Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

ИНТЕЛЛЕКТУАЛЬНЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В КОМПЬЮТЕРНЫХ СИСТЕМАХ И СЕТЯХ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для специальности 1-40 02 01 «Вычислительные машины, системы и сети»

Минск БГУИР 2019

УДК [004.8+004.7](075.8)
ББК 32.813я73+32.971.35я73
И73

Авторы:

А. А. Дудкин, Д. И. Самаль, В. А. Головко, Л. П. Матюшков

Рецензенты:

кафедра информационных систем управления
Белорусского государственного университета
(протокол №4 от 26.10.2017);

доцент кафедры экономики и управления научными исследованиями,
проектированием и производством
Белорусского национального технического университета
кандидат экономических наук, доцент П. В. Мелюшин

Интеллектуальные информационные технологии в компьютерных
И73 системах и сетях : учеб.-метод. пособие / А. А. Дудкин [и др.]. – Минск : БГУИР, 2019. – 157 с. : ил.
ISBN 978-985-543-443-7.

Рассмотрены понятия теории алгоритмов, формальных языков, грамматик и автоматов, формальные модели алгоритмов, а также содержится информация по основам знаний в области искусственного интеллекта и периферийных устройств вычислительной техники. Особое внимание уделяется рассмотрению базовых понятий и прикладным аспектам использования теоретических положений в современной технике и обществе.

Издание адресовано студентам, изучающим учебные дисциплины «Интерфейсы и периферийные устройства» и «Структурная и функциональная организация ЭВМ». Может быть полезно магистрантам и специалистам, интересующимся проблемами современных компьютерных технологий в построении вычислительных машин, систем и сетей и их использовании в различных приложениях.

УДК [004.8+004.7](075.8)
ББК 32.813я73+32.971.35я73

ISBN 978-985-543-443-7

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Понятие *технологии* имеет своими истоками материальное производство. Поэтому технологию до появления ЭВМ определяли как науку о способах воздействия на сырье, материалы и полуфабрикаты соответствующими орудиями производства. Носителем технологии выступал человек. Он знал, каким способом, средствами и инструментами реализовать процесс преобразования исходных ресурсов или полуфабрикатов. Появление ЭВМ как инструмента, способного выступить в качестве носителя технологии, привело к выработке понятия информационной технологии, а в последующем и интеллектуальной информационной технологии. Новое качество в производстве появилось благодаря введению в реализацию технологий комплекса средств для сбора, обработки, хранения, передачи и отображения информации без участия человека или с его незначительным участием. Технологии, реализуемые с применением таких средств, стали называть *информационными*. Главным элементом в них стала дистанционная передача информации и ее обработка, иногда с выработкой простейших управленческих решений на основе типовых стационарных схем.

Под интеллектуальными информационными технологиями обычно понимают такие, в которых предусмотрены возможности использования при решении плохо формализуемых задач баз знаний, принятия решения на основе нечеткой логики, нейронных сетей, вероятностных вычислений и т. п. *Интеллектуальные информационные технологии* – это средство для разработки интеллектуальных информационных систем, которые в последнее время становятся весьма распространенным коммерческим продуктом, находящим широкий спрос пользователей в самых разнообразных областях деятельности. Примерами таких систем являются экспертные системы, системы интеллектуального управления, интеллектуальные базы данных, системы когнитивной графики, самообучающиеся системы, адаптивные информационные системы. Многие из перечисленных систем могут быть реализованы как нечеткие системы, в которых используется лингвистическая модель представления информации, а решение задачи осуществляется на основе нечеткого логического вывода – частного случая вывода на знаниях.

Наблюдается рост «интеллектуализации» различных автоматизированных систем в технике, управлении и быту. Поэтому крайне необходимо для специалистов всех профилей владеть общими подходами в использовании современных компьютерных технологий во всех областях деятельности. Это позволит повысить эффективность их применения также за счет квалифицированного выбора готовых программных продуктов или при заказе индивидуальных проектов вплоть до безлюдных производств и полностью автоматических систем с принятием решений с использованием систем искусственного интеллекта. Поэтому в пособии на доступном уровне для специалистов разных профилей излагаются сведения об основных элементах и подходах, применяемых в проектировании, конструировании и эксплуатации современных интеллектуальных систем, сложных конструкций, роботов с имитацией ряда функций человека, бы-

товой техники с программным управлением, перенастраиваемых систем на выпуск изделий близкого класса в зависимости от потребностей рынка). Особую роль, конечно, играют и процессы моделирования различных объектов с использованием сетей ЭВМ, безбумажного ведения документации с применением электронной цифровой подписи, включая и документооборот.

Любые системы машин быстро совершенствуются, им на смену всегда приходят более сложные и эффективные, но законы и принципы, лежащие в основе их создания и функционирования, более долговечны и составляют фундамент, воздвигаемый поколениями людей. Задача данного пособия – в краткой и доступной форме ознакомить с основами этих знаний. Их составляют теория алгоритмов и автоматов, формальные языки, базы данных и знаний, операционные системы и системы управления базами данных, безбумажное ведение различных систем документирования, нейронные сети.

1 АБСТРАКТНЫЕ АВТОМАТЫ И ПОНЯТИЕ АЛГОРИТМА

1.1 Представление числовой информации

Системы счисления. Предшественниками вычислительных машин являются автоматы – устройства, выполняющие некоторый процесс без участия человека. Одно из отличий вычислительных устройств наших дней от традиционных средств состоит в том, что они стали машинами, пригодными для автоматизации умственной деятельности человека.

Необходимость формального описания компьютера и его отдельных частей в процессе его проектирования требует применения специального математического аппарата, который необходим при любых разработках различных методов обработки информации, при синтезе и анализе любых информационных процессов. Для этого вводится понятие абстрактного (т. е. существующего не реально, а лишь в воображении) цифрового автомата (ЦА).

Теория автоматов вошла в обиход в 50-е годы XX столетия, хотя соответствующая проблематика начала складываться еще в 30-е годы в рамках теории автоматов и релейных устройств. Тогда в теории алгоритмов были сформулированы понятия вычислительного автомата (машины Тьюринга и Поста). Было установлено, что для осуществления всевозможных преобразований информации вовсе не обязательно строить каждый раз специализированные автоматы: все это можно сделать на одном универсальном автомате при помощи подходящей программы и соответствующего кодирования. Этот теоретический результат позже получил инженерное воплощение в виде современных универсальных вычислительных машин.

Различие между задачами теории алгоритмов и теории автоматов можно кратко охарактеризовать как отличие вопросов о том, что могут делать автоматы и как они это делают. Естественно, что основой ЦА являются методы представления и обработки информации в виде цифр в закодированной форме в некотором алфавите.

Выбор алфавита для описания ЦА влияет на сложность его технической реализации. Для представления чисел существует много систем счисления. *Системой счисления* называется совокупность цифровых знаков и правил их записи для однозначного изображения чисел. Среди них выделяют два крупных класса: позиционные и непозиционные системы.

Непозиционными называются системы, в которых значение каждой цифры не зависит от ее позиции в числе. Примером непозиционной системы счисления служит римская система, в которой вместо цифр используются латинские буквы. Все остальные числа получаются при помощи двух арифметических операций: сложения и вычитания. Вычитание производится тогда, когда знак, соответствующий меньшему узловому числу, стоит перед знаком большего узлового числа. Например, число 242 можно записать в римской нумерации как *CCXL*II:

$$100 + 100 + (-10 + 50) + 1 + 1,$$

где $C = 100$;

$X = 10$;

$L = 50$;
I = 1.

Эта система для реализации в ЦА неудобна, так как количество цифр произвольно и правила выполнения операций над ними сложны.

В *позиционных* системах применяется конечный набор цифр, причем значение каждой из них зависит от позиции, занимаемой в числе. Количество различных цифр, применяемых в системе счисления, называется ее *основанием*. Произвольное число можно представить в виде суммы попарных произведений:

$$x = \sum_{p=1}^{c} a_p b^{k-p},$$

где c – число цифр в числе;
k – число цифр в целой части числа;
p – порядковый номер цифры слева направо в записи числа.

Например, в десятичной системе $x = 118{,}375$, $c = 6$, $k = 3$. Тогда $x = 1 \cdot 10^2 + 1 \cdot 10^1 + 8 \cdot 10^0 + 3 \cdot 10^{-1} + 7 \cdot 10^{-2} + 5 \cdot 10^{-3}$. Каждая позиция числа называется разрядом, и одна и та же значащая цифра имеет разный вес в зависимости от разряда, в котором она находится. В определении позиционной системы счисления нет ограничений на величину основания, поэтому могут существовать позиционные системы счисления с любым конечным числом цифр: двоичная, троичная и т. д.

Чтобы отличить числа, записанные в разных системах счисления, обычно после записи числа нижним индексом помечают основание системы. Например, $101{,}1_2 = 5{,}5_{10}$ или $201{,}2 = 19{,}7_{10}$.

Выбор системы счисления для ЦА осуществляется с учетом следующих позиций:

- простота технической реализации запоминающего элемента, который мог бы хранить любую из цифр системы счисления с заданным основанием;
- помехоустойчивость кодирования цифр на носителях информации;
- минимум затрат оборудования при построении узлов и блоков ЭВМ;
- простота арифметических действий и, следовательно, легкость реализации управления;
- наибольшее быстродействие в выполнении операций в секунду;
- возможность использования формального аппарата для анализа и синтеза ЭВМ;
- удобство работы человека и т. д.

Исходя из этих позиций рассмотрим различные системы счисления.

По критерию простоты технической реализации преимущество имеет двоичная система, так как две позиции легче удерживать и существует много простых технических реализаций для двухпозиционного элемента: электромеханическое реле (если контакт замкнут, то оно хранит 1, а в противном случае – 0);

транзистор (открыт – 0, закрыт – 1); магнитный материал (намагничен – 1, размагничен – 0) и т. д.

В современной электронной технике широко используется элемент, который называется *триггером*, для устойчивого хранения поступившего на его вход сигнала 0 или 1. Для хранения p-разрядного двоичного числа требуется p триггеров, из которых формируется так называемый регистр.

В двоичной системе счисления очень легко реализуются арифметические операции (практически все их можно свести к сложению; например, при умножении числа сдвигаются, а затем складываются). В основу *арифметико-логического устройства* (АЛУ) любой ЭВМ может быть положен сумматор.

Двоичная система счисления имеет и соответствующий формальный аппарат для анализа и синтеза вычислительных устройств на основе алгебры логики. Поэтому основной системой счисления для большинства ЭВМ стала двоичная, хотя были редкие попытки использовать и другие системы счисления, в частности, троичную и десятичную.

Методы перевода из одной системы счисления в другую позволяют обеспечить как представление информации в машине, так и отображение ее для человека.

В выбранной системе счисления в ЦА используются различные формы представления чисел (*автоматное или машинное изображение числа* в разрядной сетке ЦА).

Естественной формой представления числа называется его запись в порядке следования цифр с отделением запятой целой части числа от дробной:

$$X = X_1X_2 \ldots X_c, X_{c+1} \ldots X_p.$$

В ряде случаев удобно пользоваться *нормальной формой* представления числа, когда оно записывается как правильная дробь с фиксацией запятой перед первым значащим разрядом и умножается на соответствующую степень основания, чтобы получить его истинное значение, т. е.

$$X = 0, X_1X_2 \ldots X_p \cdot 10^a,$$

где a – порядок нормального числа;

$a = c$ – число цифр числа в целой части;

$a = -c$ – число нулей перед значащей цифрой дробной части числа, если целой части нет;

$X_1X_2 \ldots X_p$ – мантисса.

При записи порядок и мантисса представляются в естественной форме.

Приведем примеры нормальной записи чисел. Чтобы отличать порядок от мантиссы, будем записывать его вместе со знаком в круглых скобках после мантиссы. Пусть X = 118,375 – десятичное число, Y = 1110110,011 – двоичное число и T = 0,0175 – правильная десятичная дробь, тогда они могут быть пред-

ставлены в нормальной форме следующим образом: $X = 118{,}375 = 0{,}118375 \cdot 10^3 = 118375\ (+3)$, $Y = 1110110{,}011 = 0{,}1110110011 \cdot 10^{111} = 1110110011\ (+111)$, $T = 0{,}0175 = 0{,}175 \cdot 10^{-1} = 0{,}175\ (-1)$.

Так как в естественной форме положение запятой строго фиксируется между целой и дробной частями, то при машинной записи таких чисел при фиксированной разрядной сетке диапазон представления чисел будет колебаться от самого большого числа, которое играет роль «бесконечности» в машинном представлении, до самого маленького, начиная с которого последующие числа будут восприниматься как машинный нуль.

Например, при выделении в восьмиразрядном двоичном числе одного разряда под знак + (0) или – (1) и пяти разрядов под целую часть самым большим числом по модулю будет $+11111{,}11 = 2^4 + 2^3 + 2^2 + 2^1 + 2^0 + 2^{-1} + 2^{-2} = 16 + 8 + 4 + 2 + 1 + 0 + 0{,}5 + 0{,}25 = 31{,}75$, а самым маленьким – 0. Числа меньше |0,01| будут также восприниматься при вводе и выполнении машинных операций как нуль, а числа большие |31,75| будут давать переполнение разрядной сетки.

Машины, в которых фиксируется запятая после некоторого разряда при представлении чисел, называются *машинами с фиксированной запятой*. Их недостатки мы уже частично увидели из рассмотренных примеров.

При представлении чисел в нормальной форме положение запятой определяется каждый раз величиной порядка и его знаком. Поэтому запятая как бы перемещается на заданную позицию (плавает). Машины, использующие такой принцип представления чисел, называются *машинами с плавающей запятой*.

В современных ЭВМ используются обе формы представления чисел в зависимости от характера решаемых задач.

1.2 Конечные автоматы

Типы цифровых автоматов. Неформальное описание автоматов выглядит следующим образом: автомат имеет входную ленту, управляющее устройство с конечной памятью для хранения номера состояния, а также может иметь вспомогательную (рабочую) и выходную ленты.

Существует два типа автоматов:

- распознаватели-автоматы без выхода, которые распознают, принадлежит ли входная цепочка заданному множеству L;

- преобразователи-автоматы с выходом, которые преобразуют входную цепочку x в цепочку y при условии, что $x \in L$.

Входную ленту можно рассматривать как линейную последовательность ячеек, причем каждая ячейка может хранить один символ из некоторого конечного входного алфавита. Лента автомата бесконечна, но занято на ней в каждый момент времени только конечное число ячеек. Граничные слева и справа от занятой области ячейки могут занимать специальные концевые маркеры. Маркер может стоять только на одном конце ленты или отсутствует вообще.

Входная головка в каждый момент времени читает (обозревает) одну ячейку входной ленты. За один такт работы автомата входная головка может

сдвинуться на одну ячейку вправо или остаться на месте, при этом она выполняет только чтение, т. е. в ходе работы автомата символы в ячейках входной ленты не меняются.

Рабочая лента – это вспомогательное хранилище информации. Данные с нее могут читаться автоматом, а также записываться на нее.

Управляющее устройство – это программа, управляющая поведением автомата. Оно представляет собой конечное множество состояний вместе с отображением, описывающим, как меняются состояния в соответствии с текущим входным символом, читаемым входной головкой, и текущей информацией, извлеченной с рабочей ленты. Управляющее устройство также определяет направление сдвига рабочей головки и то, какую информацию записать на рабочую ленту.

Автомат работает, выполняя некоторую последовательность рабочих тактов. В начале такта читается входной символ и исследуется информация на рабочей ленте. Затем, в зависимости от прочитанной информации и текущего состояния, определяются действия автомата:

а) входная головка сдвигается вправо или стоит на месте;

б) на рабочую ленту записывается некоторая информация;

в) изменяется состояние управляющего устройства;

г) на выходную ленту (если она есть) записывается символ.

Поведение автомата удобно описывать в терминах *конфигурации автомата*, которая включает в себя:

а) состояние управляющего устройства;

б) содержимое входной ленты с положением входной головки;

в) содержимое рабочей ленты вместе с положением рабочей головки;

г) содержимое выходной ленты, если она есть.

Управляющее устройство может быть *недетерминированным*, когда для каждой конфигурации существует конечное множество возможных следующих тактов, любой из которых автомат может сделать исходя из этой конфигурации. Управляющее устройство будет *детерминированным*, если для каждой конфигурации будет возможен только один следующий такт.

Существуют следующие типы автоматов:

а) машины Тьюринга и Поста;

б) линейно-ограниченный автомат;

в) автомат с магазинной памятью (МП-автомат);

г) конечный автомат.

Сложность рабочей ленты определяет сложность автомата. Так, например:

а) машины Тьюринга и Поста имеют неограниченную в обе стороны ленту;

б) у линейно-ограниченного автомата длина рабочей ленты представляет собой линейную функцию длины входной цепочки;

в) у МП-автомата рабочая лента работает по принципу «последний пришел – первый вышел» (*LIFO − Last In First Out*);

г) у конечного автомата рабочая лента отсутствует.

Дадим формальное определение автомата. *Неинициальный автомат* – это пятерка вида

$$A = (Q, X, Y, \delta, \gamma),$$

где Q – множество состояний (алфавит состояний);
X – входной алфавит;
Y – выходной алфавит;
δ – функция переходов (отображение $Q \times X \to Q$);
γ – функция выходов (отображение $Q \times X \to Y$).

