, определяется порядок доступа к модулям, их взаимосвязи. Описание структуры может быть неформализованным — на естественном языке или с применением графических форм.

На **этапе 5** выбивается *оптимальный* для решения данной задачи *инструмент*. Критериями оптимальности могут быть простота получения результата, удобство интерфейса пользователя, точность и скорость обработки и пр. Выбор осуществляется в два шага: на первом решается вопрос, будет ли для решения задачи использоваться существующая программная система (назовем такой подход *пакетным*) или программа будет разрабатываться (*программный* подход). Разнообразие и удобство современного стандартного программного обеспечения компьютера столь велико, что позволяет решать многие весьма сложные задачи моделирования без разработки программ. Это заметно расширяет круг пользователей, избавляет от необходимости организации программного интерфейса, позволяет многие задачи решить быстрее, нежели путем программирования. Именно по этим причинам при выборе инструментальной среды пакетному подходу следует отдавать предпочтение по сравнению с программным; усилия по разработке оригинальной программы можно признать целесообразными только в тех случаях, когда отсутствуют стандартные аналоги или они не позволяют решить задачу. Второй шаг инструментального выбора при пакетном подходе состоит в определении наиболее подходящей для данной задачи прикладной программы. Например, для математического моделирования могут сопоставляться пакеты MS Excel, MathCAD, MathLab, Matematica, Derive, Maple V и др. При программном подходе на втором шаге осуществляется выбор языка программирования, наиболее отвечающего характеру данной задачи. В этом случае критериями выбора может служить удобство описания исходной модели в данном языке, эффективность технологии программирования и, безусловно, эффективность конечного программного кода.

**Этап 6** связан с выбором или построением последовательностей алгоритмической обработки данных в модулях основного алгоритма. Весьма часто одно и то же формальное представление допускает различные варианты непосредственной обработки. Например, задача о движении материальной точки под действием произвольной силы описывается дифференциальным уравнением второго порядка, для решения которого могут применяться раз-

личные численные методы: Эйлера, касательных, хорд, Рунге–Кутта и др. Одним из факторов, определяющих выбор метода, является точность исходного моделирования. Довольно очевидным представляется утверждение: обработка данных не может повысить их точность. Речь идет о том, что погрешность, возникающая в ходе начальных допущений на этапе формализации, не может быть компенсирована последующей сколь угодно тщательной и точной обработкой данных. Например, в модели, описывающей движение брошенного тела без учета сопротивления воздуха, принципиально невозможно получить верное значение дальности полета, сколь бы точно не велся расчет. Аналогично точность вычислений не может компенсировать или уменьшить относительной погрешности результатов измерений, используемых в качестве исходных данных задачи. По указанным причинам при выборе порядка обработки данных нет смысла всегда стремиться к использованию схемы максимально точной из всех возможных — как правило, они заметно повышают сложность алгоритма и, следовательно, требуют гораздо больших технических ресурсов или времени обработки. Для расчетных алгоритмов оптимальной с точки зрения эффективного использования компьютера будет выбор вычислительной схемы, в которой точность расчетов (промежуточных данных) лишь на 1–2 порядка превышает точность исходных данных (этого требует теория приближенных вычислений). Ограничение точности исходной информации необходимо учитывать и при описании типов данных в программах. Например, при программировании в PASCAL нет смысла применять тип Double (15 десятичных разрядов) или даже Real (12 разрядов) для представления вещественных чисел, определенных с точностью 0,1 % (относительная погрешность $10^{-3}$) — вполне достаточным окажется тип Single (7 разрядов), занимающий в ОЗУ в два раза меньше места, чем Double.

Схема обработки определяется также выбранным на предыдущем этапе инструментальным средством. Например, при моделировании с применением MS Excel (как и в любом другом пакете) мы ориентируемся на его возможности и его внутренний язык представления команд (указаний по обработке); по этой причине схема обработки будет иной, нежели, скажем, в пакете MathCAD или при решении задачи с использованием языков программирования.

Смысл **этапа 7** совершенно очевиден — представление алгоритмической схемы обработки в виде последовательности до-

пустимых команд исполнителя. В пакетном варианте это может быть просто цепочка действий, осуществляемых пользователем, или их запись на внутреннем языке программной системы (макрос). В программном подходе — запись алгоритма с помощью выбранного языка программирования.

**Этап 8** состоит в локализации и исправлении возможных ошибок. Выделяются несколько источников и, соответственно, типов ошибок. Наиболее простыми и легко устранимыми являются *ошибки синтаксиса*, т. е. нарушение принятых для данного исполнителя правил построения предписаний действий. Как правило, синтаксический анализ проводится самой программной системой или транслятором языка программирования — она информирует пользователя об ошибках и не позволяет проводить дальнейшую обработку данных при их наличии. Второй класс ошибок — *смысловые*, которые проявляются в том, что при определенных комбинациях входных данных их обработка проводится некорректно. Подобные ошибки локализуются тестированием, т. е. применением построенной схемы решения для задач (ситуаций) с известными результатами и сопоставление с ними. Необходимо напомнить (см. п. 9.6), что тестирование, в общем случае, не обеспечивает устранение всех возможных ошибок. К следующему классу следует отнести *ошибки выбора схемы обработки* (на этапе 6). Они проявляются в том, что схема не обеспечивает нужной точности результатов или скорости их получения — в этом случае требуется замена схемы обработки или исполнителя. Наконец, наиболее трудоемкими для устранения являются *ошибки моделирования*. Они могут возникать на начальных этапах 1 и 2 и связаны с грубостью выделения системы или построения ее модели (т. е. пренебрежением какими-то существенными для задачи компонентами и связями). Устранение подобной ошибки состоит в смене модели и, как следствие, необходимости решать задачу полностью заново.

**Этап 9** — *эксплуатация модели* — является основным с точки зрения конечного пользователя. В подавляющем большинстве случаев разработка компьютерной модели, обеспечивающей решение какой-то прикладной задачи, и ее практическая эксплуатация осуществляется разными людьми. На этом этапе существенным оказывается наличие обратной связи между пользователем и разработчиком, что позволяет устранять выявленные в процессе эксплуатации ошибки и совершенствовать модель.

В указанной последовательности решения задачи обращает

на себя внимание некоторая цикличность этапов, имеющих естественное и формальное представление: на этапе 1 описание задачи может осуществляться на естественном языке; этап 2 предполагает построения формализованной информационной модели; на этапе 3 также используются формальные методы доказательства разрешимости; этапы 4, 6 представляет собой структуризацию алгоритма и представление отдельных его фрагментов его в естественной форме (с учетом системы команд выбранного исполнителя); этап 7 — вновь формальное описание (на языке программирования). Таким образом, решение задачи на компьютере требует умения пользоваться как формальными, так и естественными способами представления алгоритмов.

Нами рассмотрена общая последовательность действий при решении прикладной задачи с использованием компьютерных средств. Безусловно, отдельные этапы последовательности в каких-то конкретных задачах могут отсутствовать или представляться в ином виде.

## 12.4. Об объектном подходе в прикладной информатике

Имеется определенная специфика использования введенных выше терминов «*объект*» и «*система*» при решении практических задач компьютерными методами, вызванная, в первую очередь тем, что все рассматриваемые сущности могут быть только информационными, т. е. нематериальными и они не существуют сами по себе, а появляются лишь в результате исполнения компьютером последовательности действий, представленной ему в виде программы. Начнем обсуждение этой специфики с термина «*система*» — обращение к нему при решении задач с применением компьютерной техники происходит постоянно. Согласно приведенному в п. 12.2.2 определению системой является сам компьютер и отдельные его устройства, что отражено даже в их названиях: системный блок, система ввода-вывода, система мультимедиа и пр. В программном обеспечении выделяются операционные системы, системы программирования, прикладные программные системы. Наконец, под определение системы подпадают и результаты применения некоторых программ — документы. Нас будут интересовать именно последние классы систем — программы и документы. Уточним понятия:

> ***Программа*** *— последовательность действий по обработке информации исполнителем «компьютер».*

Программа перед исполнением обязательно представляется на языке машинных кодов команд процессора, может запускаться к исполнению непосредственно из операционной системы и далее функционировать, вообще говоря, независимо от других программ или документов. В операционных системах семейства Windows программы получили название «*приложения*».

> ***Документ*** *— продукт, сформированный в результате исполнения некоторой программы.*

Особенности, отличающие документ от программы, состоят в следующем:

- документ не существует независимо; он доступен только из породившей его программы (или других программ, использующих тот же формат представления данных);
- для документа непосредственным исполнителем является не компьютер, а порождающая его программа; язык документа, т. е. способ представления его данных и команд также определяется программой.

Различия между программой и документом исчезают при их записи и хранении на носителе, например жестком диске компьютера, — все представляется в виде двоичных файлов. В некоторых случаях возможно преобразование документа в программу, например при компиляции текста, написанного на языке программирования, в машинные коды. Существует и операция преобразования машинной программы в текст на языке ассемблера — *дизассемблирование*. Однако в подавляющем большинстве ситуаций программы и документы четко различаются по перечисленным выше признакам.