Функционирование автомата можно задать множеством команд вида

$qx \to py$, где $q, p \in Q, x \in X, y \in Y$.

Пусть на некотором такте t_i управляющее устройство находится в состоянии q, а из входной ленты читается символ x. Если в множестве команд есть команда $qx \to py$, то на такте t_i на выходную ленту записывается символ y, а к следующему такту $t_i + 1$ управляющее устройство перейдет в состояние p, т. е.

$$y(t) = y(q(t), x(t)), q(t + 1) = \delta(q(t), x(t)).$$

Если же команда $qx \to py$ отсутствует, то автомат оказывается блокированным и не реагирует на символ, принятый в момент t_i, а также перестает воспринимать символы в последующие моменты времени.

В соответствии с определением неинициального автомата в начальный момент состояние автомата может быть произвольным.

Если зафиксировано некоторое начальное состояние $q_0 = q(0)$, то такой автомат называют *инициальным*, т. е.

$$q(0) = q_0.$$

Рассмотрим множества $Q = \{Q_1, Q_2, ..., Q_k\}$ и $X = \{X_1, X_2, ..., X_r\}$, состоящие из k и r символов соответственно.

Согласно определению конечного автомата состояние Q^i на любом такте i однозначно определяется состоянием Q^{i-1} в предыдущий такт $(i – 1)$ и входом X^i в рассматриваемый такт i, т. е. $Q^i = \delta (Q^{i-1}, X^i)$, где моменты времени, к которым относятся символы Q и X, обозначены верхним индексом. Очевидно, эта формула определяет собой рекуррентное соотношение, так как при известных X^1 и Q^0 можно отыскать Q^1, приняв $i = 1$. Далее по Q^1 и X^2 можно отыскать Q^2 и т. д. для всех элементов множеств Q и X.

Зафиксируем переменную X, положив ее равной некоторому $X_q \in X$. Конечный автомат, описываемый таким рекуррентным соотношением $Q^i = \delta(Q^{i-1}, X_c)$, назовем *автономным*.

Фиксируя разные символы из множества X, получим r автономных автоматов. Для задания r автономных автоматов достаточно описать поведение конечного автомата. Автомат будем считать полностью заданным, если известны X, Q и $\delta(Q^{i-1}, X_q)$. Функцию δ можно задавать различными способами, чаще всего используются таблицы. Таблица функций переходов состояний строится следующим образом: вычерчивается прямоугольник, содержащий $(r + 1)\cdot(\kappa + 1)$ клеток. В первую клетку в верхнем левом углу прямоугольника соответственно записываются в верхней и нижней частях ее наименования координат X_i, Q^{i-1}. Вправо от нее по строке поклеточно вписываются все возможные значения X, а по столбцу вниз – все возможные значения Q. На пересечении столбца X и строки Q вписываются соответствующие значения $\delta(Q^{i-1}, X^i)$ при $X^i = X$ и $Q^{i-1} = Q_m$. В итоге получим таблицу, которую обычно называют основной таблицей конечного автомата (таблица 1.1).

Таблица 1.1 – Основная таблица конечного автомата

X^i / Q^{i-1}	X_1	X_2	...	X_r
Q_1	Q_k	Q_1	…	Q_5
Q_2	Q_3	Q_2	…	Q_4
…	…	…	…	…
Q_r	Q_3	Q_3	…	Q_6

Пусть $X^i = P_2$, а $Q^{i-1} = Q_k$. Тогда искомое значение функции $\delta(Q_r, P_2)$ отыскивается на пересечении строки Q_k и столбца P_2 (равно Q_3).

Можно заметить, что каждый из столбцов P_1, P_2, ..., P_r этой таблицы является основной таблицей автономного автомата. Такие автоматы можно задавать с помощью *ориентированных графов* так, чтобы наблюдалось взаимнооднозначное соответствие вершин графа и символов из алфавита Q. Дуга (ориентированное ребро) будет соответствовать переходу автономного автомата из состояния Q_i в Q_j (рисунок 1.1).

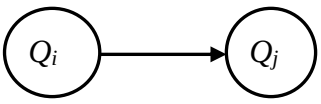


Рисунок 1.1 – Элемент задания конечного автомата автономными переходами

Такая совокупность кружков и стрелок (граф) полностью опишет функцию δ при соответствующем фиксированном P.

Покажем на примере небольшой таблицы, как ее описать на языке графов. Пусть таблица 1.2 задает конечный автомат A с двумя входами и четырьмя состояниями.

Таблица 1.2 – Основная таблица автомата A

Q^{i-1} \ X^i	X_1	X_2
Q_1	Q_2	Q_3
Q_2	Q_1	Q_2
Q_3	Q_4	Q_3
Q_4	Q_2	Q_1

Чтобы различить для какого фиксированного входа рассматривается автономный автомат, над каждым из графов поставим наименования соответствующих входов.

Исходная таблица может быть представлена двумя графами $A(X_1)$ и $A(X_2)$, как это изображено на рисунке 1.2.

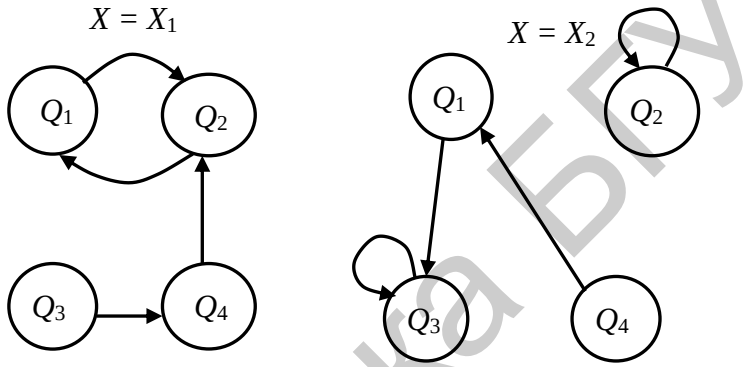


Рисунок 1.2 – Графы, соответствующие таблице автомата

Для большей компактности таблицу 1.2 можно представить одним объединенным графом, который называется *диаграммой состояний* конечного автомата. Как видно из рисунка 1.3, на объединенном графе над каждой стрелкой записывается тот вход X, который вызывает переход из исходного состояния в то, к которому направлена стрелка.

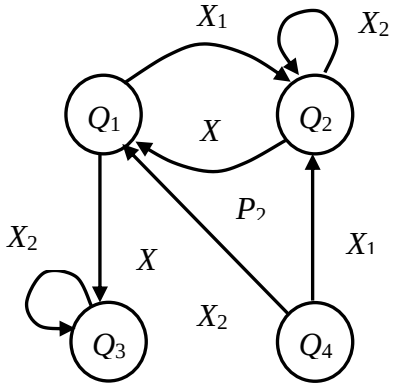


Рисунок 1.3 – Объединенный граф автомата

Диаграмма состояний автомата весьма наглядна. Допустим, необходимо установить, каким будет внутреннее состояние автомата A на третьем такте при

входной последовательности $X = \{X_1, X_1, X_2\}$ и начальном состоянии $Q^0 = Q_3$. Решение задачи отыскивается простым обходом от начального состояния $Q^0 = Q_3$ по стрелке X_1 к следующему состоянию Q_4, от него снова по стрелке X_1 к состоянию Q_2, а от Q_2 по стрелке X_2 опять возвращаемся в состояние Q_2, оно и будет искомым.

В ряде случаев желательно выполнять подобные операции более формальным путем, чтобы использовать вычислительные средства для отыскивания нужного результата. Для этой цели удобно матричное задание конечного автомата.

Графу X_c автономного автомата можно поставить во взаимно однозначное соответствие квадратную матрицу, состоящую из нулей и единиц. Пронумеруем все возможные состояния Q цифрами от 1 до $к$. Составим матрицу размером $к \times к$, такую, что в строке i на месте j будем записывать 1, если существует дуга (стрелка) от i к j на графе A, а в противном случае – 0. Обозначим эту матрицу через $A(X_c) = (a_{ij}(X_c))$.

При матричном задании конечного автомата начальное состояние Q^0 также задается матрицей $A(X_0)$, состоящей из одной строки, в которой записываются $(к - 1)$ нулей, а на месте j с номером, соответствующим номеру Q^0, – единица.

Пусть заданы входная последовательность $X = \{X^1, X^2, \ldots X^i\}$ и начальное состояние Q^0 матрицей $A(X_0)$. Тогда состояние автомата на такте i устанавливается как результат вычисления произведения:

$$\prod_{t=0}^{i} A(X^t) = A_i(X^i),$$

где $A_i(X^i)$ – матрица, состоящая из одной строки с единицей на месте, соответствующем номеру искомого состояния.

Проиллюстрируем вышесказанное на примере автомата A. Графам X_1 и X_2 будут соответствовать матрицы

$$A(X_1) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix} \text{ и } A(X_2) = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix},$$

а начальному состоянию $Q^0 = Q_3$ – матрица $A(X^0) = (0010)$. Тогда, используя формулу, получим искомый результат: $A_3(X^3) = (0100)$.

Система, состоящая из конечного автомата A, преобразующего символы алфавита X в символы алфавита Q в соответствии с функцией переходов $Q^i = \delta(Q^{i-1}, X^i)$ и функцией выходов γ, который мгновенно и однозначно ставит в соответствие каждому символу Q символ из некоторого алфавита Y, т. е. $Y^i = \gamma(X^i, Q^i)$, называется *конечным автоматом с выходным преобразователем*.

Если вычисления выполнять последовательно и фиксировать промежуточные результаты на ленте бумаги, то полученная запись даст ленту последовательного изменения состояний автомата.

Автоматы Мили и Мура. На практике наибольшее распространение получили два класса автоматов – автоматы Мили и Мура.

Закон функционирования автомата Мили задается уравнениями:

$$q(t+1) = \delta\,(q\,(t), x(t)); \;\; y(t) = \gamma\,(q\,(t), x\,(t)), \; t = 0, 1, 2, \ldots .$$

Закон функционирования автомата Мура задается уравнениями:

$$q\,(t+1) = \delta\,(q\,(t), x\,(t)); \;\; y\,(t) = \gamma\,(q\,(t)), \; t = 0, 1, 2, \ldots .$$

Из сравнения законов функционирования видно, что, в отличие от автомата Мили, выходной сигнал в автомате Мура зависит только от текущего состояния автомата и в явном виде не зависит от входного сигнала. Для полного задания автомата Мили или Мура дополнительно к законам функционирования необходимо указать начальное состояние и определить внутренний, входной и выходной алфавиты. Между автоматами Мили и Мура существует соответствие, позволяющее преобразовать закон функционирования одного из них в другой, и обратно. Автомат Мура можно рассматривать как частный случай автомата Мили, имея в виду, что последовательность состояний выходов автомата Мили опережает на один такт последовательность состояний выходов автомата Мура, т. е различие между автоматами Мили и Мура состоит в том, что в автоматах Мили состояние выхода возникает одновременно с вызывающим его состоянием входа, а в автоматах Мура – с задержкой на один такт, так как в автоматах Мура входные сигналы изменяют только состояние автомата.

При представлении автомата Мура графом дуги помечаются символами входного алфавита, а каждая вершина графа – состоянием и символом выходного алфавита. При формальном сравнении определений автоматов Мили и Мура может показаться, что автомат Мура может быть задан как входонезависимый автомат Мили. Однако это не соответствует способу функционирования автоматов Мура в соответствии с введенным определением. В автомате Мура реализована иная временная связь между переходами из одного состояния в другое и выходом, по сравнению с автоматом Мили, у которого выход, соответствующий некоторому входу и определенному состоянию, формируется во время перехода автомата в следующее состояние. У автомата Мура сначала формируется выход, а потом – переход в следующее состояние, причем выход определяется только состоянием автомата.

Комбинационные автоматы. Автомат, значения выхода Y которого зависят только от значений входов X в данный момент времени и не зависят от входных воздействий в предшествующие моменты времени, называется *комбинационным*, или *автоматом без памяти*. В структурных моделях автоматы

этого класса называются комбинационными схемами. Их функционирование описывается математической моделью вида $Y = f(X)$. Функция, описывающая это соотношение для одного выхода, называется *булевой* или *логической функцией*. Такие функции задаются с помощью полностью или не полностью определенных таблиц, которые называются таблицами истинности и таблицами решений соответственно. Более компактной формой представления является задание булевых функций в виде булевых формул, которые в отличие от таблиц всегда определены на всех 2^n наборах (n – число входных переменных). В таблице 1.3 представлены примеры функций двух переменных x_1 и x_2.

Таблица 1.3 – Логические функции

Комбинация аргументов		Функция И, $y = x_2 \wedge x_1$	Функция ИЛИ, $y = x_2 \vee x_1$	Функция И–НЕ, $y = \wedge(x_2 \wedge x_1)$	Функция ИЛИ–НЕ, $y = \wedge(x_2 \vee x_1)$	Функция исключающее ИЛИ, $y = x_2 \oplus x_1$
x_2	x_1					
0	0	0	0	1	1	0
0	1	0	1	1	0	1
1	0	0	1	1	0	1
1	1	1	1	0	0	0

При представлении логической функции математическим выражением используют два вида ее представления.

Совершенной дизъюнктивной нормальной формой (СДНФ) называется логическая сумма элементарных логических произведений, в каждое из которых аргумент или его отрицание входят один раз. СДНФ может быть получена из таблицы истинности следующим образом: для каждого набора аргументов, на котором функция равна единице, записывают элементарные произведения переменных, причем переменные, значение которых равно нулю, записывают с инверсией ($\wedge x$). Полученные произведения, которые носят название конституент единицы, суммируют. Далее по законам алгебры логики это выражение минимизируется с целью сокращения в нем как количества элементарных произведений, так и некоторых переменных в них, что упрощает техническую реализацию комбинационных автоматов.

Совершенной конъюнктивной нормальной формой (СКНФ) называется логическое произведение элементарных сумм, в каждую из которых аргумент или его отрицание входят один раз. СКНФ может быть получена из таблицы истинности: для каждого набора аргументов, на котором функция равна нулю, составляют элементарную сумму, причем переменные, значение которых равно единице, записываются с отрицанием. Полученные суммы, которые носят название конституент нуля, объединяют операцией логического умножения, СКНФ далее по законам алгебры логики минимизируется.

Задание булевых функций в виде графа переходов нерационально, так как для этого класса автоматов отсутствует необходимость в отражении динамики переходов.

Машины Поста и Тьюринга часто используются в различных теоретических построениях для доказательства общих утверждений, касающихся возможностей вычислительных машин при реализации алгоритмов. Однако на этих абстрактных машинах можно вести и обучение основам программирования на бумаге школьников и студентов.

Компьютерные модели. *Компьютерное моделирование* – метод решения задачи анализа или синтеза сложной системы на основе использования ее компьютерной модели. Его суть – получение качественных и количественных результатов, стоящих перед исследователем в соответствии с постановкой задачи. Сложность вопроса состоит в том, что степень полезности результатов во многом зависит от возможностей информационного наполнения модели, когда часть данных еще не получена по тем или иным причинам (еще не завершены научные исследования и эксперименты, данные закрыты от исследователя и т. д.).

Однако по компьютерной модели проще и удобнее проводить вычислительные эксперименты, часть из которых в натуре иногда слишком дорога или неосуществима вообще (угроза взрыва, потеря объекта и т. п.). На модели легче проверить реакцию изучаемой системы на изменение ее параметров.

К основным этапам компьютерного моделирования относят:

- постановку задачи (иногда это более половины успеха дела);
- определение границ объекта моделирования;
- определение процессов взаимодействия между частями модели;
- формализацию элементов модели;
- разработку алгоритма воспроизведения процесса на модели;
- написание программы (подбор подходящей готовой системы и ее адаптация;
- проведение экспериментов и фиксацию их результатов;
- анализ и интерпретацию итогов экспериментов.

Иногда полезно различать аналитическое и имитационное моделирование. Под *аналитическим моделированием* понимают разработку модели реального объекта путем описания его средствами математики (на основе подбора различных функций и их параметров, связей между ними и т. д.). *Имитационное моделирование* обычно сводится к построению значительного количества функций для описания отдельных элементов процесса. Его результатом является многократное воспроизведение взаимодействий между элементами модели с последующим получением результата на основе усредненных показателей изучаемого явления в целом.

При создании моделей на практике необходимо соблюдать ряд наиболее важных принципов:

- гарантия осуществимости реализации модели с позицией достаточности информации для ее функционирования;
- возможность достижения поставленной цели за заданное время;
- допустимость модификации модели по мере уточнения информации и результатов;
- использование максимально возможного набора стандартных средств, функций, программ.

Математические модели всего объекта и его частей выражают некоторые его существенные черты на языке уравнений и формул. Современные компьютеры при математическом моделировании используются не только для численных расчетов, но и для построения графиков, воспроизведения звуков и аналитических расчетов. Например, *Mathcad* и другие аналогичные системы являются хорошими готовыми средствами для таких целей. Эти новые возможности открывают пути к более широкому пониманию и использованию современных компьютерных технологий. В частности, использование графики и речи позволяет воспроизводить более сложные процессы. Например, имитацию работы технических устройств с участием человека, генерацию моделей на основе интеллектуальных банков данных.