Теперь отметим общие для документов и программ моменты:

- они являются системами, т. е. состоят из взаимосвязанных составных частей и обладают свойствами единства и системности;
- по природе это информационные системы;
- их представлением (отображением) является некоторая *конфигурация экрана.*

> ***Конфигурация экрана*** *— это все, что можно создать и наблюдать на экране компьютера.*

Под этот обобщающий термин подпадают и экранный документ, и игровая ситуация, и программная оболочка, управляющая конфигурацией. Современный подход к организации программ, порождающих экранные конфигурации, состоит в том, что

они должны предоставлять пользователю некоторый набор управляемых элементов, комбинируя которые, можно нужную конфигурацию построить. Этот подход получил название *объектного,* а сами элементы — называние *экранных объектов*. Примерами экранных объектов являются кнопки, фрагменты документов (тексты, рисунки, диаграммы, клипы и пр.), окна, меню и т. д. Экранный объект обладает всеми признаками, указанными в определении объекта вообще (см. п. 12.2.1):

- его строение (фактическая программная реализация) недоступно пользователю — по этой причине он рассматривается как неделимое целое;
- он обладает определенным (заданным) набором свойств, часть из которых могут быть изменены пользователем;
- он имеет имя в обслуживающей программной системе, по которому осуществляется доступ к нему.

Наряду с этим, экранные объекты обладают рядом специфических особенностей:

- по своей природе они являются информационными (т. е. нематериальными), хотя могут иметь свои материальные представления (например, при выводе образов экранных объектов на принтер);
- характеристики их свойств дискретны; по типам величин это целые числа, символьные или логические данные;
- любые воздействия, в результате которых происходит изменение свойств объекта, имеют дискретное представление*; каждое такое воздействие называется *событием*, а изменение свойств — *реакцией на событие*; перечень событий определяется возможностями внешних устройств ввода информации (чаще всего — это клавиатура и манипулятор «мышь»);
- все объекты размещаются в плоскости экрана — по этой причине управляющая программная оболочка обязана обеспечивать разрешение конфликтов при попадании нескольких объектов в одну область экрана; вариантами разрешения могут быть *вытеснение* (объект занимает данную область, заставляя конкурирующий объект переместиться в другое место)

---

* Поскольку любая непрерывная во времени функция допускает дискретизацию без потери точности описания (см. п. 1.3), постоянное воздействие (например, перетаскивание объекта по экрану мышью) всегда можно представить как последовательность дискретных событий.

или *перекрытие* (объект визуально размещается поверх конкурирующего или за ним).

Объектно-ориентированная прикладная программа предоставляет пользователю набор готовых объектов, инструментальные средства для создания новых объектов и средства управления, благодаря которым он может изменять свойства объектов и строить желаемую (конечно, в рамках предусмотренного) конфигурацию экрана. Таким образом, при объектном подходе документы или иные конфигурации *конструируются* (собираются) из экранных объектов. Ситуация напоминает детский конструктор, позволяющий создавать различные устройства из имеющегося в распоряжении набора деталей. При этом человеку, производящему такую сборку, можно не знать строение отдельной детали, а только представлять ее свойства и способы соединения с другими деталями. Такой подход к построению и использованию чего-либо получил название «*принцип черного ящика*».

> ***Черный ящик** — это система, строение которой неизвестно пользователю, однако известна ее реакция на определенные внешние воздействия.*

Совокупность всех средств воздействия образуют *интерфейс* черного ящика. Человеку, который желает применить черный ящик в качестве средства для решения некоторой практической задачи, вообще говоря, не требуется знать, как ящик устроен и функционирует, а достаточно овладеть его интерфейсом. Такие люди называются *пользователями*. Круг пользователей, безусловно, намного шире круга специалистов, способных создать устройство или при необходимости восстановить его работоспособность. С подобными черными ящиками мы постоянно имеем дело в повседневной жизни — телефон, телевизор, автомобиль, компьютер и т. д. Любая компьютерная программа, которой пользуется не сам разработчик, также оказывается черным ящиком — именно по этой причине, чем нагляднее и дружественнее ее интерфейс, тем шире круг пользователей.

Важными в объектном подходе для прикладных программ оказываются следующие обстоятельства:

- из всевозможных объектов выделяется некоторое подмножество объектов *инвариантных* — тех, что имеются в различных программах; это позволяет унифицировать их интерфейс (примером могут служить кнопки, окна, фрагменты текста, клипы и пр., используемые в документах пакета MS Office);

- освоение каких-либо объектных программ для их применения по сути сводится к освоению интерфейса объектов; общий же принцип — конструирование документа из объектов — одинаков для всех программ;
- для начала работы с программой не требуется ее полного изучения — достаточно освоить некоторые минимально необходимые возможности (подобно тому, как для начала изъяснения на иностранном языке можно знать небольшую часть его лексики и грамматики).

Перечисленные обстоятельства заметно облегчают освоение и практическое использование объектно-ориентированных прикладных программ, а также повышают производительность при подготовке в них документов.

Объектный подход принят в настоящее время не только в прикладных программных системах. Объектно-ориентированное программирование является одной из наиболее прогрессивных современных парадигм программирования и технологий разработки программ. Программа строится путем комбинирования *программных объектов*.

> ***Программный объект*** *есть совокупность некоторого набора данных и процедур, определяющих возможности их изменения.*

Программный объект объединяет определенную структуру данных и доступные только ему механизмы изменения данных, т. е. в объектно-ориентированном программировании ликвидируется противопоставление и неравноправность между процедурами и данными. Объединение данных и процедур в объекте называется *инкапсуляцией*. Принципиальное отличие программных объектов от объектов в документах состоит в возможности формирования новых объектов из уже имеющихся *наследованием свойств* — так образуются *классы объектов*.

Известным примером объектно-ориентированного языка является C++, а также его версии Turbo C++ и Borland C++. В идеологии объектного программирования построены и среды визуального программирования такие, как DELPHI, Visual BASIC, JAWA.

Таким образом, объектный подход объединяет и программистскую ветвь прикладной информатики, и пользовательскую. Единство подходов к построению документа или программы весьма важно с идейной точки зрения, поскольку демонстрирует

общность и универсальность методов информатики, с одной стороны, и заметно облегчает освоение компьютерных приложений, с другой стороны.

## Контрольные вопросы и задания к гл. 12

1. Являются ли моделями:

a) фоторобот преступника;

b) корреспонденции журналистов;

c) схема компьютера;

d) компьютерное изображение разрабатываемого автомобиля?

2. Приведите примеры множественности моделей для одного прототипа.

3. Как соотносятся понятия «*модель*», «*макет*», «*схема*»?

4. Постройте граф математической модели и охарактеризуйте отношения:

a) для описания: «*AA и BB являются родителями для C и D*»;

b) четные числа;

c) функция $y = x^2$;

d) $a \geqslant b$.

5. Приведите примеры, подобные рассмотренному в п. 10.2.1, когда некоторая сущность в одних задачах может считаться объектом, а в других — нет.

6. Приведите примеры классов объектов с указанием общих и индивидуальных свойств.

7. Имеет ли смысл сочетание «*модель объекта*»? Ответ обоснуйте.

8. Имеет ли смысл сочетание «*сложная система*»? Ответ обоснуйте.

9. Для нескольких систем выделите их компоненты с разнесением на объекты и подсистемы.

10. Являются ли системами:

a) природа в целом;

b) компьютер;

c) компьютерная программа;

d) учебник по информатике?

11. Поясните соотношение понятий «*модель*» и «*система*».

12. Являются ли формальными системами:

a) уголовный кодекс;

b) детский конструктор;

c) правила правописания;

d) текстовый редактор Word?

13. Опишите формальную систему игры в «*крестики-нолики*». Постройте программную реализацию на каком-либо языке программирования для полей 3×3 и 4×4 клетки.

14. Постройте конкретизацию рассмотренной в п. 10.3 последовательности решения задачи на компьютере в следующих ситуациях:

a) создание документа с помощью текстового редактора;

b) разработка программы, описывающей взаимодействие и движение нескольких тел;

c) создание базы данных для отдела кадров предприятия;

d) разработка игровой программы.

15. В чем суть объектного подхода в прикладной информатике? Каковы возможные альтернативные подходы?

# Заключение

Вернемся к определению предмета науки информатики, которое рассматривалось во введении. Его сопоставление с кругом вопросов, обсуждавшихся, в частности, в данном учебнике, позволяет увидеть колоссальное значение, которое имеет теоретическая информатика для информатики в целом. В теоретической информатике сформулирован ряд положений, которые следует отнести к категории *фундаментальных*, поскольку они оказываются справедливы и проявляются в любых информационных процессах и, следовательно, только на основе этих положений могут разрабатываться какие-либо информационные технические устройства, системы или методы переработки информации. К этим положениям следует отнести:

- представление об энтропии как мере недостатка информации и информации как сведениях, уменьшающих неопределенность;
- способ объективного измерения количества информации;
- законы передачи информации по каналу связи, в том числе подверженному воздействию шумов;
- методы эффективного кодирования информации, обеспечивающие сколь угодно высокую надежность передачи информации;
- криптографические методы;
- принципы преобразования дискретной информации и условия, при которых такая обработка может быть алгоритмизирована и, следовательно, передана для исполнения техническому устройству;
- методы доказательства алгоритмической разрешимости и методы определения сложности алгоритма;
- структура, элементная база и теоретические основы функционирования устройств по автоматической обработке информации;

- системный подход к исследованию и описанию сложных объединений и объектный подход к их созданию;
- значение и методы формализации;
- общий порядок (последовательность) решения прикладной задачи, осуществляемого с применением информационных технических устройств.

Однако решение перечисленных проблем — их можно назвать *проблемами первого эшелона* — породило проблемы новые и, соответственно, необходимость их теоретического осмысления. К ним можно отнести:

- необходимость формализации задачи на этапе ее постановки потребовало разработки теории систем и моделей;
- обобщение понятия алгоритма на любую целесообразную деятельность человека и технического устройства, в частности управление, потребовало создание теории управления — науку кибернетику;
- создание новых операционных систем, систем программирования и прикладных программ потребовало развития методов построения формальных языков;
- разработка информационных систем, решение задач искусственного интеллекта породило необходимость развития моделей данных, логических средств преобразования данных, моделирования рассуждений и т. п.