Компьютерная модель является программной реализацией математической модели (абстрактной знаковой модели), которая частично отражает исходный объект моделирования, исходя из целей его исследования, т. е. математическая модель ограничена отражением только необходимых свойств объекта, которые влияют достаточно адекватно на исследуемый процесс. В этом смысле знаковые модели могут быть довольно схематичными (план цеха в виде чертежа для решения задачи размещения оборудования и т. п.), что упрощает моделирование задаваемого процесса. Процесс исследования на модели завершается получением параметров, соответствующих заданному критерию с необходимой точностью. *Его обеспечение предполагает выполнение некоторых стандартных шагов*:

- выделение существенных свойств для объекта моделирования с позиций, предполагаемых для решения задач, и построение математической модели;
- построение и отладка компьютерной модели;
- оценка соответствия модели для решения класса задач;
- адаптация модели на предмет обеспечения полноты класса решаемых задач.

Следует отметить важную роль реализации случайных механизмов в решении сложных задач. Когда о поведении функции недостаточно информации, можно с равной вероятностью выбирать любое значение для назначения координаты начальной точки дискретной переменной или выбирать ее из заданного отрезка для непрерывной переменной. Такой прием позволяет в какой-то мере при старте из нескольких попыток повысить шансы избежать попадания в локальный экстремум. Вероятностные приемы могут быть и более сложными.

Вероятностное моделирование на ЭВМ обычно опирается на использование генераторов случайных чисел, равномерно распределенных в диапазоне (0, 1) или (0, a), которые содержатся в большинстве вычислительных систем (ВС) в виде стандартных процедур. На их основе обычно строятся случайные процессы для воспроизведения элементов моделей в виде конкретных случайных законов (экспоненциальный, нормальный и др.).

Второй путь использования генератора случайных чисел состоит в выборе случайных условий или путей продолжения процессов, когда требуется исключить влияние исследователя на выбор текущего продолжения процесса, например, стартовых точек для различных итерационных процессов вычислений для многомерных функций и т. д.

1.3 Описания алгоритмов

Интуитивное определение алгоритма. Существует несколько толкований термина «алгоритм». Вообще, под алгоритмом часто понимается некоторое формальное предписание действий, согласно которому можно получить нужное решение задачи. Обычно процесс подготовки решения задачи называется *алгоритмизацией*. Он начинается со словесной формулировки постановки задачи. Затем с помощью формул, таблиц, схем и т. п. описывается метод решения задачи. В соответствии с предлагаемым методом выбирается алгоритм решения.

Вычислительная машина, вообще говоря, перерабатывает входную последовательность знаков в выходную. Операции над этими последовательностями практически сводятся к применению ряда подстановок, которые позволяют выполнять различные преобразования:

- кодирование информации в вид, удобный для восприятия машиной;
- переработку информации по некоторому алгоритму;
- воспроизведение результата в удобной для человека форме.

При переработке информации рассматриваются конечные и бесконечные последовательности различных символов (букв). Элементы конечных последовательностей нумеруются легко, но что значит «рассмотреть» бесконечную последовательность?

Представление о такой последовательности можно составить лишь по описанию тех или иных ее свойств. Например, определим по номеру места символ в бесконечной последовательности 01 01 01 ... . Если рассматривать только начало последовательности, то никаких представлений о ней в целом сложить нельзя. Однако представление создается, если указывается, что 0 и 1 чередуются всегда. Теперь можно предсказать, какой символ, например, стоит на 93-м и 1026-м месте.

В этом примере свойство, благодаря которому имелась возможность составить представление обо всей бесконечной последовательности, заключалось в существовании процедуры, позволяющей узнавать по номеру места цифру, стоящую на этом месте. Точнее можно сказать, что для этого примера имелся алгоритм распознавания символа по номеру места.

Классическим примером является алгоритм Евклида для нахождения наибольшего общего делителя положительных чисел a и b. Алгоритм состоит из следующей системы последовательных указаний:

1 Задать числа a и b. Перейти к указанию 2. ПКУ2.

2 Сравнить числа ($a = b$ или $a <$ b или $a > b$). ПКУ3.

3 Если числа равны, то каждое из них дает искомый результат и процесс вычислений останавливается, иначе ПКУ4.

4 Если первое рассматриваемое число меньше второго, то нужно переставить их местами. ПКУ5.

5 Вычесть второе число из первого и рассматривать два числа: вычитаемое (a) и остаток (b). ПКУ2 (процесс повторяется по вышеназванной схеме).

Рассмотрим пример, иллюстрирующий применение алгоритма Евклида для чисел $a = 15$ и $b = 20$, указывая результаты и номера выполняемых операций:

1 15, 20. ПКУ2.
2 15 < 20. ПКУ3.
3 ПКУ4.
4 20, 15. ПКУ5.
5 20 − 15 = 5. ПКУ2.
2 15 > 5. ПКУ3.
3 ПКУ4.
4 ПКУ5.
5 15 − 5 = 10. ПКУ2.
2 5 < 10. ПКУ3.
3 ПКУ4.
4 10, 5. ПКУ5.
5 10 − 5 = 5. ПКУ2.
2 5 = 5. ПКУ3.
3 Остановка процесса, результат 5.

По определению акад. А. Н. Колмогорова, алгоритм или алгорифм – это всякая система вычислений, выполняемых по строго определенным правилам, которая после какого-либо числа шагов заведомо приводит к решению поставленной задачи. В инженерной практике часто используется следующее определение: алгоритм – конечная совокупность точно сформулированных правил решения какой-то задачи.

Алгоритмы, в соответствии с которыми решение поставленных задач сводится к арифметическим действиям, называются численными. Численными являются также любые алгоритмы для решения некоторого класса задач, если формулами полностью описан как состав действий, так и порядок их выполнения.

Указания, задающие элементарные действия, называются операторами. Реализация оператора сводится к переработке некоторой информации и к определению оператора, выполняемого вслед за данным.

Понятие элементарного действия (операции) существенно зависит от способа (метода) реализации алгоритма. Если алгоритм рассчитан на выполнение человеком, то ему может соответствовать совершенно другой уровень описания, чем для ЭВМ. Чем ниже будет квалификация исполнителя алгоритма или же оснащенность вычислительной машины, тем детальнее необходимо задавать алгоритм. Каждый шаг алгоритма должен описываться строго однозначно.

Наряду с арифметическими операциями над числами часто встречаются различные логические операции. Алгоритмы с преимущественным использованием этих операций называют логическими.

Классическим примером логического алгоритма является задача поиска пути в лабиринте. Лабиринт задается в виде конечной системы площадок, от которых расходятся коридоры. Каждый коридор соединяет только две смежные площадки. Геометрически лабиринт можно представить в виде кружков, соединенных отрезками (граф).

Коридору будет соответствовать ребро графа, а площадке – вершина. Будем говорить, что площадка *Y* достижима с площадки *X*, если существует путь, ведущий от *X* к *Y* через промежуточные коридоры и площадки. Очевидно, что если площадка *Y* вообще достижима с площадки *X*, то она достижима посредством простого пути, когда каждую из площадок следует проходить один раз (рисунок 1.4). Например, простой путь с площадки *A* в *D*: *ABD* или *ABCD*; площадка *K* с площадки *A* вообще недостижима.

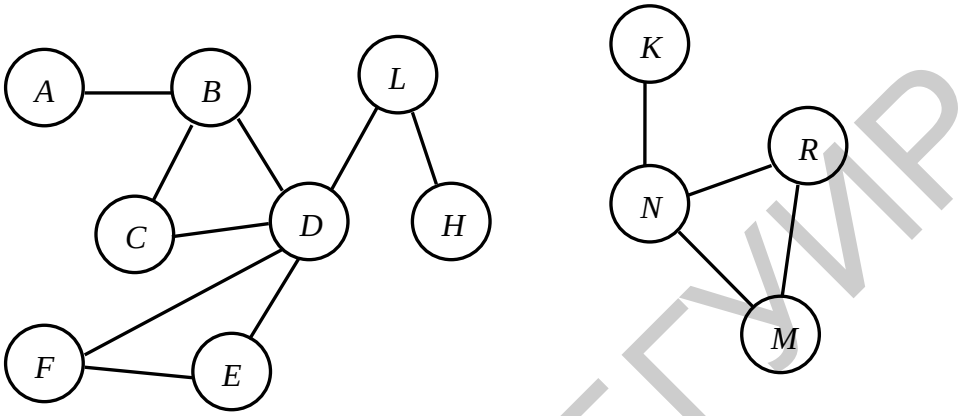


Рисунок 1.4 – Модель лабиринта

Решим задачу. Начав поиск с площадки *A*, выяснить, достижима ли произвольная площадка *Y*. Если она достижима, то найти путь из *A* в *Y*, иначе после поиска вернуться в *A*.

Устройство лабиринта заранее неизвестно. Под решением задачи понимается указание общего метода поиска для любого расположения площадок *A* и *Y*. Предположим, что в начале и конце коридоров можно делать пометки, проходился ли данный коридор и сколько раз. Коридоры, которые не проходились ни разу, будут без пометок, пройденным один раз будем присваивать метку 1, а пройденным дважды – метку 2.

Находясь на какой-либо площадке, можно попасть на смежную площадку двумя путями:

- прохождением по неотмеченному коридору, пометив его начало и конец знаком 1;

- возвратом с данной площадки по коридору с меткой 1, заменив метку коридора знаком 2.

Находясь на какой-либо площадке, можно различать следующие признаки:

а) это площадка *Y* (цель достигнута);

б) это петля (от данной площадки расходятся по крайней мере два коридора с меткой 1, и неотмеченных коридоров нет);

в) от данной площадки отходит по крайней мере один коридор без меток;

г) это исходная площадка *A*;

д) отсутствие всех предыдущих признаков.

В соответствии с этими признаками ходы делаются до тех пор, пока не наступит остановка (таблица 1.4).

Таблица 1.4 – Метод поиска пути из *A* в *Y*

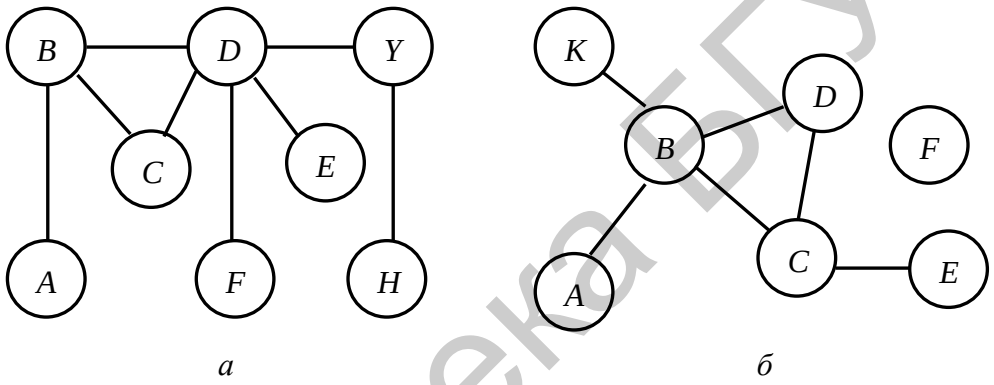
Признак	Ход
Площадка *Y*	Остановка
Петля	Возврат по последнему пройденному коридору
Имеется коридор без пометок	Движение по одному из коридоров без пометок
Площадка *A*	Остановка
Отсутствие признаков а–д	Возврат по пройденному коридору

Относительно рассматриваемого метода справедливы следующие утверждения:

а) при любом расположении *A* и *Y* за конечное число ходов наступит остановка либо в *A*, либо в *Y*;

б) если остановка на площадке *A*, то площадка *Y* недостижима.

Рассмотрим два небольших примера (рисунок 1.5).



а *б*

а – с достижимыми площадками; *б* – с недостижимыми площадками

Рисунок 1.5 – Лабиринты

Шаги процесса поиска достижимой площадки *F* в лабиринте, показанном на рисунке 1.5, *а*, представлены в таблице 1.5, а недостижимость площадки *F* для лабиринта, изображенного на рисунке 1.5, *б* – в таблице 1.6.

Этот метод поиска содержит элемент произвола, которого не было в ранее рассмотренных примерах. Например, в алгоритме Евклида операции, сделанные разными вычислителями, совпадают во всех деталях. Здесь же от одной площадки отходит несколько коридоров, и выбор можно делать произвольно. Алгоритмом обычно называют строго детерминированную систему. Поэтому дополним алгоритм требованием выбирать первый коридор по часовой стрелке при возникновении неоднозначного продолжения.

Таблица 1.5 – Поиск достижимой площадки *F*

Номер хода	Признак	Направление движения	Метка коридора
1	–	*A*	–
2	в	*BC*	1
3	в	*CD*	1
4	в	*DУ*	1
5	в	*УH*	1
6	д	*HY*	2
7	д	*УD*	2
8	в	*DB*	1
9	б	*BD*	2
10	в	*DF*	1
11	–	Остановка	–
12	–	*A*	1

Таблица 1.6 – Недостижимость площадки *F*

Номер хода	Признак	Направление движения	Метка коридора
1	в	*AB*	1
2	в	*BC*	1
3	в	*CE*	1
4	д	*EC*	2
5	в	*CD*	1
6	в	*DB*	1
7	б	*BD*	2
8	б	*DC*	2
9	б	*CB*	2
10	б	*BA*	2
11	г	Остановка	–

Общие свойства алгоритмов. Из рассмотренных примеров отчетливо выступают свойства, присущие любому алгоритму.

Дискретность алгоритма. Алгоритм – процесс последовательного построения величин, идущих в дискретном времени таким образом, что в начальный момент задается исходная конечная система величин, а в каждый следующий момент система величин строится по определенному закону из системы величин, имевшихся в предыдущий момент времени.

Детерминированность алгоритма. Система величин, получаемых в какой-то неначальный момент времени, однозначно определяется системой величин, полученных в предыдущие моменты времени.

Элементарность шагов алгоритма. Закон получения последующей системы величин из предыдущей должен быть простым и локальным.

Направленность алгоритма. Если способ получения последующей величины из предыдущей не дает результата, должно быть указано, что считать результатом.

Массовость алгоритма. Начальная система величин может выбираться из некоторого бесконечного множества. Иными словами, алгоритм служит для решения целого класса задач.

Область применения. Областью применения алгоритма называется такая наибольшая область начальных данных, на которой алгоритм результативен.

Относительно перечисленных свойств алгоритма укажем их интерпретацию для алгоритма Евклида:

1 Дискретность. В любой момент по паре чисел (*a*, *b*) строится новая пара (*a*, *b*).

2 Детерминированность. Пара (*a*, *b*) определяется однозначно.

3 Элементарность шагов. Вычитание двух чисел, сравнение, перестановка.

4 Направленность. Указано правило прекращения процесса и то, что считается результатом выполнения каждого шага.

5 Массовость. Алгоритм применим к любой паре целых чисел $a > 0$, $b > 0$.

6 Область применения. $a, b \in \{1, 2, ...\}$.

Следует отметить, что число операций при выполнении того или иного алгоритма заранее неизвестно и зависит от выбора исходных данных. Поэтому под осуществимостью алгоритма следует понимать потенциально возможный процесс для конкретных задач из области его применения, так как для решения некоторых из них может не хватить ни времени, ни памяти.

Математическое определение алгоритма. Алгоритмами в современной математике принято называть конструктивно задаваемые соответствия между изображениями объектов в абстрактных алфавитах.

Абстрактным алфавитом называется любая конечная совокупность объектов, называемых буквами или символами данного алфавита. При этом природа этих объектов нас совершенно не интересует. Символом абстрактных алфавитов можно считать буквы алфавита какого-либо языка, цифры, любые значки и даже слова некоторого конкретного языка. Основным требованием к алфавиту является его конечность. Алфавит, как любое множество, задается перечислением его элементов.

Итак, объекты реального мира можно изображать словами в различных алфавитах. Это позволяет считать, что объектами работы алгоритмов могут быть только слова. Тогда можно сформулировать следующее определение.

Алгоритм есть четкая конечная система правил для преобразования слов из некоторого алфавита в слова из этого же алфавита.

Слово, к которому применяется алгоритм, называется *входным*.

Слово, вырабатываемое в результате применения алгоритма, называется *выходным*.

Совокупность слов, к которым применим данный алгоритм, называется *областью применимости* этого алгоритма.

Можно выделить три основных типа универсальных алгоритмических моделей, различающихся исходными эвристическими соображениями относительно того, что такое алгоритм.

Первый тип основан на функциональном подходе и рассматривает понятие алгоритма с точки зрения того, что можно вычислить с его помощью. Наиболее развитая и изученная модель этого типа – рекурсивная функция – является исторически первой формализацией понятия алгоритма.

Второй тип основан на представлении алгоритма как некоторого детерминированного устройства, способного выполнять в каждый отдельный момент некоторые примитивные операции, или инструкции. Основной теоретической моделью этого типа, созданной в 30-е годы XX века, является *машина Тьюринга*, которая представляет собой автоматную модель, в основе которой лежит анализ процесса выполнения алгоритма как совокупности набора инструкций.

Третий тип алгоритмических моделей – это преобразования слов в произвольных алфавитах, в которых элементарными операциями являются подстановки, т. е. замены части слова (подслова) другим словом. Примерами моделей этого типа являются *нормальные алгоритмы Маркова* и *канонические системы Поста*.