Безусловно, перечень примеров можно было бы продолжить.

Таким образом, изучение теоретической информатики необходимо рассматривать в качестве начального и обязательного этапа, который должен предшествовать углубленному освоению любых других разделов информатики. Автор выражает скромную надежду, что данный учебник будет способствовать решению этой задачи.

# Приложение А.
# Элементы теории вероятностей

Рассматривается лишь те положения теории вероятностей, которые используются в изложении основного материала, в частности в гл. 2.

## А.1. Понятие вероятности

В естественных науках изучаются явления, исход которых определяется однозначными причинно-следственными связями, выраженными с помощью математического понятий аргумент-функция. Например, электросопротивление цепи и напряжение на ее концах однозначно задает силу тока, что является сутью закона Ома. Причем сколько раз мы не повторили бы опыт, результат измерения силы тока окажется одинаковым (в пределах погрешности измерений).

Однако часто приходится иметь дело с явлениями, исход которых неоднозначен и зависит от факторов, которые мы не знаем или не можем учесть. Простейший и традиционный пример — предсказание результата бросания монеты, т. е. выпадение «орла» или «решки». Подобной же оказывается ситуация с выигрышем по лотерейному билету, получением определенной карты в карточных играх, попадание в цель при стрельбе, результат спортивной встречи, количество пассажиров в автобусе и пр.

> *Явления, исход которых не может быть однозначно определен до того, как они произошли, называются* ***случайными****.*

Раздел математики, в котором строится понятийный и математический аппарат для описания случайных событий, называется *теорией вероятностей*. Количественное описание случайных событий опирается на то, что при многократном повторении явления с неоднозначным исходом в одних и тех же условиях *частота* появления некоторого результата остается приблизительно

одинаковой. Будем называть отдельный повтор случайного явления *опытом*, а интересующий нас исход этого явления — *благоприятным событием* (или благоприятным исходом). Тогда, если $N$ — общее число опытов, а $N_A$ — количество благоприятных исходов случайного события $A$, то отношение $N_A/N$ будет показывать долю благоприятных исходов в проведенной серии опытов или *относительную частоту* появления благоприятного исхода. Однако в разных сериях при небольшом количестве опытов в каждой значение частоты может оказаться различной. Например, в серии из 3 опытов по бросанию монеты у нас 2 раза выпал орел и 1 решка. Если благоприятным исход считать выпадение орла, то частота получается равной $2/3$. В другой же серии из трех опытов результат может оказаться совершенно иным, например все 3 раза выпадет решка, и, следовательно, частота появления орла оказывается равной 0. Частота стремится к некоторой определенной (и постоянной) величине только в том случае, если количество опытов будет велико, в предельном случае стремится к бесконечности. Это величина и называется *вероятностью случайного события* $A$:

$$p(A) = \lim_{N\to\infty} \frac{N_A}{N}. \qquad \text{(A.1)}$$

Данное определение вероятности называется *частотным*; оно применимо для возможных исходов, образующих *дискретный* (конечный) набор. Существуют и случайный события, имеющие непрерывный ряд возможных исходов, например значение скорости молекулы газа или время ее пребывания в некоторой области пространства; для таких событий используется иное определение вероятности. Мы будем иметь дело только с дискретными событиями и пользоваться приведенным выше определением.

Безусловно, различные события имеют разную вероятность. Можно считать, что значение вероятности характеризует событие и в то же время является его функцией; по этой причине будем придерживаться обозначения $p(A)$.

Альтернативой случайному событию является событие, относительно которого мы точно знаем, что оно произойдет (например, наступление дня после ночи), — такие события будем называть *достоверными*. Достоверное событие можно рассматривать как предельный случай события случайного — для него в любом опыте $N_A = N$ и согласно (A.1) $p(A) = 1$. Наоборот, события, которые никогда не могут произойти, — будем называть их *невероятными* (например, вынуть красный шар из урны с

белыми и черными) — имеют всегда $N_A = 0$ и, следовательно, $p(A) = 0$. Таким образом, случайные события располагаются между невероятными и достоверными, а для их частотной характеристики — вероятности — очевидно, будет справедливо соотношение

$$0 \leqslant p(A) \leqslant 1. \tag{A.2}$$

Полученное выражение называется *условием нормировки* вероятностей (в дальнейшем мы получим более общую форму его записи).

Безусловно, важной задачей является определение (или оценка) вероятности некоторого случайного события. Однако произвести практически бесконечное число опытов, что требует определение вероятности, конечно, невозможно. Поэтому приходится привлекать некоторые иные соображения. Рассмотрим ситуацию, когда случайное событие имеет несколько независимых исходов, но все они *равновероятны*, т. е. относительные частоты их наступления одинаковы.

Пусть $n$ — общее число равновероятных событий, которые обозначим $A_1, A_2, \ldots, A_n$; вероятности их наступления будут, соответственно, $p(A_1), p(A_2), \ldots, p(A_n)$. Рассмотрим сложное событие $A$, для которого благоприятным окажется любой из исходов $A_1, \ldots, A_n$; очевидно, такое событие будет достоверным (так как какое-то из перечисленных событий все равно произойдет) и, следовательно, вероятность сложного события $p(A) = 1$. С другой стороны, поскольку отдельные события независимы, т. е. наступление любого из них никак не связано и не обусловлено другими, каждое из них внесет свою лепту величиной $p(A_i)$ в вероятность сложного события. Таким образом:

$$p(A) = p(A_1) + p(A_2) + \ldots + p(A_n) = 1.$$

Так как рассматриваются равновероятные события, очевидно

$$p(A_1) = p(A_2) = \ldots = p(A_n) = p.$$

Следовательно, $p(A) = np = 1$, откуда получаем, что вероятность любого из равновероятных событий

$$p = \frac{1}{n}. \tag{A.3}$$

**Пример А.1.**

Бросание монеты: $n = 2$, $p = 1/2$.

Бросание игральной кости (кубика): $n = 6$, $p = 1/6$.

Получение дамы пик из колоды 36 карт: $n = 36$, $p = 1/36$.

Конечно, для применения этой простой формулы необходимо доказать, что исходы равновероятны, — такое доказательство выходит за рамки теории вероятности, однако является условием применимости ее соотношений. Часто для доказательства прибегают к соображениям симметрии или исходят из неких интуитивно ясных (но недоказуемых!) посылок. Например, принимается, что при условии однородности материала игрального кубика и правильности его геометрической формы равновероятно выпадение любой из граней.

Формулу (А.2) легко обобщить на ситуацию, когда благоприятное событие осуществляется несколькими способами $m$ из $n$ равновероятных (очевидно, $m \leqslant n$). Например, выпадение четной цифры при броске кубика ($n = 6$, $m = 3$). Тогда

$$p = \frac{m}{n}. \tag{А.4}$$

**Пример А.2.**

Какова вероятность вытащить из колоды 36 карт туза?

$n = 36$, $m = 4$, $p = 1/9$.

Какова вероятность достать красный шар из ящика с 10 шарами, из которых 5 красных, 3 зеленых и 2 черных?

$n = 10$, $m = 5$, $p = 1/2$.

## А.2. Сложение и умножение вероятностей

**Сложение вероятностей независимых несовместных событий.** Пусть среди возможных равновероятных исходов благоприятными для нас оказывается *любое* из событий $A$ или $B$, причем $A$ и $B$ — *независимы* (т. е. между ними отсутствуют причинно-следственные связи — $A$ не влияет на $B$ и наоборот) и *несовместны* (т. е. они не могут наступить одновременно, а наступает лишь одно из них). Путь из общего числа равновероятных исходов $n$ событие $A$ осуществляется $m_1$ числом способов, а событие $B$ — $m_2$. Тогда

$$p(A) = \frac{m_1}{n}; \quad p(B) = \frac{m_2}{n}.$$

Спрашивается, как найти вероятность события $C = (A$ или $B)$*? $C$ называют также *суммой событий* $A$ и $B$. Очевидно,

---

* Используя формализм математической логики, можно записать $C = A \vee B$.

общее число способов, которым может реализоваться $C$, равно $m_1 + m_2$. Следовательно,

$$p(A \vee B) = \frac{m_1 + m_2}{n} = \frac{m_1}{n} + \frac{m_2}{n} = p(A) + p(B).$$

**Вероятность какого-либо одного из двух благоприятных исходов независимых и несовместных событий равна сумме их вероятностей:**

$$p(A \vee B) = p(A) + p(B). \tag{A.5}$$

Например, какова вероятность вынуть красный или зеленый шар в рассмотренном выше примере? $p(A) = 5/10$, $p(B) = 3/10$, следовательно, $p(A \vee B) = 8/10$.

Формулу (A.5) легко обобщить на произвольное число благоприятных исходов. Пусть из $n$ несовместных событий $k$ являются благоприятными исходами, причем вероятности каждого составляют $p_1, p_2, \ldots, p_k$. Тогда вероятность наступления любого (хотя бы одного) из благоприятных событий будет равна

$$p = \sum_{j=1}^{k} p_j. \tag{A.6}$$

Из соотношения (A.6) и условия нормировки вероятностей (A.2) легко вывести два следствия:

**Следствие 1**. Если общее количество *всех* возможных исходов равно $n$ с вероятностями $p_1, \ldots, p_n$, то

$$\sum_{i=1}^{n} p_i = 1, \tag{A.7}$$

поскольку появление хоть какого-то из исходов достоверно. Это выражение можно считать *обобщением условия нормировки вероятностей*.

**Следствие 2**. Если возможных исходов всего два ($A$ или $B$), то наступление одного означает не наступление второго. Поэтому любой из них является *противоположным* другому (записывается $B = \overline{A}$ «$B$ есть не $A$»). Поскольку $p(A) + p(B) = 1$, то $p(B) = 1 - p(A)$ или

$$p(\bar{A}) = 1 - p(A). \tag{A.8}$$

Например, если мы определили ранее (пример A.2), что вероятность вытащить красный или зеленый шар равна $8/10$, то это означает, что вероятность не вытащить любой из них равна $1 - 8/10 = 2/10$.

**Умножение вероятностей независимых совместных событий.** Пусть событие $A$ реализуется $m_1$ способами из $n_1$ равновероятных исходов одного опыта, а событие $B$ — $m_2$ способами из $n_2$ равновероятных исходов другого опыта, независимого от первого. Спрашивается, какова вероятность одновременного наступления обоих событий (или сложного события $C$, состоящего в наступлении и $A$ и $B$)? То есть мы хотим определить вероятность *совместного* события $C = (A \text{ и } B)^*$. Часто $C$ называют *произведением событий* $A$ и $B$. Поскольку каждому из $n_1$ исходов первого опыта соответствует $n_2$ исходов второго, то общее количество возможных равновероятных исходов, очевидно, становится равным $n_1 n_2$. Из них благоприятными окажутся $m_1 m_2$ исходов. Следовательно, вероятность совместного события оказывается равной:

$$p(A \wedge B) = \frac{m_1 m_2}{n_1 n_2} = \frac{m_1}{n_1}\frac{m_2}{n_2} = p(A)p(B).$$

Вероятность одновременного наступления двух благоприятных исходов независимых событий равна произведению их вероятностей:

$$p(A \wedge B) = p(A)p(B). \qquad (A.9)$$

**Пример А.3.**

Какова вероятность выбросить 2 раза подряд по 6 очков в двух бросках кубика (или, что эквивалентно, при однократном броске двух кубиков)?

Поскольку $p(A) = p(B) = 1/6$, то $p = 1/6 \cdot 1/6 = 1/36$.

Как и в случае сложения вероятностей, формула (А.9) может быть обобщена на произвольное число совместных благоприятных событий в $k$ независимых опытах. Если вероятности их наступления в каждом из опытов $p_1, p_2, \ldots, p_k$, то вероятность совместного события

$$p = \prod_{j=1}^{k} p_j. \qquad (A.10)$$

*Следствие 1.* Поскольку $p_j \leqslant 1$, очевидно, что $p \leqslant p_j$, т. е. *вероятность совместного события не может превышать вероятность любого из них.*

---

* Используя формализм математической логики, можно записать $C = A \wedge B$.

*Следствие 2.* Нахождение среднего для значений случайных независимых величин. Рассмотрим частную ситуацию, когда случайным событием является числовое значение некоторой величины. Например, число, указанное на грани кубика; сумма выигрыша в лотерею; номер этажа, на котором живет человек, масса атома химического элемента и т. п. Пусть этих значений $n$ и они образуют дискретный ряд $x_1, x_2, \ldots, x_n$. Среди этих значений могут оказаться одинаковые. Пусть таких групп одинаковых значений $k$. Очевидно, $k \leqslant n$. Попробуем найти ответ на вопрос: каково *среднее значение* величины $x$? Например, сколько в среднем очков выпадает при одном броске кубика? Будем исходить из определения среднего значения:

$$\langle x \rangle = \frac{1}{n}(x_1 + x_2 + ... + x_n) = \frac{1}{n}\sum_{i-1}^{n} x_i.$$

Однако эта же сумма может быть получена, если провести суммирование по группам одинаковых значений:

$$\sum_{i=1}^{n} x_i = \sum_{j=1}^{k} n_j x_j,$$

где $n_j$ — количество значений в группе $j$. Тогда

$$\langle x \rangle = \frac{1}{n}\sum_{j=1}^{k} n_j x_j = \sum_{j=1}^{k} \frac{n_j}{n} x_j = \sum_{j=1}^{k} p_j x_j,$$

поскольку отношение $n_j/n$ есть не что иное, как относительная частота появления результата из группы $j$, которая в пределе (при $n \to \infty$) превращается в вероятность $p_j$. Таким образом, окончательно получаем, что

$$\langle x \rangle = \sum_{j=1}^{k} p_j x_j. \qquad (A.11)$$

**Пример А.4.**

На трех сторонах кубика нарисована цифра «2», на двух сторонах — цифра «4» и на одной стороне цифра «1». Какое количество очков в среднем выпадает при однократном броске кубика?

$n = 6$ (число возможных исходов);

$n_1 = 3$; $p_1 = 3/6$; $x_1 = 2$;

$n_2 = 2$; $p_2 = 2/6$; $x_2 = 4$;

$n_3 = 1$; $p_3 = 1/6$; $x_3 = 1$.

Согласно (А.11)

$$\langle x\rangle = \frac{3}{6}\cdot 2 + \frac{2}{6}\cdot 4 + \frac{1}{6}\cdot 1 = \frac{1}{6}(6+8+1) = \frac{15}{6} = 2{,}5.$$

Имеет смысл обратить внимание на то обстоятельство, что среднее значение выражается дробным числом, хотя все усредняемые значения были целыми — в дальнейшем мы неоднократно будем сталкиваться с такой ситуацией.

Величина, определяемая соотношением (А.11), называется *взвешенным средним для выборки* $x_1, x_2, \ldots, x_k$.

## А.3. Условная вероятность

Попробуем построить обобщение формулы для вероятности суммарного события (А.5) на ситуацию, когда отдельные события $A$ и $B$ могут оказаться совместными, т. е. произойти одновременно. В этом случае $p(A \vee B) \leqslant p(A) + p(B)$. Действительно, пусть по-прежнему событие $A$ выполняется при $m_1$ из $n$ равновероятных исходов опыта, а событие $B$ — при $m_2$ из тех же $n$ возможных исходов. Однако из-за того, что возможно совместное наступление $A$ и $B$, исходы $m_1$ и $m_2$ окажутся не полностью различными, т. е. среди них могут быть одни и те же. Тогда суммарное количество благоприятных исходов $m \leqslant m_1 + m_2$ и, следовательно, $p(A \vee B) \neq p(A) + p(B)$.

Пусть среди $m_1$ и $m_2$ имеются $m'$ общих исходов; очевидно, это те случаи, когда $A$ и $B$ наступают одновременно. Следовательно, вероятность *совместного* события

$$p(A \wedge B) = \frac{m'}{n},$$

а общее количество различных благоприятных исходов оказывается равным $m = m_1 + m_2 - m'$. Тогда вероятность

$$p(A\vee B) = \frac{m_1 + m_2 - m'}{n} = \frac{m_1}{n} + \frac{m_2}{n} - \frac{m'}{n} = p(A)+p(B)-p(A\wedge B).$$

Окончательно,

$$p(A \vee B) = p(A) + p(B) - p(A \wedge B). \qquad (А.12)$$

Таким образом, при определении вероятности суммы двух событий $(A \vee B)$ в общем случае требуется находить и вероятность их произведения $(A \wedge B)$. Это несложно сделать, если события $A$ и $B$ *независимы* — в этом случае можно воспользоваться формулой (А.9), подстановка которой в (А.12) дает

$$p(A \vee B) = p(A) + p(B) - p(A)p(B). \qquad (А.13)$$

Легко видеть, что (А.5) оказывается частным случаем (А.12) при условии несовместности $A$ и $B$, тогда $p(A \wedge B) = 0$.

**Пример А.5.**

Какова вероятность вынуть козырную карту или туза из колоды 36 карт, если козырной объявлена одна из мастей?

Событие $A$ — получение козыря — имеет вероятность $p(A) = 9/36 = 1/4$, поскольку карт одной масти 9. Событие $B$ — получение туза — имеет вероятность $p(B) = 4/36 = 1/9$, поскольку тузов 4; однако из них один — козырный (т. е. реализуются и $A$ и $B$); вероятность его появления $p(A \wedge B) = p(A)p(B) = 1/36$. Тогда согласно (А.13)

$$p(A \wedge B) = p(A) + p(B) - p(A)p(B) = 1/4 + 1/9 - 1/36 = 12/36 = 1/3.$$

Теперь при нахождении вероятности произведения событий $p(A \wedge B)$ попробуем учесть то обстоятельство, что события $A$ и $B$ не обязательно могут быть независимыми, — очевидно, это наиболее общий случай. Отсутствие независимости случайных событий означает, что одно из них оказывает влияние на другое, т. е. вероятность второго события зависит от того, произошло ли первое. Например, вы случайно встретили знакомого на вечеринке (событие $B$); однако решение пойти на эту вечеринку (событие $A$) вы приняли случайно, выбирая из нескольких возможностей; таким образом, случайное событие $B$ оказывается следствием случайного события $A$.