Всякий алгоритм отражает последовательность преобразований информации. Алгоритм обычно состоит из операционной и информационной частей.

Естественно, что при описании алгоритмов нужно задавать операторы и связи между ними. Как правило, оператор представляет собой довольно простую конструкцию, и поэтому наибольшие трудности при разработке алгоритма приходятся на указание связей между операторами, т. е. на описание структуры алгоритма.

При построении арифметических операторов предпочитают включать в каждый из них определенное законченное действие, например, вычисление определителя, извлечение корня и т. п. На этом этапе последовательность размещения операторов не рассматривается.

Для осуществления вычислений на ЭВМ важна очередность арифметических операторов, так как один из операторов может обеспечить возможность выполнения следующего. При построении описания вычислительной схемы алгоритма необходимо разместить арифметические операторы в такой последовательности, чтобы выполнение вычислений одного из них обеспечивало возможность реализации следующих за ним операторов. Естественно, что в частных случаях перестановка некоторых операторов местами допускается.

Процесс разработки алгоритма. Каждый разработчик алгоритма, руководствуясь накопленным опытом и знанием закономерностей алгоритмизуемого процесса, изображает его в соответствии со своими представлениями. Однако можно выделить некоторые типичные шаги этой работы и наиболее употребительные средства описания алгоритмов:

1 Разработка алгоритма начинается с изучения задания, данного для алгоритмизации. Оно часто представляется в описательной форме с использованием формул, таблиц, графиков и т. п. Перед разработчиками алгоритма возникает необходимость глубоко изучить алгоритмизуемый процесс, уяснить закономерности составляющих его явлений, установить все факторы, влияющие на его течение, оценить степень влияния отдельных явлений на окончательный результат и т. д.

Необходимо определить входную и выходную информацию, задать области изменения аргументов, точность вычислений.

2 Входная информация должна быть полной для обеспечения получения всей выходной информации. Входная информация бывает двух видов: постоянная и переменная. Постоянная входная информация сохраняет значение в процессе счета по всему алгоритму. Переменная информация зависит от конкретной частной задачи, решаемой в данный момент.

При разработке алгоритмов универсальность построенного алгоритма зависит от соотношения объемов переменной и постоянной входной информации. При уменьшении переменной информации, как правило, алгоритм становится менее универсальным.

3 Далее выполняется математическая формализация словесно-описательного условия задачи. Цель ее – построить массивы арифметических $\{A\}$ и логических $\{P\}$ операторов. В массив $\{P\}$ входят все условия, которые отражают закономерности алгоритмизуемого процесса. Массивы $\{A\}$ и $\{P\}$ являются тем материалом, из которого строится вычислительная схема алгоритма.

4 Совокупность данных, которые в соответствии с условиями задачи должны быть получены в результате ее решения, составляют выходную информацию.

Способы описания алгоритмов. В процессе разработки алгоритмов для их анализа, сокращения числа элементов, удобства программирования и других целей используется много способов описания алгоритмов. Среди них наибольшее распространение получили:

а) словесный (на естественном языке);

б) схемы алгоритмов (графический);

в) операторные схемы;

г) псевдокоды;

д) языки программирования;

е) таблицы.

Названные средства описания имеют определенные достоинства и недостатки. Одни из них более удобны на этапе разработки алгоритма, другие – на этапе анализа, третьи – на этапе минимизации алгоритма, четвертые – на этапе общения человека с ЭВМ и т. д.

Для иллюстрации средств записи алгоритмов приведем простейший пример. Пусть задана последовательность X_1, X_2, ..., X_i, ..., X_n из n произвольных действительных чисел и требуется подсчитать в этой последовательности количество чисел M для случая $X_i \geq a$ и количество чисел B для случая $X_i < a$.

Очевидно, что i может принимать только целые значения 1, 2, ..., n. Отдельные операторы алгоритма будем обозначать числами 1, 2, 3, ... Они будут играть роль меток (номеров) операторов. Тогда решение задачи можно описать в виде следующего алгоритма:

1 $B := 0$, $M : = 0$.

2 $i := 1$.

3 Если $X_i < a$, то ПКУ4, иначе ПКУ6.

4 $B := B + 1$.

5 ПКУ7.

6 $M := M + 1$.

7 Если $i < n$, то ПКУ8, иначе ПКУ10.

8 $i := i + 1$.

9 ПКУ3.

10 Конец.

Знак «:=» следует рассматривать как символ операции присваивания величине, стоящей слева от него, значения величины, стоящей справа.

Эта последовательность операций и представляет собой запись алгоритма для человека, знающего алгебру и правила выполнения арифметических действий. Однако в ней присутствуют «необычные» операторы 4, 6 и 8, которые бессмысленны для немашинной математики, поскольку в ней такие равенства, как $B = B + 1$ и другие, аналогичные ему, невозможны. Суть такой записи в том, что старое значение B увеличивается на единицу и новое значение присваивается снова величине с именем B. Это равносильно операции, когда результат заносится в заданную клетку таблицы, а вычисления проводятся на черновике. После вычисления нового результата на черновике старый результат в этой клетке таблицы стирается и на его месте записывается новый и т. д. Такая запись порожде-

на удобством для отображения процессов вычислений на машине, где роль таблицы выполняет память машины. В этом случае после вычислений машина направляет новый результат на заранее выбранный адрес памяти, что приводит к автоматическому стиранию старого результата и запоминанию нового.

Воспользуемся данным алгоритмом для иллюстрации различных общепринятых средств его записи. В принятой нами терминологии операторы 1, 2, 4, 6, 8 отнесем к арифметическим (группа ***A***), а 3, 5, 7, 9 – к логическим (группа ***P***). При необходимости подчеркнуть принадлежность оператора к одной из этих групп будем записывать A_1, A_2, A_4, A_6, A_8 и соответственно P_3, P_5, P_7, P_9.

Мы остановимся более детально на **операторной форме** представления алгоритмов и их схем, так как предварительная подготовка задачи для программирования крайне редко обходится без укрупненной разбивки на операторы и описания связей между ними на языке схем.

Сложность операторов определяется характером решаемой задачи. Чтобы после объединения операторов получилось описание алгоритмического процесса, необходимо каким-то образом описать взаимосвязь операторов.

Для удобства записи логических схем алгоритмов операторы обычно располагают в одну строку и руководствуются *следующими правилами*:

а) порядковый номер оператора в данной схеме изображается нижним индексом оператора. Нумерация операторов сквозная;

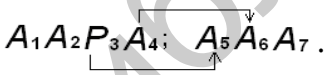
б) если оператор зависит от параметра, то этот параметр изображается верхним индексом при его буквенном обозначении;

в) если знаки двух операторов располагаются в схеме рядом, то оператор, стоящий в схеме слева, передает управление оператору, записанному справа;

г) если между двумя записанными рядом знаками операторов стоит точка с запятой, то от оператора, записанного слева, нет передачи управления оператору, записанному справа;

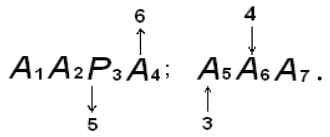
д) передача управления оператору, записанному не рядом справа, обозначается стрелкой.

Например, операторная схема может выглядеть так:



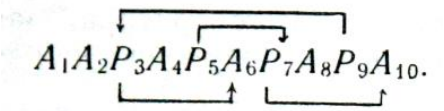
Для удобства записи горизонтальную часть стрелки можно опускать. При этом начало стрелки отмечается малой вертикальной стрелкой, направленной от оператора и снабженной номером оператора, которому передается управление с указанием номера, передающего управление оператора.

Тогда эту же схему можно представить так:



При начертании схем программ обычно принято стрелки от логических операторов изображать над строкой, если логическое условие истинно, и под строкой в противном случае.

В соответствии с указанными правилами приведенный здесь алгоритм в операторной форме можно записать так:



Во многих случаях бывает удобно группу элементарных операторов обозначать одной буквой. В этом случае придерживаются правила, что управление извне (от операторов, не принадлежащих данной группе) может получать лишь один элементарный оператор группы. Такую группу элементарных операторов называют *обобщенным оператором*.

С помощью операторной схемы алгоритма составляется схема решения задачи, которая дополняется операторами, свойственными машинному вычислительному процессу. Для каждого оператора в порядке его записи затем составляется своя частная программа. Совокупность этих частных программ и дает программу решения задачи. Использование операторной схемы алгоритма при программировании рассчитано на получение *следующих преимуществ*:

а) облегчается работа по составлению программы;

б) уменьшается количество ошибок при программировании;

в) наличие операторной схемы алгоритма облегчает проверку готовых программ;

г) появляется возможность возобновлять работу над программой после длительного перерыва или поручать продолжение начатой работы другому лицу.

Среди всех задач выделяют расчетные, алгоритмы которых легко описываются с помощью математических формул, и логические задачи.

К логическим задачам относятся преимущественно многочисленные задачи управления и планирования. При их решении на ЭВМ схемы алгоритмов приобретают важное значение, так как они являются пока почти единственным формальным аппаратом, отображающим динамику сложных процессов.

Схема алгоритма – функционально ориентированное изображение, с помощью которого, используя текст и специальные символы, описывают последовательность шагов процесса во времени, связи рассматриваемой системы с внешней средой и разветвление процесса. Краткое текстовое или формальное описание шагов приводится внутри символов. Сведения о входных и выходных данных некоторых шагов процесса могут даваться в закодированной форме. Схемы алгоритмов обладают тем преимуществом, что для их понимания не требуется специальных знаний.

Для удобства чтения схем алгоритмов в ряде стран разработаны стандартные графические обозначения для наиболее часто употребляемых и специфических операторов.

Поясним понятие схемы алгоритма. Пусть требуется вычислить функцию

$$y = \begin{cases} x^2, \text{если } x < 0, \\ x^2 + \sqrt{x}, \text{если } x \geq 0 \end{cases}$$

для всех дискретных значений x_i с шагом 0,1 из сегмента [−7; 5].

Так как y представлен через аналитические выражения x^2 и $x^2 + \sqrt{x}$, по которым могут быть произведены вычисления в результате подстановки в них значений x_i, то алгоритм для вычисления очевиден. Остается построить схему алгоритма.

Из условий задачи вытекает необходимость выполнения следующих операций:

1 Ввести исходное значение $x := X_0 := -7$. ПКУ2.

2 $M := x^2$. ПКУЗ.

3 Проверить $x \geq 0$. Если да, то ПКУ4; если нет, то ПКУ5.

4 $y := (M + \sqrt{x})$. ПКУ6.

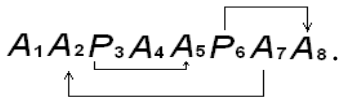
5 $y := M$. ПКУ6.

6 Проверить $x = 5$. Если да, то ПКУ8; если нет, то ПКУ7.

7 $x : x + 0{,}1$. ПКУ2.

8 Конец.

Обозначим эти восемь операций соответственно A_1, A_2, P_3, A_4, A_5, P_6, A_7, A_8 (или еще короче: 1, 2, ..., 8). Если расположить эти символы в строчку и с помощью стрелок указать связи, то получим операторную схему алгоритма:



Если теперь вместо символов операторов начертить прямоугольники, в них записать содержание действий, выполняемых данными операторами, и показать связями последовательность этих действий, то получим схему алгоритма (рисунок 1.6).

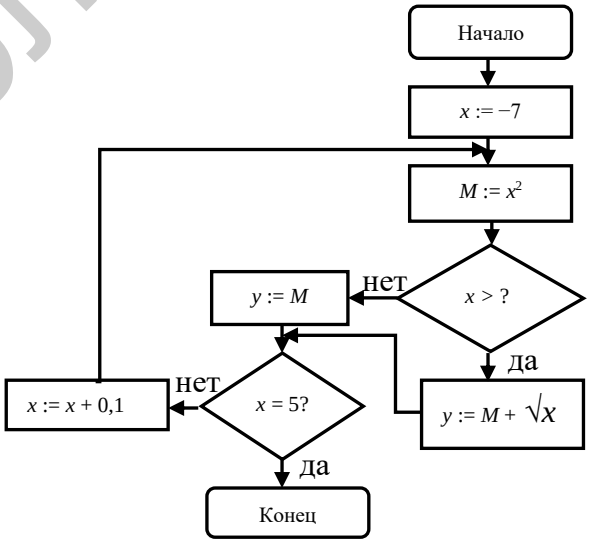


Рисунок 1.6 – Схема алгоритма

Граф-схема алгоритма (ГСА) – конечный связный ориентированный граф

$$G = (V,\ E),$$

где V – множество вершин графа $v_i \in V$, $1 \le i \le N$, которые соответствуют операторам;

E – множество дуг графа $\{\vec{e}_k = e_{i,j} = (v_i, v_j)\} \in E$, $k = 1, \ldots, M$, $i = 1, \ldots, N$, $j = 1, ..., N$, $i \neq j$, которые задают порядок следования операторов;

$N = |V|$ – число вершин графа;

$M = |E|$ – число дуг графа.

Таким образом, ГСА – графическое изображение последовательности операций, согласно которым получают решение задачи. Каждый этап процесса обработки информации представляется в виде геометрических символов (блоков), имеющих определенную конфигурацию в зависимости от характера выполняемых операций. Перечень символов, их наименование, отображаемые ими функции, форма и размеры определяются ГОСТами.

Основные свойства граф-схем алгоритмов следующие:

а) граф-схема состоит из конечного числа точек, называемых вершинами (узлами), и соединяющих их стрелок;

б) в граф-схеме имеются два особых узла: входной, в который не входит ни одна стрелка, и выходной, из которого не выходит ни одна стрелка;

в) все узлы граф-схем отмечены своей логической переменной P_i или арифметическим оператором A_j;

г) из каждого узла P_i отходят, как правило, две стрелки, а из узла A_j – только одна.

ГСА для приведенного выше алгоритма показана на рисунке 1.7. Если в ГСА логические переменные и арифметические операторы заменить соответствующими описаниями, то граф-схема превратится в схему алгоритма (см. рисунок 1.6).

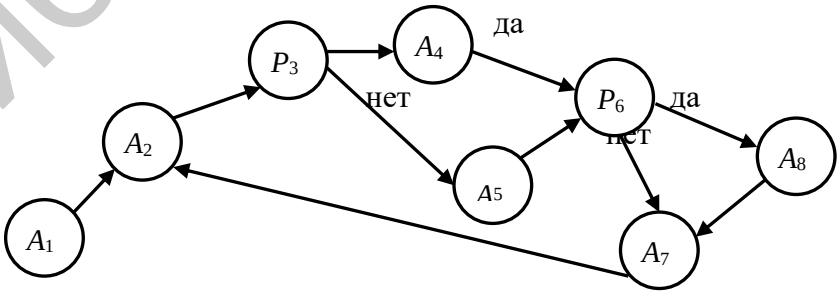


Рисунок 1.7 – Граф-схема алгоритма

При всем многообразии алгоритмов решения задач в них можно выделить *три основных вида вычислительных процессов*:

- линейный;
- ветвящийся;
- циклический.

Линейным называется такой вычислительный процесс, при котором все этапы решения задачи выполняются в естественном порядке следования записи этих этапов.

Ветвящимся называется такой вычислительный процесс, в котором выбор направления обработки информации зависит от исходных или промежуточных данных (от результатов проверки выполнения какого-либо логического условия).

Циклическим (циклом) называется многократно повторяемый участок вычислений.

Графовая модель представления параллельного алгоритма. *Параллельный алгоритм* – алгоритм, операции которого могут выполняться одновременно. Приведенные выше толкования алгоритма рассматривают его как процесс последовательной обработки структур данных, протекающий в дискретном времени так, что в каждый следующий момент времени структура данных получается по заданным правилам по структуре данных величин, имевшихся в предыдущий момент. Структура элементарных шагов в определении алгоритма не оговаривается, поэтому, правила преобразования величин могут допускать или предписывать их параллельную обработку. Алгоритм может быть параллельным, когда некоторые шаги обработки выполняются параллельно. Так, алгоритм сложения векторов может быть представлен как последовательный в виде $C_i = A_i + B_i$, $i = 1, 2, ..., n$, или в матричной записи – $C = A + B$, семантика которого допускает параллельную обработку элементов векторов.

Если при решении некоторой задачи процессоры осуществляют одинаковую последовательность вычислений, но используют разные данные, то говорят о *параллелизме по данным*. Например, при поиске по базе данных каждый процессор может работать со своей частью базы данных. Если процессоры осуществляют разные задания одной задачи, выполняют разные функции, то говорят о *функциональном параллелизме*.

Основная цель параллельных вычислений – уменьшение времени решения задачи. Чтобы ускорить решение задачи, недостаточно иметь параллельную ВС. Для такой системы нужно еще создать специальную (параллельную) программу. Для того чтобы алгоритм мог быть эффективно реализован в виде параллельной программы, он должен обладать *внутренним параллелизмом* (алгоритм можно разбить на параллельные, но не обязательно независимо выполняемые части). Распараллеливание, осуществляемое до начала выполнения алгоритма, называют *статическим*, а осуществляемое во время выполнения алгоритма, – *динамическим*.