*Вероятность события $B$ при условии, что имело место влияющее на него событие $A$, называется* ***условной вероятностью****.*

Обозначать условную вероятность будем $p_A(B)$. Объективности ради следует заметить, что вероятность любого случайного события зависит от каких-то условий, при которых возможно его наступление или ненаступление. Например, условием того, что вероятность выпадения всех цифр игральной кости одинакова и равна $1/6$, является ее правильная геометрическая форма и однородность материала. Если условия изменятся (например, форма будет не куб, а параллелепипед), то изменится и вероятность. Просто договорились считать вероятность событий, для которых условия не изменяются в различных сериях опытов, *безусловной*; если же условия могут изменяться, используется термин «*условная вероятность*». Достаточно очевидным представляется также утверждение: если $A$ и $B$ независимы, то $p_A(B) = p(B)$. Более того, данное утверждение можно считать математически точным определением понятия «*независимые события*»:

*Два случайных события $A$ и $B$ являются* ***независимыми****, если их условные вероятности равны безусловным, т. е. $p_A(B) = p(B)$ и $p_B(A) = p(A)$.*

Пусть из $n$ равновероятных исходов событие $A$ реализуется $m$ способами, из которых $k$ являются благоприятными и для наступления события $B$, связанного с $A$. Тогда, очевидно,

$$p_A(B) = \frac{k}{m}.$$

Вероятность совместного выполнения событий $A \wedge B$

$$p(A \wedge B) = \frac{k}{n}.$$

Но

$$\frac{k}{n} = \frac{k}{m}\frac{m}{n} = p_A(B)p(A).$$

Окончательно имеем

$$p(A \wedge B) = p(A)p_A(B). \tag{A.14}$$

Полученное выражения является наиболее *общим правилом умножения вероятностей*; выражение (A.9), очевидно, оказывается частным случаем (A.14) при условии, что $A$ и $B$ независимы.

Подстановка полученного выражения в формулу (A.13) позволяет получить *общее правило сложения вероятностей*

$$p(A \vee B) = p(A) + p(B) - p(A)p_A(B). \tag{A.15}$$

Наконец, обобщим (A.14) на ситуацию, когда имеется несколько *несовместных* событий $B_1, \ldots, B_n$ и $A$ может реализоваться только через одно из них, но при этом не является достоверным. Тогда

$$p(A) = \sum_{i=1}^{n} p(B_i)p_{B_i}(A). \tag{A.16}$$

Эта формула называется *формулой полной вероятности.*

Перечислим (без доказательства) некоторые свойства условной вероятности:

1) условная вероятность $p_A(B)$ может быть как больше безусловной $p(B)$, так и меньше ее (т. е. событие $A$ может как понижать вероятность $B$, так и повышать ее); однако всегда $0 \leqslant p_A(B) \leqslant 1$. Для ситуации, когда $A$ с необходимостью влечет за собой $B$ (например, $A$ — выпадение четверки при бросании игральной кости, а $B$ — выпадение четной цифры) будем использовать обозначение

«$\subset$» ($A \subset B$ — читается «*$A$ влечет $B$*»). Очевидно, если $A \subset B$, то $p_A(B) = 1$. Если $A$ и $B$ несовместны, $p_A(B) = 0$;

2) если событие $A$ может реализоваться только через одно из *несовместных* событий $B_1, \ldots, B_n$; условиями независимости $A$ и $B_i$ будут:

$$p(A) = p_{B_i}(A), \quad i = 1, \ldots, n.$$

Если событий $A_j$ может быть несколько (образуют множество, $j = 1, \ldots, m$) и каждое из них может реализоваться через одно из несовместных событий $B_1, \ldots, B_n$, то условиями независимости $A_j$ и $B_i$ будут

$$p(A_j) = p_{B_i}(A_j), \quad i = 1 \ldots n;\ j = 1 \ldots m. \tag{A.17}$$

3) если $B \subset B'$, то $p_A(B) \leqslant p_A(B')$;

4) для дополнительных событий $p_A(\bar{B}) = 1 - p_A(B)$;

5) если $B$ и $C$ несовместны, то $p_A(B \vee C) = p_A(B) + p_A(C)$;

6) поскольку $p(A)p_A(B) = p(A \wedge B) = p(B \wedge A) = p(B)p_B(A)$, несложно получить выражение, которое называется *формулой Байеса*:

$$p_A(B) = p_B(A)\frac{p(B)}{p(A)}. \tag{A.18}$$

Последнее выражение легко обобщается на ситуацию с несколькими исходами $B_1 \ldots B_n$:

$$p_A(B_i) = p_{B_i}(A)\frac{p(B_i)}{p(A)}. \tag{A.19}$$

**Пример A.6.**

Имеется 6 карточек, на которые наносятся номера от 1 до 6, карточки переворачивают и перемешивают. Очевидно, вытянуть карточку с четной цифрой составляет 0,5. Каковы вероятности получить четную цифру во втором извлечении?

Возможны две ситуации:

1) первой была извлечена нечетная цифра; тогда в оставшихся 5 карточках содержится три с четными и, следовательно, вероятность оказывается $3/5 = 0{,}6$; т. е. первое извлечение повысило вероятность благоприятного исхода во втором;

2) первой была извлечена четная цифра — тогда в пяти оставшихся только две являются четными, т. е. вероятность получить четную во втором извлечении составляет 0,4; т. е. первое извлечение понизило вероятность благоприятного исхода во втором.

Этот пример служит иллюстрацией свойства 1 условной вероятности.

**Пример А.7.**

Имеется три урны, содержащие белые и черные шары, причем в первой урне 4 белых и 2 черных шара, во второй — 3 белых и 3 черных, в третьей — 2 белых и 4 черных. Из одной из урн (неизвестно из какой) наугад вынут шар. Какова вероятность того, что шар оказался белым при условии, что он вынут из первой урны?

Событие $A$ — выбор урны 1; его вероятность $p(A) = 1/3$.

Событие $B$ — достать белый шар при условии $A$; $p_A(B) = 2/3$.

Из (А.14) $p(A \wedge B) = p(A)p_A(B) = 2/9$.

Пусть событие $A$ — вытаскивание белого шара, а $B$ — то, что он вынут из первой урны. Из всех имеющихся шаров событию $A$ благоприятствует 9; из которых лишь 2 благоприятствуют событию $B$. Таким образом, $p_A(B) = 2/9$.

**Пример А.8.**

На карточках отдельными буквами написано слово «ПАПАХА». Карточки переворачивают, перемешивают и случайным образом открывают подряд 4 из них. Какова вероятность получить таким путем слово «ПАПА»?

Пусть событие $A$ — извлечение первой «П», событие $B$ — извлечение второй «А», $C$ — третьей «П» и $D$ — четвертой «А». Тогда интересующее событие можно записать как $A \wedge B \wedge C \wedge D$; его вероятность можно найти, применяя несколько раз формулу (А.14):

$$p(A) = 2/6 = 1/3;$$

$$p(A \wedge B) = p(A)p_A(B) = 1/3 \cdot 3/5 = 1/5;$$

$$p(A \wedge B \wedge C) = p(A \wedge B)p_{A\wedge B}(C) = 1/5 \cdot 1/4 = 1/20;$$

$$p(A \wedge B \wedge C \wedge D) = p(A \wedge B \wedge C)p_{A\wedge B\wedge C}(D) = 1/20 \cdot 2/3 = 1/30.$$

**Пример А.9.**

В магазин поступила новая продукция с трех предприятий. Процентный состав этой продукции следующий: 20 % — продукция 1-го предприятия, 30 % — продукция 2-го предприятия, 50 % — продукция 3-го предприятияю При этом 10 % продукции 1-го предприятия высшего сорта, на 2-м предприятии — 5 % и на 3-м — 20 % продукции высшего сорта. Найти вероятность того, что случайно купленная новая продукция окажется высшего сорта.

Обозначим через $B$ событие, заключающееся в том, что будет куплена продукция высшего сорта, а $A_1, A_2, A_3$ — события, заключающиеся в покупке продукции, принадлежащей 1-му, 2-му и 3-му предприятиям соответственно. Тогда $p(A_1) = 0{,}2 p_{A_1}(B) = 0{,}1$; $p(A_2) = 0{,}3$; $p_{A_2}(B) = 0{,}05$; $p(A_3) = 0,5 p_{A_3}(B) = 0{,}2$. Подставляя в (А.16), получаем

$$p(B) = 0{,}2 \cdot 0{,}1 + 0{,}3 \cdot 0{,}05 + 0{,}5 \cdot 0{,}21 = 0{,}135.$$

## Контрольные вопросы и задания к Приложению А

1. Почему в определении вероятности количество попыток $N \to \infty$? Зависит ли вероятность случайного события от числа проведенных однотипных опытов, в которых оно проявляется? Почему?

2. Мы наугад открываем страницу книги и наугад выбираем строку и номер буквы в ней. Можно ли считать случайным событием выбор конкретной буквы? Одинаковы ли вероятности «наткнуться» таким образом на различные буквы русского алфавита?

3. Известно, что в текстах на русском языке наиболее часто встречаемым символом является «пробел» (разделитель слов) — вероятность его появления составляет около 17 %. На основании этих данных определите среднюю длину слова русского языка.

4. Какие исходы случайного события названы равновероятными? Как обосновывается равновероятность при решении практических задач?

5. Дайте определения дополнительным случайным событиям; приведите примеры. Получите формулу, связывающую их вероятности.

6. Какие случайные события называются независимыми? Совместными? Несовместными? К какой из этих категорий следует отнести дополнительные события?

7. Какова вероятность того, что при бросании игральной кости выпадет число, делящееся на 3?

8. Какова вероятность того, что при бросании двух игральных костей

a) выпадет сумма очков, делящаяся на 3?

b) первой выпадет «1»?

c) выпадут две одинаковые цифры («пара»)?

d) выпадет хотя бы одна «6»?

9. При бросках монеты 3 раза подряд выпадала «решка». Какова вероятность того, что

a) в 4-м броске снова выпадет «решка»?

b) выкинуть «решку» 4 раза подряд?

10. В доме 8 этажей (выше 1-го), на каждом живут по 10 человек. Я живу на 5-м этаже. Человек садится на 1-м этаже со мной в лифт и едет наверх. Найти вероятность того, что:

a) человек живет на моем этаже?

b) человек живет на следующем этаже?

c) человек живет выше меня?

d) человек не живет на моем этаже?

11. Докажите следующее утверждение: если имеются дополнительные события $A$ и $\bar{A}$, а также некоторое независимое им событие $B$, то справедливо соотношение

$$p((A \wedge B) \vee (\bar{A} \wedge B)) = p(B).$$

12. На соревнованиях по стрельбе на мишенях нанесены области с очками 0, 2, 6 и 10. Стрелок A сделал 100 выстрелов, а стрелок B — 150 выстрелов; при этом они показали следующие результаты:

| Очки | 0 | 2 | 6 | 10 |
|---|---|---|---|---|
| Число попаданий A | 5 | 20 | 65 | 10 |
| Число попаданий B | 15 | 15 | 90 | 30 |

Кого из них следует признать более метким стрелком?

13. В чем отличие условной и безусловной вероятностей? Можно ли считать безусловную вероятность предельным случаем условной?

14. Приведите пример ситуации со случайными связанными событиями $A$ и $B$, когда выполняются соотношения: $p_A(B) > p(B)$ и $p_A(B) < p(B)$.

15. Докажите свойства условной вероятности 2–4 в п. А.3.

16. Докажите, что $p(A)p_A(B) = p(B)p_B(A)$.

17. Докажите, что $p(B) = p(A)p_A(B) + p(\bar{A})p_{\bar{A}}(B)$ (рекомендация: воспользуйтесь соотношением из задания 11).

18. Решите задачу из примера А.7 для черного шара.

19. В ящике 4 белых и 6 черных шаров. Случайным образом и без возврата извлекаются два шара. Найти вероятность того, что:

a) будут вынуты 2 белых шара?

b) будут вынуты 2 черных шара?

c) будут вынуты 1 белый и 1 черный шар (в любой последовательности)?

d) будет вынут сначала черный, а затем белый шар?

20. Из колоды 36 игральных карт случайным образом выбирают 5. Какова вероятность:

a) получить «пару» — две карты одинакового значения (например, две семерки)?

b) получить все карты одинаковой масти?

21. Один из трех стрелков вызывается на линию огня и производит два выстрела. Вероятность попадания в мишень при одном выстреле для первого стрелка равна 0,3, для второго — 0,5; для третьего — 0,8. Мишень не поражена. Найти вероятность того, что выстрелы произведены первым стрелком. (Рекомендация: воспользуйтесь формулой (А.19).)

# Приложение В.
# Некоторые соотношения логики

Рассмотрим без доказательства ряд соотношений математической логики, которые могут оказаться необходимыми в ряде разделов курса. Интересующихся доказательной стороной можно адресовать к многочисленным учебникам по математической логике.

Пусть $x_1, x_2$ и $x_3$ — переменные булевского типа (логические переменные), способные принимать лишь два значения (*True* и *False*), которые для удобства мы будем обозначать 1 и 0 соответственно.

Над логическим переменными определен ряд *простейших* логических функций, значения которых определяются по правилам, приведенным в табл. В.1.

Таблица В.1

| $x_1$ | $x_2$ | $x_1 \wedge x_2$ | $x_1 \vee x_2$ | $x_1 \to x_2$ | $x_1 \oplus x_2$ | $x_1 \sim x_2$ | $\neg x_1$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | |

Функции имеют следующие названия и обозначения:

- *конъюнкция* (логическое умножение) — $\wedge$ (И);
- *дизъюнкция* (логическое сложение) — $\vee$ (ИЛИ);
- *импликация* — $\to$;
- *сумма по модулю 2* — $\oplus$;
- *функция эквивалентности* — $\sim$;
- *отрицания* (логическое отрицание) — $\neg$ (НЕ).

Справедливы следующие *соотношения эквивалентности* между простейшими функциями:

$$x_1 \to x_2 = \neg x_1 \vee x_2;$$

$$x_1 \oplus x_2 = (\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2);$$
$$x_1 \sim x_2 = (\neg x_1 \wedge \neg x_2) \vee (x_1 \wedge x_2),$$

т. е. истинно элементарными являются логические функции И, ИЛИ и НЕ — остальные могут быть выражены через них.

Для выражений, построенных с использованием функций И, ИЛИ, НЕ, справедливы следующие свойства:

$$\left.\begin{array}{r}(x_1 \wedge x_2) \wedge x_3 = x_1 \wedge (x_2 \wedge x_3);\\ (x_1 \vee x_2) \vee x_3 = x_1 \vee (x_2 \vee x_3)\end{array}\right\}\ \text{свойства ассоциативности;}$$

$$\left.\begin{array}{r}x_1 \wedge x_2 = x_2 \wedge x_1\\ x_1 \vee x_2 = x_2 \vee x_1\end{array}\right\}\ \text{свойства коммутативности;}$$

$$\left.\begin{array}{r}x \wedge x = x;\\ x \vee x = x\end{array}\right\}\ \text{свойства идемпотентности;}$$

$$\left.\begin{array}{l}x_1 \wedge (x_2 \vee x_3) = (x_1 \wedge x_2) \vee (x_1 \wedge x_3);\\ x_1 \vee (x_2 \wedge x_3) = (x_1 \vee x_2) \wedge (x_1 \vee x_3)\end{array}\right\}\ \text{свойства дистрибутивности;}$$

$$\left.\begin{array}{r}\neg(x_1 \wedge x_2) = \neg x_1 \vee \neg x_2;\\ \neg(x_1 \vee x_2) = \neg x_1 \wedge \neg x_2\end{array}\right\}\ \text{законы де Моргана;}$$

$$\neg(\neg x) = x \text{ закон двойного отрициния.}$$

Соотношения, в которые входят логические константы:

$$x \wedge 0 = 0; \quad x \wedge 1 = x; \quad x \wedge (\neg x) = 0;$$
$$x \vee 1 = 1; \quad x \vee 0 = x; \quad x \vee (\neg x) = 1.$$

Соотношения (правила), позволяющие упрощать логические выражения:

$$\left.\begin{array}{r}x_1 \vee (x_1 \wedge x_2) = x_1;\\ x_1 \wedge (x_1 \vee x_2) = x_1\end{array}\right\}\ \text{правила поглощения;}$$

$$(x_1 \wedge x_2) \vee (x_1 \wedge (\neg x_2)) = x_1 \quad \text{правило склеивания;}$$

$$(x_1 \wedge x_2) \vee (\neg x_1) = x_2 \vee (\neg x_1) \quad \text{правило вычеркивания.}$$

Несколько слов о формализме записи логических выражений. Приведенные выше обозначения трех основных логических функций является «классическим», однако не очень удобным при решении практических задач (например, в теории автоматов).

Можно предложить другой способ записи, сходный с обозначением арифметических операций:

И ($\wedge$) — *логическое умножение* — обозначать знаком «$\cdot$»;

ИЛИ ($\vee$) — *логическое сложение* — обозначать знаком «$+$»;

НЕ ($\neg$) — отрицание — обозначать чертой над переменной.

Тогда получающиеся выражения будут подобны арифметическим с формальным выполнением всех законов арифметики (дистрибутивного, ассоциативного и пр.) — с точки зрения автора, это заметно упрощает вычисление значений логических выражений.

**Пример В.1.**

Требуется упростить логическое выражение

$$(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge x_2 = (x_1 + x_2) \cdot (x_1 + \bar{x}_2) + x_2 =$$
$$= x_1 \cdot x_1 + x_2 \cdot x_1 + x_1 \cdot \bar{x}_2 + x_2 \cdot \bar{x}_2 + x_2 =$$
$$= x_1 + x_1 \cdot (x_2 + \bar{x}_2) + 0 + x_2 = x_1 + x_1 \cdot 1 + x_2 = x_1 + x_2.$$

# Глоссарий

*Алгоритм (нестрогое определение)* — точно определенная (однозначная) последовательность простых (элементарных) действий, обеспечивающих решение любой задачи из некоторого класса (9.1).

*Алгоритм* — любая конечная система правил преобразования информации (данных) над любым конечным алфавитом (определение В.М. Глушкова) (9.3.1).

*Алгоритм структурный*, если он может быть представлен стандартным функциональным блоком (10.3).

*Алфавит* — набор знаков, в котором установлен порядок их следования (лексикографический порядок) (1.2).

*Анализ* — метод исследования, основанный на выделении отдельных компонентов системы и рассмотрении их свойств и связей (12.2.2).

*Аналоговая форма представления информации* — представление сообщения, содержащего информацию, посредством сигналов, информационный параметр которых является непрерывной функцией времени (1.2).

*Бит* — единица измерения энтропии при двух возможных равновероятных исходах опыта (2.1.1).

*Вес кодовой комбинации* — число ненулевых (единичных) разрядов в данной кодовой комбинации (6.3.1).

*Внешние запоминающие устройства* (ВЗУ) — устройства, выполняющие операции, связанные с сохранения и считывания данных на материальном носителе (8.4).

*Данные* — сведения, характеризующие какую-то систему, явление, процесс или объект, представленные в определенной форме и предназначенные для дальнейшего использования (8.1).

*Декодер* — устройство, обеспечивающее выполнение операции декодирования (6.3.3).

*Декодирование* — операция, обратная кодированию, т. е. восстановление информации в первичном алфавите по полученной последовательности кодов (3.1).