С параллельностью связано понятие масштабируемости. *Масштабируемая параллельная система* – это такая параллельная система, производительность которой пропорциональна числу содержащихся в ней процессоров. Масштабируемость зависит от возможностей коммуникационных сетей и параллельного

алгоритма: алгоритм, хорошо работающий в вычислительной системе (ВС) с малым числом процессоров, может плохо работать (не давать ожидаемого ускорения) в системе с большим числом процессоров.

Рассмотрим, например, алгоритм вычисления выражения $a \cdot b + \sqrt{c \cdot d}$. Вначале нужно вычислить произведения $a \cdot b$ и $c \cdot d$, затем извлечь корень, и, наконец, выполнить сложение. Описанный алгоритм можно изобразить следующим образом (рисунок 1.8).

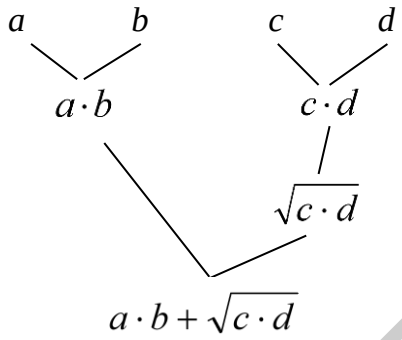


Рисунок 1.8 – Схема параллельного алгоритма

Мы видим, что процесс обработки данных может быть выражен в виде однонаправленного графа. Такой граф можно изобразить на плоскости, причем каждую арифметическую операцию располагать максимально высоко (если ось времени направлена вниз), но не выше тех операций, результат которых нужен для ее вычисления. В таком случае высота графа будет равна минимальному времени (числу шагов) решения этой задачи на идеальной параллельной ВС с неограниченным числом вычислителей.

Степенью параллелизма (параллельной сложностью) алгоритма называется число его операций, которые можно выполнить параллельно (ширина графа, построенного описанным выше способом). Степень параллелизма алгоритма сложения векторов равна *n*. Операции, которые всегда могут быть выполнены на идеальной ЭВМ параллельно независимо от их количества, иногда называют *операциями фиксированной глубины*.

Этапы параллельного проектирования. Получить параллельный алгоритм решения задачи можно путем распараллеливания имеющегося последовательного алгоритма или путем разработки нового параллельного алгоритма. Возможно, для осуществления распараллеливания алгоритм решения задачи придется заменить или модифицировать (например, устранить некоторые зависимости между операциями).

Разработка алгоритмов параллельных вычислений состоит в следующем:

а) выполнить анализ имеющихся вычислительных схем и осуществить их разделение (декомпозицию) на части (подзадачи), которые могут быть реализованы в значительной степени независимо друг от друга;

б) выделить для сформированного набора подзадач информационные взаимодействия, которые должны осуществляться в ходе решения исходной поставленной задачи;

в) определить доступную для решения задачи ВС и выполнить распределение имеющего набора подзадач между процессорами системы;

г) разделить вычисление на независимые части;

д) выделить информационное взаимодействие между частями;

е) масштабировать задачи;

ж) распределить все задачи между процессорами.

При рассмотрении параллельных алгоритмов вводится понятие *параллельной ГСА*, в состав которой входят вершины распараллеливания/синхронизации, функциональность которых обычно совмещается (фрагмент такой ГСА приведен на рисунке 1.9).

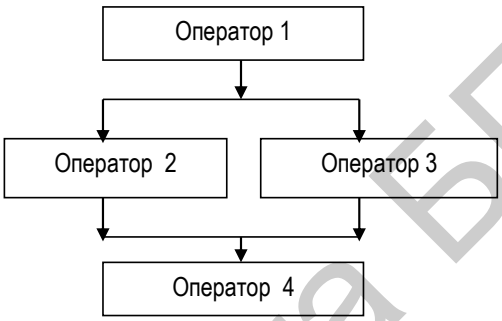


Рисунок 1.9 – Фрагмент параллельной ГСА

Иногда в состав ГСА вводятся вершины дополнительных типов с целью обеспечения возможности моделирования выполнения алгоритма сетью Петри. Однако не любой ориентированный граф, составленный из вершин указанных выше типов, может быть отождествлен с корректным алгоритмом. Например, из операторной вершины не может выходить более одной дуги. Поэтому на практике обычно ограничиваются рассмотрением подкласса граф-схем алгоритмов, удовлетворяющих свойствам безопасности, живости и устойчивости.

Сети Петри. Среди методов описания и анализа дискретных параллельных систем выделяют подход, который основан на использовании сетевых моделей, относящихся к сетям специального вида, предложенных Карлом Петри для моделирования асинхронных информационных потоков в системах преобразования данных. Сети Петри обеспечивают описание как параллельных алгоритмов, так и собственно ВС и ее устройств.

Структура сети представляется *ориентированным двудольным графом*. Множество вершин графа V разбивается на два подмножества T и P: $V = T \cup P$, $T \cap P = \varnothing$. Дугами могут связываться вершины из множеств T и P. Динамика развития процессов отражается в вершинах P *метками* (марками). Распределение меток по вершинам P называют *маркированием сети*. Каждое маркирование соответствует определенному состоянию сети.

Сеть Петри определяется следующим образом:

$$N = \{P, T, F, M_0\},$$

где $P = \{p_i\}$, $i = 1, 2, ..., n$ – непустое конечное множество позиций (мест);

$T = \{t_j\}$, $j = 1, 2, ..., m$ – непустое конечное множество переходов;

$F : T \times P \cup P \times T \to \{0, 1\}$ – *функция инцидентности*, такая, что каждое место инцидентно какому-либо переходу ($T \times P \to \{0, 1\}$ – функция следования), каждый переход инцидентен какому-либо месту ($P \times T \to \{0, 1\}$ – функция предшествования), т. е. она задает множества дуг $\{t_j, p_i\}$ и $\{p_i, t_j\}$;

$M_0 : P \to N_0$ – начальное маркирование (состояние) сети (N_0 – множество положительных целых чисел).

Обозначим $J(p_i) = \{(t_j, p_i) \mid J(t_j, p_i) = 1\}$ и $J(t_j) = \{(p_i, t_j) \mid O(p_i, t_j) = 1\}$ как множества дуг, предшествующих позиции p_i и переходу t_j соответственно.

Здесь запись $J(t_j, p_i) = 1$ обозначает наличие дуги (t_j, p_i), а запись $O(p_i, t_j) = 1$ – дуги (p_i, t_j). Аналогично дуги, следующие из p_i и t_j, представим множествами $O(p_i) = \{(p_i, t_j) \mid O(p_i, t_j) = 1\}$, $O(t_j) = \{(t_j, p_i) \mid J(t_j, p_i) = 1\}$.

Входные позиции перехода t_j объединяются в множества его предшественников: $Pre(t_j) = \{p_i \in P \mid O(p_i, t_j) = 1\}$, а выходные позиции – в множества позиций-последователей: $Post(t_j) = \{p_i \in P \mid J(t_j, p_i) = 1\}$.

Маркирование сети представляется вектором $M = < m(p_i) >$, где $m(p_i)$ – число меток в позиции p_i. Переход t_j возбужден при маркировании M и может сработать, если выполняется условие $\forall p_i \in Pre(y_j)\,(m(p_i) - O(p_i, t_j) \geq 0)$, т. е. число меток $m(p_i)$ больше или равно числу дуг (p_i, t_j). Условием срабатывания является истинность функции инцидентности при заданных p_i и t_j.

Срабатывание перехода t_j приводит к тому, что каждая позиция $p_i \in Pre(t_j)$ теряет $O(p_i, t_j)$ меток, а каждая из позиций $Post(t_j)$ получает $J(t_j, p_i)$ меток.

Сеть Петри представляет собой *двудольный ориентированный граф*, в котором позициям соответствуют вершины, изображаемые кружками, а переходам – вершины, изображаемые утолщенными черточками; функциям *J* соответствуют дуги, направленные от позиций к переходам, а функциям *O* – дуги, направленные от переходов к позициям.

Как и в системах массового обслуживания, в сетях Петри вводятся объекты двух типов:

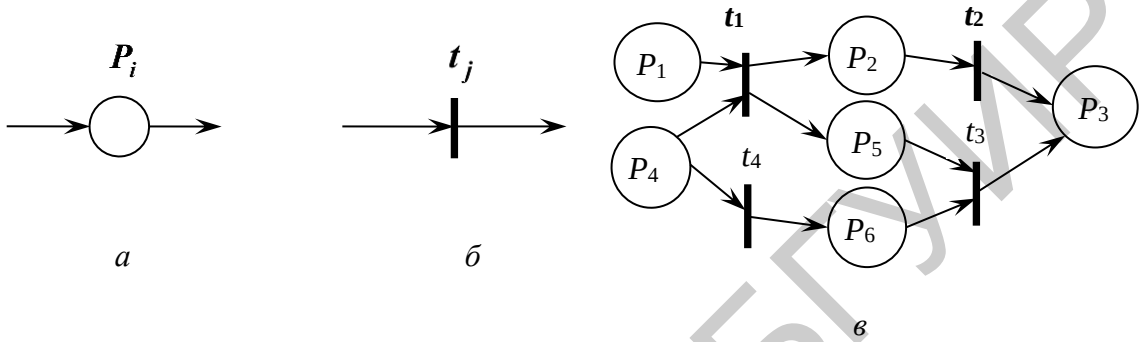
- динамические, которые изображаются метками (маркерами) внутри позиций;

- статические, которым соответствуют вершины сети Петри.

Распределение маркеров по позициям называют *маркировкой*. Маркеры могут перемещаться в сети. Каждое изменение маркировки называют событием, причем каждое событие связано с определенным переходом. Считается, что

события происходят мгновенно и разновременно при выполнении некоторых условий. Последовательность событий образует моделируемый процесс.

Каждому условию в сети Петри соответствует определенная позиция. Совершению события соответствует срабатывание (возбуждение или запуск) перехода, при котором маркеры из входных позиций этого перехода перемещаются в выходные позиции. В соответствии с этим граф сети Петри является двудольным ориентированным мультиграфом. Изображение позиции и перехода на графе показано на рисунке 1.10.

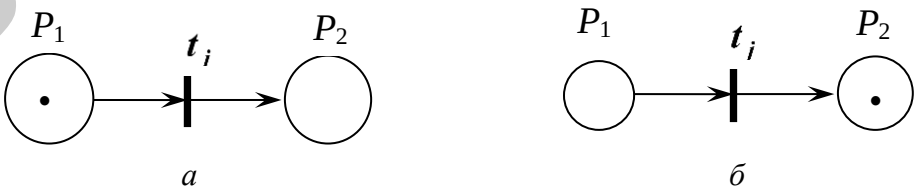


а – позиция; *б* – переход; *в* – граф

Рисунок 1.10 – Графическое изображение сети Петри

Ориентированные дуги могут соединять только позиции и переходы в прямом и обратном направлениях (свойство двудольности). Сеть Петри является мультиграфом, так как допускается кратность дуг между позициями и переходами (вершинами графа).

Правила срабатывания переходов конкретизируют следующим образом: переход срабатывает, если для каждой из его входных позиций выполняется условие $N_i > K_i$, где N_i – число маркеров в i-й входной позиции, K_i – число дуг, идущих от i-й позиции к переходу; при срабатывании перехода число маркеров в i-й входной позиции уменьшается на K_i, а в j-й выходной позиции – увеличивается на M_j, где M_j – число дуг, связывающих переход с j-й позицией. Пример смены разметки показан на рисунке 1.11. Процесс срабатывания переходов и смены разметки называют работой сети Петри.



а – до срабатывания перехода t_j; *б* – после срабатывания перехода t_j

Рисунок 1.11 – Смена разметки сети Петри

Произвольная ГСА является частным случаем сети Петри с одной меткой, которая соответствует текущему состоянию автомата при выполнении последовательной части алгоритма, и с несколькими метками, количество которых соответствует количеству параллельных ветвей алгоритма при выполнении параллельной части алгоритма управления. Множество переходов *t* при этом соответствует множеству условных вершин, множество событий – множеству наборов управляющих сигналов, а множество мест – множеству состояний автомата.

Если при маркировании *M* возбуждено несколько переходов, то порядок их срабатывания не определен, и, следовательно, может быть представлено несколько последовательностей срабатывающих переходов.

Представление алгоритма в виде диаграммы расписания. Для удобства представления параллельных алгоритмов наряду с другими способами используют также временные диаграммы выполнения операторов при заданных значениях времени начала (окончания) их выполнения. Операторы обозначаются прямоугольниками с длиной, равной времени выполнения соответствующего оператора. Для указания связей между прямоугольниками-операторами используются стрелки, которые соответствуют дугам в графе алгоритма. Когда связей становится много, из-за множества стрелок проследить эти связи становится трудно.

Для того чтобы диаграмма оставалась наглядной при любом количестве связей и однозначно отражала расписание параллельного алгоритма, введем следующие дополнительные правила. Ось ординат разобьем на интервалы, каждый из которых соответствует одному из параллельно работающих процессоров. В каждом интервале будем размещать только те прямоугольники-операторы, которые закреплены за соответствующим этому интервалу процессором. Операторы на диаграмме будем обозначать в центре прямоугольников цифрой в кружке, а связи между операторами – цифрами слева и справа от этого кружка. В левой части прямоугольника-оператора будем записывать номера предшествующих по информационной связи операторов, а в правой части – номера операторов, следующих за данным оператором. Поскольку каждый прямоугольник диаграммы размещается в интервале «своего» процессора, описанный способ их нумерации отражает также связь между процессорами. Поэтому этот способ представления является исчерпывающим наглядным представлением расписания параллельного алгоритма и может быть удобным программисту для написания параллельной программы. При этом информация о номерах связанных операторов может использоваться для оценки объема передаваемых данных, как между процессами, так и между процессорами.

Выбор численных методов реализации алгоритмов. Массовое использование ЭВМ в решении ряда прикладных задач привело к бурному развитию теории численных методов вычислений. Численный метод – это метод приближенного или точного решения математических задач, основанный на построении конечной последовательности действий над конечным множеством чисел. Дискретная форма представления информации в ЭВМ потребовала и создания адекватных методов ее обработки. Если по своей природе задача носит дискретный характер, и имеются соответствующие модели вычислений, то в этом случае метод может практически

сразу использоваться для машинной реализации, например, метод нахождения наибольшего общего делителя двух целых положительных чисел. Проблемы могут лишь возникнуть в связи с задачей представления в памяти очень больших чисел или же из-за ограничений на время ожидания ответа. Дополнительных затрат потребовала переориентация методов, использующих непрерывные математические модели вычислений. Непрерывные модели вычислений стали заменяться дискретными. Например, вычисление интеграла методом подстановок свелось к суммированию. Если раньше задача взять одномерный интеграл на отрезке решалась для произвольных границ отрезка, то теперь она стала решаться для фиксированных границ отрезка. В первом случае точность результата определялась на базе погрешностей вычислений по итоговой формуле, а во втором случае она стала зависеть еще и от количества точек деления отрезка, определяющих число элементарных трапеций при суммировании. Таким образом, появилась специфическая задача автоматического выбора числа точек деления отрезка, чтобы избежать больших погрешностей метода.

Эти проблемы породили задачу анализа устойчивости вычислений по дискретным алгоритмам.

Во многих вычислительных алгоритмах, разработанных до появления ЭВМ, исследовались только вопросы получения погрешностей из-за аппроксимации непрерывных методов (моделей) вычислений дискретными операциями. Погрешности представления данных и накопления систематических ошибок при большом числе простейших операций над приближенными числами не изучались.

Такого рода трудности стали обнаруживаться, когда теоретически сходящиеся процессы в некоторых случаях на ЭВМ не дали достоверных результатов. Поэтому, применяя ЭВМ для решения математических задач, необходимо учитывать ее особенности как вычислительного инструмента (в первую очередь это касается персональных ЭВМ и программируемых микрокалькуляторов с небольшой разрядностью для представления чисел):

- количество цифр в изображении чисел, над которыми производятся действия, и погрешности их представления;
- большую скорость операций над числами, хранящимися в оперативной памяти;
- сравнительно малую скорость ввода исходных данных и программ, а также вывода результатов;
- сравнительно малую скорость обмена числами между оперативной и внешней памятью;
- ограниченную емкость оперативной памяти при практически безграничной емкости накопителя;
- возможность случайных сбоев в работе машин.

Выбирая численный метод, нельзя забывать о необходимости проверки правильности вычислений при решении задачи на машине. Способ контроля путем повторных расчетов не всегда эффективен. Более экономным и эффективным может оказаться контроль путем проверки каких-либо заранее известных соотношений между вычисляемыми величинами (например, $\sin^2 x + \cos^2 x = 1$).

Некоторые численные методы практически не требуют контроля правильности вычислений. К этому классу, в частности, относятся сходящиеся итерационные методы. Их достоинством является то обстоятельство, что получение ошибочного результата при одной из итераций не приводит к ухудшению окончательного результата вычислений, а лишь увеличивается количество итераций, которые должна выполнять машина. При быстрой сходимости итерационного процесса затраты машинного времени будут сравнительно небольшими. Приведем пример такого итерационного процесса. Пусть необходимо вычислить $y = \sqrt{x}$ с точностью до 0,01.

На ЭВМ для извлечения квадратного корня из x многократно применяется следующая процедура вычисления: $y_{i+1} = 0{,}5\ (y_i + x : y_i)$.