*Дискретный канал* — канал связи, используемый для передачи дискретных сообщений (5.2).

*Дискретные устройства* — те, у которых дискретны множества внутренних состояний, входных и выходных сигналов, а также множество моментов времени, в которые поступают входные сигналы, меняются внутренние состояния и выдаются выходные сигналы (11.1).

*Дискретная форма представления информации* — представление сообщения, содержащего информацию, посредством конечного числа знаков (алфавита) (1.2).

*Документ* — продукт, сформированный в результате исполнения некоторой программы (12.4).

*Запись логическая* — поименованная совокупность элементарных данных, имеющая смысловую завершенность (8.3.2).

*Запись физическая* — элемент поверхности носителя, на котором в соответствии с физическими принципами функционирования носителя размещаются данные, составляющие логическую запись (8.4.1).

*Запоминающие устройства с произвольным доступом* — те, в которых доступ к данным осуществляется по адресу ячейки, где они хранятся (8.2).

*Знак* — элемент некоторого конечного множества отличных друг от друга сущностей, используемого для представления дискретных сигналов (1.2).

*Избыточность языка* — мера бесполезно совершаемых альтернативных выборов при чтении текста, обусловленная его лексическими особенностями (2.3).

*Избыточность первичного кода* — показатель относительного увеличения длины исходного сообщения, обусловленного кодированием (3.1).

*Избыточность помехоустойчивого кода* — доля помехозащищенной кодовой комбинации, которая не содержит первичной информации (6.1).

*Информатика* — фундаментальная естественная наука, изучающая общие свойства информации, процессы, методы и средства ее обработки (сбор, хранение, преобразование, перемещение, выдача) (определение А.П. Ершова и Б.Н. Наумова) (введение).

*Информация* (статистическое определение) — это содержание сообщения, понижающего неопределенность некоторого опыта с неоднозначным исходом; убыль связанной с ним энтропии является количественной мерой информации (2.2).

*Информационный процесс* — изменение с течением времени содержания информации или представляющего его сообщения (1.1).

*Исполнитель алгоритма* — субъект или устройство, способные правильно интерпретировать описание алгоритма и выполнить содержащийся в нем перечень действий (9.1).

*Источник информации* — субъект или объект, порождающий информацию и представляющий ее в виде сообщения (1.1).

*Источники информации шенноновские* — см. *Сообщения шенноновские.*

*Источники информации с памятью* — см. *Сообщения с памятью.*

*Канал связи* — материальная среда, а также физический или иной процесс, посредством которого осуществляется передача сообщения, т. е. распространение сигналов в пространстве с течением времени (5.1).

*Класс* — множество объектов, обладающих одним или несколькими одинаковыми атрибутами; эти атрибуты называются *полем свойств класса* (12.2.1).

*Классификация* — распределение однотипных объектов в соответствии с выделенными свойствами (признаками, категориями, классами) (12.1.2).

*Ключ* — некоторый секретный параметр шифра (обычно последовательность знаков алфавита), позволяющий выбрать для шифрования только одно конкретное преобразование из всего множества преобразований, составляющих шифр (7.1.1).

*Код* — (1) правило, описывающее соответствие знаков или их сочетаний одного алфавита знакам или их сочетаниям другого алфавита; (2) знаки вторичного алфавита, используемые для представления знаков или их сочетаний первичного алфавита (3.1).

*Кодер* — устройство, обеспечивающее выполнение операции кодирования (3.1).

*Кодирование* — перевод информации, представленной посредством первичного алфавита, в последовательность кодов (3.1).

*Конечный автомат* — система $\langle X, Y, Q, \Psi, \Theta \rangle$, в которой $X$ и $Y$ являются конечными входным и выходным алфавитами, $Q$ — конечным множеством внутренних состояний, $\Psi(x, q)$ — функцией переходов и $\Theta(x, q)$ — функцией выходов (11.3).

*Криптографическое преобразование* — преобразование исходной информации, заключающееся в приведении составляю-

щих ее элементов (слов, букв, цифр) с помощью специальных алгоритмов к виду, не позволяющему воспроизвести исходные данные без знания секрета обратного преобразования (восстановления) или специального ключа (7.1.1).

*Криптографическая система (криптосистема)* — множество обратимых преобразований открытого текста в шифрованный (7.1.1);

— *симметричная* (закрытого типа) — (7.2.1);

— *асимметричная* (с открытым ключом) — (7.3).

*Линия связи* — совокупность средств связи и канала связи, посредством которых осуществляется передача инфорамции от источника к приемнику (5.1.1).

*Массив* — упорядоченная линейная совокупность однородных данных (8.3.1).

*Материальный носитель информации* — материальный объект или среда, которые служат для представления или передачи информации (1.1).

*Машинное слово* — совокупность двоичных элементов, обрабатываемая как единое целое в устройствах и памяти компьютера (8.2).

*Моделирование* — построение упрощенного варианта прототипа, обеспечивающего приемлемую для данной задачи точность описания его строения или поведения (12.1.1).

*Моделирование имитационное* — метод исследования, основанный на том, что изучаемый прототип заменяется ее *имитатором* — натурной или информационной моделью, с которым и проводятся эксперименты с целью получения информации об особенностях прототипа (12.1.2).

*Модель* — объединение составных частей (элементов) и связей между ними, отражающая существенные для данной задачи свойства прототипа (12.1.1).

*Модель математическая* — множество элементов произвольной природы, на которых определено конечное множество отношений (12.1.3).

*Непрерывный канал* — канал, который обеспечивает передачу непрерывных (аналоговых) сигналов (5.4).

*Объект* — простейшая составляющая сложного объединения, обладающая следующими качествами:

- в рамках данной задачи он не имеет внутреннего устройства и рассматривается как единое целое;

- у него имеется набор свойств (атрибутов), которые изменяются в результате внешних воздействий;
- он идентифицирован, т. е. имеет имя (название) (12.2.1).

*Однородный дискретный канал* — если для любой пары $i$ и $j$ условная вероятность $p_{a_i}(b_j)$ с течением времени не изменяется (т. е. влияние помех все время одинаково) (5.2).

*Оптимальный* $(n, k)$*-код* — код, который обеспечивает минимальную вероятность ошибочного декодирования среди всех иных кодов с теми же $n$ и $k$ (6.4).

*Помехоустойчивый код* — код, позволяющий обнаружить и при необходимости исправить ошибки в принятом сообщении (6.1).

*Правило интерпретации сообщения* — соотношение (закон), устанавливающий соответствие между сообщением и содержащейся в нем информацией (1.1).

*Приемник информации* — субъект или объект, способный принять сообщение и правильно его интерпретировать (1.1).

*Программа* — последовательность действий по обработке информации исполнителем «компьютер» (12.4).

*Программный объект* — это совокупность некоторого набора данных и процедур, определяющих возможности их изменения (12.4).

*Пропускная способность канала связи* — максимальное количество информации, передаваемое по каналу за единицу времени (5.2).

*Свойство* (*атрибут*) — качество объекта, для которого установлена мера (12.2.1).

*Сигнал* — изменение характеристики материального носителя, которое используется для представления информации (1.1).

*Сигнал непрерывный* (*аналоговый*) — его параметр может принимать любое значение в пределах некоторого интервала (1.2).

*Сигнал дискретным* — его параметр может принимать конечное число значений в пределах некоторого интервала (1.2).

*Синтез* — (1) метод исследования (изучения) системы в целом (т. е. компонентов в их взаимосвязи), сведение в единое целое данных, полученных в результате анализа; (2) создание системы путем соединения отдельных компонентов на основании законов, определяющих их взаимосвязь (12.2.2).

*Синтез конечного автомата* (11.3.2).

*Система* — совокупность взаимодействующих компонентов, каждый из которых в отдельности не обладает свойствами системы в целом, но является ее неотъемлемой частью (12.2.2).

*Систематический код* — $(n, k)$-код), в котором значения всех проверочных бит $(p_j)$ определяются линейными комбинациями информационных бит $(u_i)$ (5.3).

*Система счисления* — правило записи чисел с помощью заданного набора специальных знаков — цифр (4.1).

*Система счисления позиционная* — те, в которых значение каждой цифры в изображении числа определяется ее положением (позицией) в ряду других цифр (4.1).

*Сложность алгоритма временна́я* — функция, которая каждой входной длине слова $n$ ставит в соответствие *максимальное* (для всех конкретных однотипных задач длиной $n$) время, затрачиваемое алгоритмом на ее решение (9.7).

*Сообщение* — последовательность сигналов (1.1).

*Сообщения (источники) с памятью* — те, в которых в которых существуют статистические связи (корреляции) между знаками или их сочетаниями (2.3).

*Сообщения (источники) шенноновские* (без памяти) — те, в которых вероятность появления любого отдельного знака алфвита в любом месте сообщения остается неизменной и не зависит от того, какие знаки предшествовали данному (2.3).

*Средства связи* — совокупность устройств, обеспечивающих преобразование первичного сообщения от источника информации в сигналы заданной физической природы, их передачу и прием (5.1).

*Структура данных* — перечень объединяемых одиночных данных, их характеристики, а также особенности связей между ними образуют (8.1).

*Схема* — это комбинация базисных элементов, в которой выходы одних элементов присоединяются к входам других (11.2).

*Тезис Тьюринга:* всякий алгоритм может быть задан посредством тьюринговой функциональной схемы и реализован в соответствующей машине Тьюринга (9.3).

*Тезис Черча*: класс алгоритмически (или машинно) вычислимых частичных числовых функций совпадает с классом всех частично рекурсивных функций (9.2).