Всего $(i + 1)$ раз, пока не выполнится неравенство $|y_{i+1} - y_i| \le e$, тогда y полагают равным y_{i+1}. За y_0 обычно можно выбрать x.

В нашем примере $y_{i+1} = 0{,}5\ (y_i + 2 : y_i)$, $|y_{i+1} - y_i| < 0{,}01$ и $y_0 = x = 2$.

Применим эти соотношения для иллюстрации процесса вычислений без ошибок и с одной ошибкой.

Процесс вычисления y без сбоя с точностью до 0,01:

1 $y_1 = 0{,}5\ (2 + 2 : 2) \approx 1{,}5$ $|1{,}5 - 2| > 0{,}01$.

2 $y_2 = 0{,}5\ (1{,}5 + 2 : 1{,}5) = 1{,}67$ $|1{,}67 - 1{,}5| > 0{,}01$.

3 $y_3 = 0{,}5\ (1{,}67 + 2 : 1{,}67) \approx 1{,}43$ $|1{,}43 - 1{,}67| > 0{,}01$.

4 $y_4 = 0{,}5\ (1{,}43 + 2 : 1{,}43) \approx 1{,}41$ $|1{,}41 - 1{,}43| > 0{,}01$.

5 $y_5 = 0{,}5\ (1{,}41 + 2 : 1{,}41) = 1{,}41$ $|1{,}41 - 1{,}41| < 0{,}01$.

Ответ: $y = 1{,}41$.

Процесс вычисления y с одним сбоем (пусть на шаге 3 вместо 1,43 было получено $y_3 = 0{,}43$, и с этого шага продолжим вычисления):

4 $y_4 = 0{,}5\ (0{,}43 + 2 : 0{,}43) = 2{,}54$ $|0{,}43 - 2{,}54| > 0{,}01$.

5 $y_5 = 0{,}5\ (2{,}54 + 2 : 2{,}54) = 1{,}66$ $|2{,}54 - 1{,}66| > 0{,}01$.

6 $y_6 = 0{,}5\ (1{,}66 + 2 : 1{,}66) \approx 1{,}44$ $|1{,}44 - 1{,}66| > 0{,}01$.

7 $y_7 = 0{,}5\ (1{,}44 + 2 : 1{,}44) \approx 1{,}40$ $|1{,}40 - 1{,}44| > 0{,}01$.

8 $y_8 = 0{,}5\ (1{,}40 + 2 : 1{,}40) \approx 1{,}41$ $|1{,}41 - 1{,}40| = 0{,}01$.

Ответ: $y = 1{,}41$.

Выбирая численный метод, следует учитывать особенности подготовки задачи для решения ее на машине:

- необходимость сведения ее к выполнению последовательности арифметических действий (практически можно использовать и другие элементарные действия, для которых уже составлены стандартные программы);
- трудоемкость процесса программирования;
- необходимость отладки программы на машине.

По условиям задачи всегда требуется знать и точность решения. Точность решения обычно задают значением максимально допустимой погрешности. В связи с этим необходимо выбирать, а иногда и специально разрабатывать соответствующий численный метод.

Во всякой ЭВМ числа, представленные в режиме с плавающей запятой, обычно имеют небольшую погрешность. ЭВМ оперирует с довольно большим количеством цифр, и в результате вычислений накапливается арифметическая погрешность, которая складывается с погрешностью метода. В этом случае бывает необходимо оценивать общую погрешность, которая не должна превышать максимально допустимую погрешность.

В целях экономии времени и количества ячеек в памяти полезно выбирать алгоритмы с небольшой связностью.

Связностью алгоритма называется количество данных, которые нужно накапливать в запоминающих устройствах машины для перехода от одного этапа вычислений к другому.

Если связность алгоритма велика, то возникает необходимость переноса промежуточных результатов из оперативной памяти во внешнюю. Последнее обстоятельство влечет за собой резкое увеличение времени, расходуемого на решение задачи. Иногда удается путем незначительных изменений значительно уменьшить связность алгоритма.

Пример – Пусть требуется вычислить многочлен

$$y = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_i x + a_0.$$

Если сначала вычислить все одночлены полинома, то для перехода от этапа вычисления величин $y_i = a_i x_i$ к этапу вычисления y требуется запомнить числа $y_0, y_1, \ldots, y_n$, что приведет к связности алгоритма, равной $(n+1)$.

Но эти вычисления можно упростить по следующим формулам:

$$T_i = T_{i-1} x, \; y_i = y_{i-1} + a_i T_i, \; T_0 = 1, \; y_0 = a_0.$$

После выполнения всех вычислений получим результат: $y = y_{n+1}$.

В этой схеме вычислений по сравнению с предыдущей требуется помнить при переходе к следующему этапу только два числа, т. е. связность алгоритма равна 2.

Почти все решения задач связаны с вычислением некоторых функций и, в частности, функции одной переменной $f(x)$.

Функция $y = f(x)$ может задаваться многими способами, но машинные вычисления функций опираются на стандартные подпрограммы и следующие способы их представления:

а) в явном виде с помощью аналитической формулы, содержащей конечное число основных операций (арифметических, а также операций типа $|x|$, x^n, $\ln x$, $\sin x$, e^x и т. п.);

б) конечной системой многочленов;

в) таблицами, графиками.

При численном решении задач значение функции вычисляется всегда с заданной точностью. Поэтому многие способы задания функций, теоретически требующие выполнения бесконечного числа операций, на практике сводятся к способам с конечным числом операций. Например, задание функции в виде суммы членов сходящегося степенного ряда можно отнести к первому случаю,

т. к. в памяти машины хранится конечное число первых членов ряда и отбрасываются остальные. Точнее, программа вычисления функции с заданной точностью реализована на базе необходимого числа членов ряда.

Выбор способа задания функции и соответственно вычисления ее значений базируются на двух основных факторах:

а) экономия количества ячеек памяти, необходимого для осуществления выбираемого способа вычислений;

б) экономия машинного времени, расходуемого на вычисление всех значений функций.

Если имеется ряд способов, не требующих использования внешней памяти, то прибегают к тому из них, который экономнее относительно временных затрат. Лишь в случае небольших временных затрат можно применять легко программируемые алгоритмы, требующие использования внешней памяти. Пусть функция задана явно с помощью формулы, содержащей конечное число операций и описанной различными элементарными функциями ($|x|$, x^n, $\ln x$, $\sin x$ и т. п.). Программы для вычисления элементарных функций имеют все современные ЭВМ. Поэтому на основе стандартных программ можно получить программы для вычисления функции. Если оказывается, что вычисления длительны и громоздки, то часто прибегают к описанию функции многочленами, таблицами и т. п.

Возможна и обратная ситуация, когда большие таблицы заменяются несколькими простыми функциями, чтобы ускорить время выборки данных и сэкономить количество ячеек памяти.

Интервал (a, b) изменения независимой переменной x разбивают на несколько подынтервалов, на каждом из которых выбирают нужный способ задания функции.

Рассмотрим реализацию способа задания функции на одном из подынтервалов путем подбора соответствующего многочлена.

Степень многочлена подбирается так, чтобы выполнялись условия:

а) степень должна быть как можно более низкой;

б) значение многочлена должно отличаться от соответствующего значения функции на величину, меньшую чем некоторое малое число E, характеризующее допустимую погрешность получаемых значений функции.

Если $P_i(x)$ – многочлен, то на соответствующем подынтервале должно выполняться неравенство $|f(x) - P_i(x)| < E$.

В этом случае в память вместо большой таблицы значений функции или большой по объему программы для вычисления ее значений вводят небольшую таблицу коэффициентов многочленов и программу их вычисления. По значению x выбирается соответствующий многочлен, и по стандартным программам вычисляется его значение.

Комбинаторные методы реализации алгоритмов. Характерной особенностью задач проектирования ВС и их компонентов на сверхбольших интегральных схемах (СБИС) является большая область поиска решений. Многие задачи являются *NP*-полными (число шагов в процессе поиска растет по экспоненте с возрастани-

ем размерности задачи). Большинство же задач являются *NP*-сложными, т. е. более сложные, чем *NP*-полные.

Все задачи решаются методами комбинаторного поиска, основанными на построении дерева поиска. Эти методы порождают ветвящиеся процессы последовательного конструирования искомых решений, реализуя многошаговые переходы от начальной ситуации, отражающей лишь исходные данные решаемой задачи, через некоторое множество промежуточных ситуаций (в каждой из которых решается одна и та же задача, изменяются лишь данные) к заключительным ситуациям, в которых будут представлены найденные решения. Решением является либо путь из корня дерева в концевую вершину, либо множество дуг или множество концевых вершин.

Повышение размерности решаемых задач такого рода достигается разработкой приближенных алгоритмов, которые не гарантируют получение оптимального решения, но позволяют строить достаточно качественные решения за приемлемое для практики время. Таким алгоритмам соответствуют некоторые усеченные деревья поиска. В вырожденном случае такое дерево будет состоять из одной лишь ветви, и соответствующий ей алгоритм будет цепным (без возврата), основанным лишь на редуцировании текущей ситуации. Очевидно, наиболее приемлемыми с точки зрения практики являются именно такие алгоритмы, или алгоритмы с сильно ограниченным перебором. При их построении основное внимание уделяется нахождению эвристических правил построения пути из корня дерева поиска в его концевую вершину, дающую наилучшее приближение к оптимальному.

1.4 Формальные языки

Естественные языки. Многие годы в математике и других точных науках широко использовались естественные языки (русский, английский и др.). Однако вместе с необходимостью однозначно описывать различные процессы для последующей реализации на ЭВМ вскрылся ряд недостатков в построении естественных языков, и пришлось прибегнуть к созданию специальных формальных языков.

Математические языки отличаются от естественных более простым строением, позволяющим достигать в их описании предельной четкости и однозначности, зато они сильно уступают естественным языкам в широте сферы применимости.

В естественных языках различают структуру предложений (их форму) и содержание (их смысл).

Форме конкретного предложения не всегда соответствует единственное содержание. Например, фраза «он встретил ее на поляне с цветами» не является определенной (она была с цветами, поляна была с цветами или он был с цветами). Фраза «предмет имеет синий цвет» обладает расплывчатым смыслом, так как разные люди из множества допустимых голубых и синих предметов выберут разное их количество по признаку «синий».

Грамматические правила естественного языка могут зависеть от смысла получаемых с их помощью предложений. Например, правильными будут «я вижу

стол» (а не стола) и «я вижу студента» (а не студент). Чтобы правильно строить предложения, нужно знать, говорится в нем об одушевленном или неодушевленном предмете, так как от этого зависит смысл предложения. Зависимость формы грамматических правил от смысла предложений приводит к ряду затруднений, если мы, например, желаем осуществлять на ЭВМ перевод с одного языка на другой, так как ЭВМ не может понимать смысл. Естественный язык допускает и противоречивые конструкции: «Высказывание, которое я произношу, ложно» (а ведь оно не может быть без указания противоречия ни истинным, ни ложным).

Следовательно, естественные языки противоречивы, неоднозначны и неточны. Наиболее простой путь для преодоления этих затруднений состоит в использовании некоторого подмножества естественного языка, фразы которого однозначны и в смысловом отношении не зависят ни от каких внешних для выбранного подмножества языка условий. Смысл каждой фразы такого подмножества определяется только ее формой.

Описание формального языка. Для описания формального языка нужен другой язык. Описываемый язык называют языком-объектом в отличие от так называемого метаязыка («мета» – вне, за пределом), применяемого для описания языка-объекта. Говоря о языке, в дальнейшем, чтобы избегать путаницы, мы будем четко различать метаязыки (внешние языки) и язык-объект.

Завершенную конструкцию естественного языка обычно называют предложением. Сохраним этот термин и для формальных языков. Внешний язык должен обладать возможностями для описания как структуры, так и смысла предложений языка-объекта. Часто его разделяют на два языка, один из которых предназначен для описания структуры предложений языка-объекта, а другой – для описания смысла. Первый язык при этом называют метасинтаксическим, а второй – метасемантическим. Систему правил, определяющих структуру предложений языка-объекта, называют его синтаксисом; соответствие между предложениями языка-объекта и их значениями – его семантикой.

При описании структуры предложений формального языка обычно указывают его алфавит (конечное множество символов) и приводят правила для построения предложений. Для определенности будем считать, что каждое правило называет некоторую операцию, которую можно применить в процессе конструирования предложений формального языка, и указывает допустимые исходные данные для этой операции.

Следовательно, под синтаксисом формального языка понимается некоторый перечень операций, для каждой из которых известна область ее определения. Кроме того, предполагается, что синтаксис содержит формулировку некоторого условия, которое выполняется для законченных конструкций формального языка и не выполняется для конструкций, не являющихся его предложениями.

В качестве наиболее известного примера формальных языков можно назвать коды. Внешний язык такого формального языка состоит из двух подъязыков, первый из которых является множеством осмысленных предложений, относящихся к интересующей нас области, а второй содержит описания двух правил, называемых соответственно правилом кодирования и правилом деко-

дирования. Синтаксис этого формального языка состоит из одного правила кодирования, которое при применении к предложению первого подъязыка дает предложение формального языка.

Правило декодирования определяет обратную операцию по отношению к операции кодирования (которая должна допускать обратную операцию). Вообще говоря, правило декодирования задает семантику кода. Чтобы код был формальным языком, его внешний язык тоже должен быть формальным языком. Например, число семь арабскими цифрами можно следующим образом закодировать в различных системах счисления: 111_2; 21_3; 13_4; 12_5; 11_6; 10_7; 7_8; 7_9 (нижний индекс указывает систему счисления: двоичную, троичную и т. д.). Правило декодирования тогда сведется к алгоритму перевода из кодированной записи в десятичную: $111_2 = 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 7$; $21_3 = 2 \cdot 3^1 + 1 \cdot 3^0 = 7$.

Формальная грамматика. В большинстве описаний различных разработок по алгоритмическим языкам и языкам описаний различных объектов широко используется *метасинтаксический язык*, предложенный Бэкусом. Этот язык состоит из конечного числа предложений, называемых металингвистическими формулами Бэкуса. При их построении используются два универсальных метасимвола: « ::= » и « | », первый можно читать как «по определению есть», второй – «или». Остальные метасимволы выбираются произвольно разработчиком формального языка, который является также и разработчиком внешнего языка.

Метасимволы являются произвольными фразами естественного языка, заключенными в угловые скобки (<...>). Будем называть такие метасимволы составными. Также в формулах Бэкуса используются символы формального языка, которые легко узнать по тому, что они отличны от « ::= » и « | » и стоят вне угловых скобок. В левой части формулы Бэкуса должен стоять составной метасимвол, за ним следует знак « ::= », после которого записывается правая часть формулы. Она может быть строкой либо пустой, либо состоящей из конечного числа составных метасимволов и символов формального языка, либо конечной последовательностью таких строк, разделенных знаками « | ». По определению две формулы Бэкуса, имеющие одинаковые левые и различные правые части, обозначают то же самое, что формула, которая получится, если к правой части любой из первых формул приписать символ « | », а за ним правую часть другой из них.

Начало и конец формулы Бэкуса ничем не обозначены, поэтому начинать формулу удобно, отступив от начала, а кончать, не доходя до конца строки. При переносе со строки на строку никаких дополнительных знаков ставить не следует. Смысл формулы Бэкуса легко понять, читая ее на естественном языке (при этом угловые скобки не употребляются), пустая строка в правой части формулы читается как «пусто».

Пример – Формальный язык для представления четных натуральных чисел с помощью формулы Бэкуса может быть описан так:

<ненулевая цифра> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

<цифра> ::= 0 | <ненулевая цифра>

<четная цифра> ::= 0 | 2 | 4 | 6 | 8 |

<начало> ::= <ненулевая цифра> | <начало> <цифра>

<четное число> ::= <четная цифра> | <начало> <четная цифра>

Читая эти строки на естественном языке, убеждаемся в однозначности восприятия определяемых в них понятий: четко сказано, что *ненулевая цифра* есть по определению 1, или 2, или 3, или 4, или 5, или 6, или 7, или 8, или 9, т. е. фактически понятие ненулевой цифры задано прямым перечислением составляющих его объектов; *начало* есть по определению ненулевая цифра или число, которое начинается не с нулевой цифры. Например, 2, 25 и 20 подпадают под определение понятия начала, так как 2 – ненулевая цифра в первом случае, а во втором и третьем случаях 2 играет роль начала, а другие цифры могут быть любыми. Анализируя аналогичным образом понятие *четного числа*, видим, что оно может состоять из одной *четной цифры* или любого числа с окончанием на *четную цифру*. Действительно, это правило позволяет порождать четные числа из любого количества цифр, так как, приписывая к началу справа еще одну цифру, мы снова получаем начало, к которому можно приписать четную цифру.

Если для формального языка известно точное и однозначное правило, преобразующее его предложение в предложения некоторого другого языка, обладающего семантикой, то тем самым задается семантика нашего формального языка. Семантика называется формальной, если указанное правило является алгоритмом. Теория формальных грамматик занимает в математической лингвистике центральное место, так как она обеспечивает возможность вести переработку смыслов в тексты и обратно.

Формальная грамматика – система правил, описывающая множество конечных последовательностей символов. Конечные последовательности символов (цепочки), входящие в указанное множество, называются предложениями, а само множество – языком, описываемым данной формальной грамматикой.

Различают два типа формальных грамматик:

- порождающие, которые представляют собой системы правил, позволяющие строить предложения языка;
- распознающие – это системы правил, распознающие по любой цепочке, является ли она предложением.

Это деление в некоторой степени условно, так как любая распознающая грамматика по существу задает способ построения предложений.

Формальные грамматики чаще всего применяют для описания естественных и искусственных (в частности, алгоритмических) языков. Остановимся более детально на изучении порождающих грамматик, исходя из следующих соображений:

1 Математическое значение порождающих грамматик определяется тем, что они *представляют собой* одно из средств эффективного задания множеств.

2 Одним из важных направлений теории порождающих грамматик является изучение сложности вывода как по числу шагов вывода (временная сложность), так и по объему используемой памяти (по максимальной промежуточной цепочке вывода).

3 Большая прикладная роль порождающих грамматик проявляется в построении и изучении искусственных языков, в особенности языков программи-

рования, а также и в ряде других случаев (исследование естественных языков, машинный перевод и т. д.).

Порождающей грамматикой называется упорядоченная четверка

$$\Gamma = <V, W, I, R>,$$

где V и W – непересекающиеся конечные множества, называемые соответственно основным и вспомогательным алфавитом (словарем);

V – словарь терминальных символов;

W – словарь вспомогательных (нетерминальных) символов;

I – начальный (или выделенный вспомогательный) символ, $I \in W$;

R – набор правил вывода, или синтаксических правил, каждое из которых имеет вид $\varphi \rightarrow \Phi$, где φ и Φ – цепочки, состоящие из основных и вспомогательных символов.

Основные символы обычно интерпретируются как слова языка, вспомогательные – как названия классов слов и словосочетаний, начальный символ – как символ предложения.

Синтаксические правила описывают связи между частями предложения. Применение правила $\varphi \rightarrow \Phi$ к цепочке, имеющей вид $\alpha\varphi\beta$, означает преобразование ее в цепочку $\alpha\Phi\beta$ (здесь α и β – цепочки, одна из которых, или даже обе, могут быть пустыми).

Вывод в данной порождающей грамматике есть последовательность цепочек, в которой любая цепочка, кроме первой, получается из предыдущей применением какого-либо правила вывода.

Цепочка основных символов, выводимая из начального символа, называется предложением, а множество всех предложений – языком, порождаемым данной грамматикой.

Пример – Пусть

$V = \{a, b, c\}$,

$W = \{A, B\}$,

$I = A$,

$R = \{A \rightarrow aAB,\ A \rightarrow Bc,\ B \rightarrow b\}$.

Одним из выводов будет последовательность A, aAb, $aBcB$, $abcb$. Здесь цепочка $abcb$ – предложение. Другой пример вывода – A, Bc, bc.

Основные классы порождающих грамматик выделяются в зависимости от ограничений, налагаемых на вид синтаксических правил.

В *бесконтекстной или контекстно-свободной грамматике* (КС – грамматике) используется правило вывода:

$$A \rightarrow \Phi,$$

где A – вспомогательный символ;

Φ – непустая цепочка.

Языки, порождаемые такими грамматиками, называются *бесконтекстными* языками.

В грамматике *непосредственно составляющих* (НС-грамматике) или *контекстной грамматике* синтаксические правила имеют вид

$$vAw \rightarrow v\Phi w.$$

В НС-грамматике каждый шаг вывода состоит в замене одного вхождения символа A вхождением цепочки Ф, причем замена обусловлена наличием контекстов V и W. На каждом шаге вывода заменяется только один символ, поэтому с каждым выводом предложения ассоциируется так называемое дерево вывода (рисунок 1.12), строящееся следующим образом.

Корень дерева соответствует начальному символу. Каждому символу цепочки, на которую заменяется начальный символ на первом шаге вывода, ставится в соответствие узел дерева, и к нему проводится ветвь от корня. Для тех из полученных узлов, которые помечены вспомогательными символами, строится аналогичная конструкция и т. д.

Например, если грамматика содержит следующие правила: $I \rightarrow AAB$; $A \rightarrow a$; $B \rightarrow C$; $C \rightarrow c$ (a, b, c – основные символы; A, B, C – вспомогательные символы, I – начальный символ), то вывод I, AAB, aAB, aaB, aaC, aac можно представить в виде иерархической структуры (дерева) так, что I будет корнем; A, A, B – вершинами второго уровня; a, a, C – вершинами третьего уровня; c – вершиной четвертого уровня.

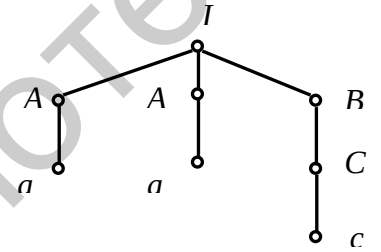


Рисунок 1.12 – Дерево вывода

Иногда в определение грамматики еще включают алфавит связей между буквами или словами, а также выделяют операции и формулы, которые используются в построении цепочки, являющейся предложением.

В теоретических исследованиях разных авторов имеется сильное взаимное проникновение теории автоматов и математической лингвистики, одним из важных объектов которой являются порождающие грамматики. В некотором смысле к классу порождающих автоматов можно отнести и любой вычислитель (человека, машину), который в процессе вычисления пишет цифры или другие знаки; эти знаки можно рассматривать как элементы, которые он порождает в процессе вычисления.

Математический вывод всегда осуществляется средствами заданной формальной системы. Сначала происходит кодирование входной информации в языке данной формальной системы (или машины), а затем эти коды перераба-

тываются в соответствии с правилами функционирования рассматриваемой системы (или машины). Процессы вывода в порождающих грамматиках аналогичны преобразованиям слов в ассоциативных исчислениях.

Общие подходы к построению алгоритмических языков. Алгоритмический язык является средством точного формулирования вычислительных процессов для их последующей реализации на вычислительных машинах.

Можно назвать *три основных круга задач и процессов*, для описания которых созданы алгоритмические языки:

а) алгоритмы численного анализа (расчеты по формулам и т. п.);

б) процессы обработки данных (обработка таблиц);

в) переработка символьной информации (обработка текстов, аналитические выкладки, моделирование интеллекта и т. п.).

Алгоритмический язык сам по себе может лишь использоваться как средство для публикации алгоритмов, удобное для однозначного восприятия человеком. Эффективность алгоритмического языка возрастет во много раз, если на ЭВМ имеются соответствующие средства трансляции (перевода) алгоритма на язык ЭВМ.

Графовые модели программы. Одним из наиболее ответственных этапов создания программного обеспечения (ПО) различного назначения является разработка спецификаций на будущую программу. Под спецификацией в общем виде понимается описание задачи, которую должна решать программа. В многообразии средств, применяемых в спецификациях, важное место занимают графовые средства – математические понятия, имеющие дело с относительно простыми видами связей между объектами и допускающие наглядное графическое изображение.

Наиболее распространенными подклассами графов, применяемых в спецификациях ПО, являются деревья, конечно-автоматные диаграммы, обобщенные диаграммы переходов, сети Петри, семантические сети и схемы алгоритмов.

Интерпретация перечисленных видов графов, их получение и практическое использование рассматриваются в соответствующих разделах специальных дисциплин. Остановимся лишь на особенностях построения графовой модели структуры программы, применяемой на этапах тестирования, анализа производительности и контроле структуры ПО. Для перечисленных целей достаточно представить программу ориентированным графом $G(X,U)$, множество вершин X которого соответствует линейным участкам программы, а дуги U указывают на связи между участками. При необходимости каждая вершина x_i графа взвешивается числом p_i, указывающим на длину участка (число команд, операций).

В корректно составленной программе через каждую вершину должен проходить хотя бы один путь, соответствующий варианту реализации. Эффективным методом выявления структурных ошибок по графу программы является использование матрицы достижимости $R = \| r_{ij} \|_{n \times n}$ графа, элементы которой формируются по правилу

$$q_{ij} = \begin{cases} 1, \text{ если вершина } x_j \text{ достижима из } x_i \text{ хотя бы по одному пути,} \\ 0, \text{ если вершина } x_j \text{ недостижима из } x_i \text{ ни по одному из путей.} \end{cases}$$

Матрица достижимости R может быть получена из матрицы смежности A путем моделирования движения по графу в направлении дуг либо аналитически по формуле

$$R = \sum_{i=1}^{n} A_i ,$$

где A_i – i-кратное произведение матрицы смежности A самой на себя.

Вершины x_s, x_f, соответствуют формальному началу и концу программы. При определении структурных ошибок по матрице R проверяются следующие соотношения, которые указывают на отсутствие лишних (недостижимых из начальной ни по одному из путей) и тупиковых компонент (из которых недостижима конечная вершина) в графовой модели программы:

$$\sum_{j=1}^{n} r_{sj} = n-1; \quad \sum_{j=1}^{n} r_{jf} = n-1. \tag{1.1}$$

При невыполнении соотношений (1.1) локализация ошибочных компонент выполняется проверкой условий: $r_{sj} = 0$ и $r_{jf} = 0$. Анализ матрицы достижимости графа программы (рисунок 1.13) показывает наличие лишних x_7, x_8, x_9 ($r_{s7} = r_{s8} = r_{s9} = 0$) и тупиковых компонент x_2, x_3, x_4, x_5, ($r_{2f} = r_{3f} = r_{4f} = r_{5f} = 0$).

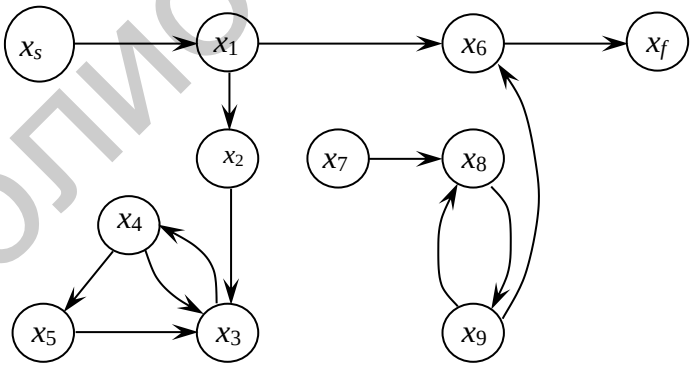


Рисунок 1.13 – Графовая модель программы

Графовая модель программы может использоваться для определения ее статистических характеристик и показателей надежности.

Ассоциативные исчисления и нормальный алгоритм Маркова. Изученные ранее свойства алгоритмов эмпирические. Выводы о них сделаны на основании опыта. Но сами по себе свойства не могут являться основой для точ-

ной математической формулировки понятия алгоритма. В связи с этим перейдем к более строгому описанию алгоритмов.

Удобными для этой цели оказались понятия и средства, накопленные в процессах преобразования различных слов как цепочек любых символов, что фактически использовалось нами при решении задачи поиска пути в лабиринте. На абстрактном уровне вычислительная машина всегда выполняет операции по переработке исходного текста (набор букв и чисел, задающих условие задачи) в некоторый заключительный текст (описание результата решения задачи). В машине буквы и цифры задаются в виде последовательностей нулей и единиц, поэтому решение задачи фактически сводится к переработке исходной конечной последовательности из нулей и единиц (условие задачи) в заключительную последовательность из нулей и единиц (результат). Понятие цепочки символов (слова) определяется через алфавит.

Назовем алфавитом любую конечную систему различных символов. Символы, составляющие алфавит, будем называть буквами. Например, {*a*, З, ?, *} – алфавит; *a*, З, ?, * – буквы.

Любая конечная последовательность букв некоторого алфавита называется словом в этом алфавите. Например, в алфавите *A*={*a, b, c*} словами будут последовательности *ab, ac, abac, bbbb* и т. п.

Рассмотрим два слова *H* и *M* в некотором алфавите *A.* Если *H* является частью *M,* то говорят, что *H* входит в *M.* Например, *H = ac; M = bbacab.*

Опишем процесс преобразования слов. Зададим в некотором алфавите конечную систему подстановок *H – M*, ..., *C – T*, где *H, M, ..., C, T* – слова этого алфавита.

Любую подстановку вида *H – M* можно применить к некоторому слову *K* этого алфавита следующим способом: если в слове имеется одно или несколько вхождений слова *H*, то любое из этих вхождений может быть заменено словом *M*, и наоборот, если имеется вхождение *M*, то его можно заменить словом *H.*

Например, *H = ab*, *M = bcb*, *K = abcbcbab*. Заменив в слове *K* слово *H* на *M*, можно получить такие слова: *bcbcbcbab* или *abcbcbbcb*. И наоборот, заменив *M* на *H*, получим *aabcbab* или *abcabab*.

Подстановка *ab – bcb* недопустима к слову *bacb,* так как ни *ab,* ни *bcb* не входят в это слово. К полученным с помощью допустимых подстановок словам можно снова применить допустимые подстановки и т. д.

Совокупность всех слов в данном алфавите вместе с системой допустимых подстановок называется ассоциативным исчислением. Чтобы задать ассоциативное исчисление, достаточно задать алфавит и систему подстановок.

Слова P_1 и P_2 в некотором ассоциативном исчислении называются смежными, если одно из них может быть преобразовано в другое однократным применением допустимой подстановки.

Последовательность слов P, P_1, P_2, M называется дедуктивной цепочкой, ведущей от слова *P к M,* если каждые из двух рядом стоящих слов этой цепочки смежные.

Слова P и M называются эквивалентными, если существует дедуктивная цепочка от P к M и обратно.

Пример:

$\{a, b, c, d, e\}$ – алфавит,

$ac – ca$; $abac – abacc$;
$ad – da$; $eca – ae$;
$bc – cb$; $eda – be$;
$bd – db$; $edb – be$.

– подстановки.

Слова *abcde* и *acbde* – смежные (подстановка $bc – cb$). Слова *abcde* и *cadbe* эквивалентны.

Ассоциативному исчислению можно поставить в соответствие бесконечный лабиринт, приняв каждое слово данного алфавита за площадку и соединив смежные площадки (слова) ребрами. Так как число слов бесконечно, то и лабиринт бесконечен.

Если слова P и M эквивалентны, то в построенном лабиринте это означает, что площадка, соответствующая M, достижима с площадки P.

Иногда рассматривается специальный вид ассоциативного исчисления, которое задается алфавитом и системой ориентированных подстановок типа $H \rightarrow M$. Стрелка означает, что разрешается производить подстановку лишь слева направо. Это исчисление соответствует бесконечному лабиринту, в котором разрешается движение только в одном направлении.

Для каждого ассоциативного исчисления своя специальная проблема слов: для любых двух слов требуется узнать, эквивалентны они или нет.

Это та же проблема достижимости, которая была рассмотрена в примере с лабиринтом, но лабиринт теперь стал бесконечным. Поэтому метод поиска, пригодный для конечного лабиринта, становится непригодным из-за невозможности в конечное время обследовать лабиринт.

Зная алгоритм поиска в конечном лабиринте, с его помощью можно лишь установить, можно ли одно из заданных слов преобразовать в другое применением допустимых подстановок не более k раз. Для этого можно построить все смежные слова с исходным, затем для каждого из полученных слов снова построить все смежные слова и т. д., всего k раз. Отсюда следует, что логическая задача о поиске пути в лабиринте может быть сформулирована на языке ассоциативного исчисления.

Вообще любой процесс вывода формул, математические выкладки и преобразования также являются дедуктивными цепочками в выбранном подходящим образом ассоциативном исчислении. Алгоритмические процессы также могут трактоваться как ассоциативное исчисление.

Естественно предположить, что построение ассоциативных исчислений может быть универсальным методом для задания детерминированного процесса «переработки» исходных данных, т. е. для задания алгоритма. Для этой цели необходимо уточнить понятие алгоритма.

Алгоритмом в алфавите A называется всякое общепонятное точное предписание, определяющее потенциально осуществимый процесс над словами из A, допускающий любое слово в качестве исходного и последовательно определяющий новые слова в этом алфавите.

Будем считать, что алгоритм в алфавите A задается в виде некоторой системы допустимых подстановок, дополненной общепонятным точным предписанием о том, в каком порядке и как нужно применять допустимые подстановки и когда наступает остановка.

Пример:

$A = \{a, b, c, d, e\}$ – алфавит,

$$\left.\begin{array}{l} cb - cc; \\ cca - ab; \\ cca - ab. \end{array}\right\} \text{ – система подстановок } B.$$

Условие дополняется указанием о способе применения подстановок. Для произвольного слова P разыскать первую подстановку, левая часть которой входит в P. Если такой подстановки нет, то процесс прекратить. В противном случае взять первую из найденных подстановок и сделать замену ее правой части вместо первого вхождения ее левой части в слово P. Затем полученное слово P_1 играет роль P и т. д.

Итак, схема подстановок вместе с указанием, как ими пользоваться, определяет алгоритм в алфавите A.

Рассмотрим применение системы подстановок B на конкретном примере для слов *babaac* и *bcacabc*:

Babaac → *bbcaaac* → остановка,

Bcacabc → *bcacbcac* → *bcaccccac* → *bcacabc* → процесс бесконечен, т. е. остановка не наступит, так как мы получили исходное слово.

Для того чтобы формально уточнить понятие алгоритма, советский ученый А. А. Марков ввел *понятие нормального алгоритма*. Опишем его.