*Теорема Бома-Джакопини*: любой алгоритм может быть сведен к структурному (10.3).

*Теорема Котельникова (теорема отсчетов)*: непрерывный сигнал можно полностью отобразить и точно воссоздать по последовательности измерений или отсчетов величины этого сигнала через одинаковые интервалы времени, меньшие или равные половине периода максимальной частоты, имеющейся в сигнале (1.3).

*Теорема Шеннона* (*первая*): при отсутствии помех передачи всегда возможен такой вариант кодирования сообщения, при котором среднее число знаков кода, приходящихся на один знак кодируемого алфавита, будет сколь угодно близко к отношению средних информаций на знак первичного и вторичного алфавитов (3.1).

*Теорема Шеннона* (*вторая*): при передаче информации по каналу с шумом всегда имеется способ кодирования, при котором сообщение будет передаваться со сколь угодно высокой достоверностью, если скорость передачи не превышает пропускной способности канала (6.1).

*Условие Фано*: неравномерный код может быть однозначно декодирован, если никакой из кодов не совпадает с началом какого-либо иного более длинного кода (3.2.3).

*Файл* — определенным образом оформленная совокупность физических записей, рассматриваемая как единое целое и имеющая описание в системе хранения информации (8.4.1).

*Формальная грамматика* — система правил, описывающая множество конечных последовательностей символов формального алфавита (10.1.1).

*Формальный исполнитель* — субъект или устройство, способные воспринимать и анализировать указания алгоритма, изменять в соответствии с ним свое состояние, а также обладающие механизмом исполнения, способным выполнять пошаговую обработку информации (10.2.1).

*Формальная система* — это математическая модель, задающая множество дискретных компонентов путем описания исходных объектов и правил построения новых компонентов из исходных и уже построенных (12.2.3).

*Функциональный блок* — часть алгоритма, организованная как простое действие, т. е. имеющая один вход (выполнение начинается всегда с одного и того же действия) и один выход (10.3).

*Хеш* — выходная битовая строка фиксированной длины, независящей от его объема электронного документа, получаемая после его обработки по определенному алгоритму (7.4.1).

*Черный ящик* — система, строение которой неизвестно пользователю, однако известна ее реакция на определенные внешние воздействия (12.4).

*Шифрование* — процесс нормального применения криптографического преобразования открытого текста на основе алгоритма и ключа, в результате которого возникает шифрованный текст; методы шифрования (7.2.2).

*Экономичность системы счисления* — количество чисел, которое можно записать в данной системе с помощью определенного количества цифр (4.2.3).

*Электронная (цифровая) подпись* — присоединяемое к электронному документу его криптографическое преобразование, которое позволяет при получении текста другим пользователем проверить его авторство и целостность (7.4).

*Энтропия* — мера неопределенности опыта, в котором проявляются случайные события, равная средней неопределенности всех возможных его исходов (2.1).

# Литература


1. Аветисян Р.Д., Аветисян Д.О. Теоретические основы информатики. — М.: РГГУ, 1997. — 167 с.

2. Алексеев П.А. Информатика 2001. — М: СОЛОН-Р, 2001. — 364 с.

3. Аршинов М.Н., Садовский Л.Е. Коды и математика. — М.: Наука, 1983. — 144 с.

4. Анисимова Н.С. и др. Информатика и вычислительная техника. Алгоритмизация и основы программирования: Учебное пособие. — СПб.: ЛГОУ, 1997. — 132 с.

5. Бауэр Ф., Гооз Г. Информатика. Вводный курс. — М.: Мир, 1976. — 484 с.

6. Берлекэмп Э. Алгебраическая теория кодирования. — М.: Мир, 1971. — 478 с.

7. Брассар Ж. Современная криптология. — М.: ПОЛИМЕД, 1999. — 176 с.

8. Бриллюэн Л. Наука и теория информации. — М.: ГИФМЛ, 1960. — 392 с.

9. Ватолин Д. Методы сжатия данных. — www.compresion.ru.

10. Вирт Н. Систематическое программирование. Введение. — М.: Мир, 1977. — 183 с.

11. Вольфовиц Д. Теоремы кодирования теории информации. — М.: Мир, 1967. — 248 с.

12. Гаврилов М.А. и др. Логическое проектирование дискретных автоматов. Языки, методы, алгоритмы. — М.: Наука, 1974. — 351 с.

13. Галлагер Р. Теория информации и надежная связь. — М.: Советское радио, 1974. — 238 с.

14. Глушков В.М. Основы безбумажной информатики. — М.: Наука, 1987. — 552 с.

15. Гнеденко Б.В. Курс теории вероятностей. — М.: Наука, 1969. — 440 с.

16. Гольдштейн С.Л., Ткаченко Т.Я. Введение в системологию и системотехнику. — Екатеринбург: ИРРО, 1994. — 198 с.

17. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. — М.: Мир, 1982. — 416 с.

18. Дехтярь М.И. Введение в схемы, автоматы и алгоритмы. — Интернет-университет информационных технологий — ИНТУИТ.РУ. http://www.intuit.ru/department/ds/introsaa/

19. Информатика: Учебник / Под ред. Н.В. Макаровой. — М.: Финансы и статистика, 1997. — 768 с.

20. Касами Т. и др. Теория кодирования. — М.: Мир, 1978. — 576 с.

21. Коини С.К. Математическая логика. — М.: Мир, 1973. — 480 с.

22. Колмогоров А.Н. Основные понятия теории вероятностей. — М.: Наука, 1974. — 119 с.

23. Колмогоров А.Н., Драгалин А.Г. Математическая логика. — М.: Изд-во МГУ, 1984. — 119 с.

24. Криптографическая защита информации: учебное пособие / А.В. Яковлев, А.А. Безбогов, В.В. Родин, В.Н. Шамкин. — Тамбов : Изд-во Тамб. гос. техн. ун-та, 2006. — 140 с.

25. Кудрявцев В.Б., Подколзин А.С., Ушчумлич Ш. Введение в теорию абстрактных автоматов. — М.: Изд-во МГУ, 1985. — 174 с.

26. Мазур М. Качественная теория информации. — М.: Мир, 1974. — 239 с.

27. Мальцев А.И. Алгоритмы и рекурсивные функции. — М.: Наука, 1965. — 391 с.

28. Марков А.А. Элементы математической логики. — М.: Изд-во МГУ, 1984. — 78 с.

29. Марков А.А. Введение в теорию кодирования. — М.: Наука, 1982. — 364 с.

30. Могилев А.В., Пак Н.И., Хеннер Е.К. Информатика: Учеб. пособие для студ. пед. вузов. — М.: Академия, 1999. — 816 с.

31. Нечаев В.И. Элементы криптографии (Основы теории защиты информации): учебное пособие для университетов и педвузов / Под.ред. В.А. Садовничего. — М.: Высш. шк., 1999. — 256 с.

32. Острейковский В.А. Информатика. — М.: Высш. шк., 1999. — 511 с.

33. Павловский Ю.Н. Имитационные модели и системы. — М.: Фазиз, 2000.

34. Пирс Дж. Символы, сигналы, шумы. — М.: Мир, 1967. — 334 с.

35. Питерсон У., Уэлдон Э. Коды, исправляющие ошибки. — М.: Мир, 1976. — 594 с.

36. Реньи А. Трилогия о математике. — М.: Мир, 1980. — 376 с.

37. Роганов Е.А. Практическая информатика. Интернет-университет информационных технологий — ИНТУИТ.РУ. — http://www.intuit.ru/department/se/pinform/

38. Стратонович Р.Л. Теория информации. — М.: Советское радио, 1975. — 424 с.

39. Трахтенброт Б.А. Алгоритмы и вычислительные автоматы. — М.: Советское радио, 1974. — 200 с.

40. Успенский В.А. Машина Поста. — М.: Наука, 1988. — 96 с.

41. Файнстейн А. Основы теории информации. — М.: ИЛ, 1960. — 233 с.

42. Фомин С.В. Системы счисления. — М.: Наука, 1987. — 48 с.

43. Хэмминг Р.В. Теория кодирования и теория информации. — М.: Радио и связь, 1983. — 176 с.

44. Чисар И., Кернер Я. Теория информации: теоремы кодирования для дискретных систем без памяти. — М.: Мир, 1985. — 400 с.

45. Чистяков В.П. Курс теории вероятностей. — М.: Наука, 1987. — 240 с.

46. Шеннон К. Математическая теория связи // Работы по теории информации и кибернетике. — М.: ИЛ, 1963. С. 243–332.

47. Шеннон К. Теория связи в секретных системах // Введение в криптографию / Под ред. В.В. Ященко. — М.: МИФИ, 1999. С. 235–272.

48. Шеннон Р.Е. Имитационное моделирование систем — искусство и наука. — М.: Мир, 1978. — 418 с.

49. Шоломов Л.А. Основы теории дискретных логических и вычислительных устройств. — М.: Наука, 1980. — 400 с.

50. Яглом А.М., Яглом И.М. Вероятность и информация. — М.: Наука, 1973. — 511 с.

# Оглавление

*Адрес издательства в Интернет WWW.TECHBOOK.RU*

Учебное издание

**Стариченко** Борис Евгеньевич

**Теоретические основы информатики**

Учебник для вузов

Редактор Ю. Н. Чернышов
Компьютерная верстка Ю. Н. Чернышова
Обложка художника О. В. Карповой

Подписано в печать 01.10.2015. Формат 60×88/16. Уч. изд. л. 25.
Тираж 1000 экз. (1-й завод 500 экз.)