Задается алфавит и схема подстановок B. Для произвольного слова P просматриваются формулы подстановок в том порядке, в каком они заданы в схеме B, и разыскивается формула с левой частью, входящей в P. Если такой формулы нет, то процесс обрывается. В противном случае берется первая из таких формул, и делается подстановка ее правой части вместо первого вхождения ее левой части в P, что дает слово P_1, с которым поступают аналогично. Обрывается процесс в двух случаях: во-первых, когда получим такое слово P_i, при котором ни одна из левых частей подстановок не будет входить в него; во-вторых, когда при получении P_i нам придется применять последнюю формулу.

Различные нормальные алгоритмы отличаются друг от друга лишь алфавитами и системами допустимых подстановок.

Приведем пример нормального алгоритма, описывающего процесс сложения чисел:

A = {1, +} – алфавит,

$$\left.\begin{array}{l} 1 + \rightarrow +1; \\ + 1 \rightarrow 1; \end{array}\right\} \text{– система подстановок } B.$$

$1 \rightarrow 1$.

Слово P = 11 + 11 + 111.

Переработаем слово P с помощью алгоритма Маркова, подчеркивая выбранные вхождения левых частей подстановок следующими линиями: штриховой (для первой подстановки $1 + \rightarrow +1$), сплошной (для второй подстановки $1 + \rightarrow 1$) и двойной (последней подстановки $1 \rightarrow 1$):

P = 1**1 +** 11 + 111,

P_1 = **1 +** 111 + 111,

P_2 = + 111**1 +** 111,

P_3 = + 11**1 +** 1111,

P_4 = + 1 **1 +** 11111,

P_5 = + **1 +** 111 111,

P_6 = + + 1111111,

P_7 = **+ 1**111111,

P_8 = **1**111111.

Нормальный алгоритм Маркова можно рассматривать как стандартную форму для задания любого алгоритма. В процессе построения теории численных алгоритмов выяснилось, что любой алгоритм можно свести к вычислению значений некоторой целочисленной функции при целочисленных значениях аргументов. Таким образом, теория численных алгоритмов (она тождественна понятию «теория вычислимых функций») стала универсальным аппаратом для исследования алгоритмических проблем.

Контрольные вопросы

1 Чем отличается автомат от других типов устройств для выполнения процессов?

2 Назовите отличительные особенности арифметических и логических операторов.

3 Зачем разработано и используется много средств для описания алгоритмов?

4 В чем заключается особенность численных и комбинаторных методов реализации алгоритмов?

5 Укажите на особенности использования ассоциативных исчислений в процессах доказательств и общего описания функционирования ЭВМ.

6 Чем принципиально отличается нормальный алгоритм Маркова от других алгоритмов?

7 Почему модель конечного автомата может использоваться для управления дискретными процессами?

8 Каковы причины использования формальных языков в вычислительной технике?

9 Чем отличаются понятия синтаксис и семантика формального языка?

10 В чем заключается преимущество нотации Бэкуса при ее использовании при описании различных формальных и алгоритмических языков?

11 В чем заключаются особенности использования дескрипторных систем с грамматикой и без грамматики?

12 Какие основные этапы необходимо выполнить при разработке алгоритмического языка и его транслятора?

Рекомендуемая литература [1, 4, 8–11, 16, 17, 19, 20, 23].

2 ИНТЕРФЕЙСЫ И ПЕРИФЕРИЙНЫЕ УСТРОЙСТВА

2.1 Контроль работы вычислительных систем

Основные подходы к обеспечению нормального функционирования ЦА. Решение задач контроля и надежного функционирования ЦА требует определенной избыточности его структуры, которая создается за счет схемотехнических или логических средств, а также использования дополнительной информации.

Так как позиционные системы не несут в себе избыточности информации при представлении чисел, то они в чистом виде не годятся для контроля.

Большой интерес для особо ответственных блоков ЦА представляют различные мажоритарные структуры, когда процесс вычислений продолжается лишь при наличии совпадения большинства результатов, выполняемых на нескольких (трех и более) идентичных устройствах.

Ошибка в работе ЦА может быть систематической – возникающей в результате отказа, и случайной – возникающей в результате сбоя (кратковременного нарушения правильной работы, происходящего, как правило, в результате разного рода помех: электрических, вибрационых и др.). Уменьшить вероятность ошибки можно двумя путями: увеличением надежности отдельных элементов и узлов, а также посредством выявления и исправления ошибок. Мы будем рассматривать в основном второй путь, который связан с введением избыточности в перерабатываемую информацию, что является основой аппаратурного контроля ЦА.

Возможность обнаружения ошибок базируется на простой идее, которая заключается в том, что в n-разрядном двоичном слове используются не все $N = 2^n$ возможных комбинаций (от 0…0 до 1…1 = $2^n - 1$). Некоторые комбинации являются запрещенными. Ошибки в двоичных словах заключаются в том, что цифры в неверных разрядах заменяются на взаимообратные. Если кодовая комбинация перешла в разряд запрещенных, то это означает наличие ошибки. Если же кодовая комбинация является ошибочной и переходит в разрешенную, то такая ошибка не обнаруживается. Чем больше избыточность, тем выше корректирующая способность применяемого кода.

Любой код, обладающий ненулевой избыточностью, называется корректирующим, независимо от того, исправляет он или только обнаруживает ошибки. Коды разделяются на посылочные и арифметические, так как задачи обнаружения (исправления) ошибок при передаче (хранении) информации и выполнении арифметических и логических операций в общем случае значительно различаются. Задача проектирования ЦА сводится к выбору такого кода, который при возможно меньшей избыточности обеспечивает нужную корректирующую способность.

Все методы диагностики ЦА делятся на два основных класса.

К первому классу относятся методы, при реализации которых диагностирование осуществляется в процессе функционирования ЦА. Их преимущество в

том, что неисправность немедленно обнаруживается или же происходит автоматическая корректировка результатов. Однако это влечет и дополнительные затраты на постоянно функционирующее избыточное оборудование.

Ко второму классу относятся методы, основанные на внесении временной избыточности – тестовое диагностирование. При их реализации ЦА находится в режиме тестирования и не функционирует по своему прямому назначению. Для этого случая готовятся заранее специальные последовательности входных воздействий – тесты, которые позволяют обнаружить заданные виды неисправностей.

Для реализации тестового контроля часто используется схема, основанная на применении эталонного ЦА. В этом случае нет необходимости в запоминании эталонных сигналов на выходе, что упрощает стендовое оборудование и дает больше возможности для поиска неисправностей.

Эффективность тестового диагностирования зависит от выбора входных воздействий, позволяющих обнаруживать определенный класс неисправностей.

Контроль достоверности передачи. Для повышения достоверности и качества передачи применяются групповые методы защиты от ошибок, избыточное кодирование и системы с обратной связью.

Из групповых методов получили широкое применение мажоритарный метод, реализующий принцип Вердана, и метод передач информационными блоками с количественной характеристикой блока.

Суть *мажоритарного метода* состоит в том, что каждое сообщение ограниченной длины передается несколько раз (чаще всего три раза), принимаемые сообщения запоминаются, а потом производится их поразрядное сравнение. Суждение о правильности передачи выносится по совпадению большинства из принятой информации методом «два из трех». Например, кодовая комбинация 01101 при трехразовой передаче была частично искажена помехами, поэтому приемник принял такие комбинации: 10101, 01110, 01001. В результате проверки по отдельности каждой позиции правильной считается комбинация 01101.

Передача блоками с количественной характеристикой предполагает передачу данных блоками с количественной характеристикой блока. Такими характеристиками могут быть: число единиц или нулей в блоке, контрольная сумма передаваемых символов в блоке, остаток от деления контрольной суммы на постоянную величину и др. На приемном пункте эта характеристика вновь подсчитывается и сравнивается с переданной по каналу связи. Если характеристики совпадают, считается, что блок не содержит ошибок. В противном случае на передающую сторону поступает сигнал с требованием повторной передачи блока. В современных телекоммуникационных вычислительных сетях такой метод получил самое широкое распространение.

Помехоустойчивое (избыточное) кодирование предполагает разработку и использование корректирующих (помехоустойчивых) кодов. Он применяется не только в телекоммуникационных сетях, но и в ЭВМ для защиты от ошибок при передаче информации между устройствами машины. Помехоустойчивое кодирование позволяет получить более высокие качественные показатели работы систем связи. Его основное назначение заключается в обеспечении малой

вероятности искажений передаваемой информации, несмотря на присутствие помех или сбоев в работе сети.

Существует довольно большое количество различных помехоустойчивых кодов, отличающихся друг от друга по ряду показателей и прежде всего по своим корректирующим возможностям. К числу наиболее важных показателей корректирующих кодов относятся:

- *значность* кода n, или длина кодовой комбинации, включающей информационные символы (m) и проверочные, или контрольные, символы (k): $n = m + k$ (значения контрольных символов при кодировании определяются путем контроля на четность количества единиц в информационных разрядах кодовой комбинации. Значение контрольного символа равно 0, если количество единиц будет четным, и равно 1 при нечетном количестве единиц);

- *избыточность* кода ($K_{изб}$), выражаемая отношением числа контрольных символов в кодовой комбинации к значности кода: $K_{изб} = k/n$;

- *корректирующая способность* кода ($K_{кс}$), представляющая собой отношение числа кодовых комбинаций L, в которых ошибки были обнаружены и исправлены, к общему числу переданных кодовых комбинаций M в фиксированном объеме информации: $K_{кс} = L/M$.

Выбор корректирующего кода для его использования в данной компьютерной сети зависит от требований по достоверности передачи информации. Для правильного выбора кода необходимы статистические данные о закономерностях появления ошибок, их характере, численности и распределении во времени. Например, корректирующий код, обнаруживающий и исправляющий одиночные ошибки, эффективен лишь при условии, что ошибки статистически независимы, а вероятность их появления не превышает некоторой величины. Он оказывается непригодным, если ошибки появляются группами. При выборе кода надо стремиться, чтобы он имел меньшую избыточность. Чем больше коэффициент $K_{изб}$, тем менее эффективно используется пропускная способность канала связи и больше затрачивается времени на передачу информации, но зато выше помехоустойчивость системы.

Рассмотрим основные механизмы обеспечения целостности передаваемой информации с помощью введения избыточности.

Механизм контрольных сумм. Контрольная сумма – некоторое значение, рассчитанное по набору данных путем применения определенного алгоритма и используемое для проверки целостности данных при их передаче или хранении. Под контрольную сумму может быть выделена некоторая совокупность n бит (один бит, полубайт, байт, несколько байт). Контрольной суммой некоторого массива будет являться величина, полученная путем деления с остатком суммы всех элементов массива на максимально возможное числовое значение контрольной суммы, увеличенное на единицу, т. е. 2^n. Значение контрольной суммы добавляется в конец блока данных непосредственно перед началом передачи или записи данных на какой-либо носитель информации. Впоследствии оно проверяется для подтверждения целостности данных. Вероятность ошибки равна $1/2^n$.

Функция, на основе которой выполняется преобразование исходного массива данных для получения контрольной суммы, называется «*хэш-функцией*», само преобразование – «*хэшированием*». Важная отличительная особенность хороших алгоритмов хэширования заключается в том, что генерируемые с его помощью значения настолько уникальны и трудноповторимы, что задача нахождения коллизий, т. е. ситуаций, когда разным наборам входных блоков данных соответствует один хэш, не решается вообще либо является чрезвычайно тяжелой как по ресурсоемкости, так и по производительности (практически не решается за приемлемое время).

Механизм четности (*parity*). Для передаваемых данных формируется специальный сигнал – *признак нечетного количества единиц* на линиях. В случае обнаружения нарушения четности в фазе данных сигнал ошибки вырабатывает приемник, для фазы адреса проверку четности выполняет целевое устройство, выставляется соответствующий бит в регистре состояния.

Пример – Для $X = 1011$ бит четности $e = 1$. Передается 10111, принята комбинация 10111, для которой бит четности $e^* = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 1 = e$, т. е. ошибки нет. Если принята комбинация 00111, то $e^* = 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 0 \neq e$, и есть ошибка.

Этот метод ограничен определением изменения состояния одиночного бита в байте. В случае изменения состояния двух бит, возможна ситуация, когда вычисление паритетного бита совпадет с записанным. В этом случае система не определит ошибку, и произойдет экстренная остановка системы. Так как приблизительно 90 % всех нерегулярных ошибок происходит именно с одиночным разрядом, проверки четности бывает достаточно для большинства ситуаций.

Недостаток этих двух названных выше способов контроля достоверности в том, что алгоритмы суммирования в них слишком просты. Для формирования надежной контрольной суммы можно увеличивать количество бит под контрольную сумму, а также необходим такой алгоритм расчета (хэширования), когда каждый новый байт может оказать влияние на любые биты контрольной суммы.

Проверка на четность может осуществляться по нескольким битам. В этом случае определяется четность или нечетность целого блока символов. Например, дополнительно формируется пересылаемое слово, составленное из битов четности разрядов пересылаемых слов, включая и их бит четности. Измененной версией этого способа является горизонтальный и вертикальный контроль четности, который выявляет некоторые двойные ошибки. Отличие в том, что начальные данные рассматриваются в виде матрицы, строки которой являются байтами информации. Контрольный разряд считает для каждой строки и каждого столбца матрицы.

Циклический избыточный код (*cyclic redundancy check – CRC*). Основная идея алгоритма *CRC* состоит в представлении сообщения в виде одного двоичного числа, делении его на другое фиксированное двоичное число и использовании остатка этого деления в качестве контрольной суммы. Получив сообщение, приемник может выполнить аналогичное действие и сравнить полученный остаток с контрольной суммой (переданным остатком).

В основе метода *CRC* лежит понятие *полинома,* или *многочлена*. Каждый бит некоторого блока данных соответствует одному из коэффициентов двоичного полинома $A(x)$. Например, полином двоичного числа 10110110 будет выглядеть следующим образом:

$$A(x) = 1 \cdot x^7 + 0 \cdot x^6 + 1 \cdot x^5 + 1 \cdot x^4 + 0 \cdot x^3 + 1 \cdot x^2 + 1 \cdot x^1 + 0 \cdot x^0 =$$
$$= x^7 + x^5 + x^4 + x^2 + x.$$

Для вычисления контрольного кода необходим еще один полином $G(x)$, называемый *порождающим* полиномом. Для каждой реализации алгоритма контроля *CRC* порождающий полином выбирается заранее. Полином $R(x)$ называется *контрольным кодом* полинома $A(x)$ при порождающем полиноме $G(x)$ степени r, если $R(x)$ является остатком от деления полинома $A(x) \cdot x^r$ на полином $G(x)$, т. е.

$$R(x) = (A(x) \cdot x^r) \bmod G(x).$$

Пример – Для порождающего полинома, соответствующего 11010, G(x) = $= 1 \cdot x^4 + 1 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x^1 + 0 \cdot x^0 = x^4 + x^3 + x$, *R(x) = x3 +x2 +x,* передается 101101101110, и если остаток от деления на 11010 равен нулю, то принята комбинация без изменения, иначе – есть ошибка.

Так же как и для контрольных сумм, контрольный код не занимает много места (обычно 8, 16 или 32 бита в *CRC*8, *CRC*16 и *CRC*32 соответственно), однако вероятность обнаружения ошибки существенно выше. Например, в отличие от метода контрольных сумм, метод *CRC* сможет обнаружить перестановку двух байт либо добавление единицы к одному и вычитание единицы из другого.

Алгоритм широко используется в аппаратных устройствах (дисковые контроллеры, сетевые адаптеры и др.) для верификации неизменности входной и выходной информации, а также для выявления ошибок при передаче данных по каналам связи.

Такая контрольная сумма проста в реализации и обеспечивает низкую вероятность возникновения коллизий.

Коррекция ошибок, или код исправления ошибок (*Error Correcting Code – ECC*)**.** Этот метод включает определение ошибки не только в одиночном разряде, но и двух, трех и четырех разрядах.

После передачи данных *ECC* результат сравнивается с рассчитанным подобно тому, как это происходит и в описанном выше методе проверки контрольных сумм. Основное различие состоит в том, что в проверке четности каждый бит связан с одним байтом, в то время как ECC связано с совокупностью байт. Если четность корректна для всех групп, то это свидетельствует об отсутствии ошибок. Если одно или более значений четности для переданного блока не верно, генерируется уникальный код, называемый синдромом, по которому можно идентифицировать переданный с ошибкой бит.

Введем основные понятия из области корректирующих кодов. Количество единиц в двоичной кодовой комбинации называется ее кодовым весом *W*. Например, если $X = 1{,}1010_2$, то $W = 3$.

Количество разрядов *d*, в которых не совпадают двоичные цифры в двух кодовых комбинациях, называется расстоянием между этими комбинациями. Оно обычно определяется методом поразрядного сложения по модулю два (исключающее ИЛИ) с последующим вычислением веса суммы:

$$X_1 = 1,010,\ X_2 = 0,1011,\ C = X_1 \oplus X_2 = 1,001,\ W = 2,\ d = 2.$$

Минимальным кодовым расстоянием $d_{\min}$ называется самое малое кодовое расстояние, возможное между двумя любыми кодовыми комбинациями из рассматриваемых множеств. В обычном двоичном коде *n*-разрядных чисел возможны кодовые расстояния от 1 до *n*, т. е. здесь $d_{\min} = 1$.

В общем случае, когда в кодовой комбинации могут появляться ошибки любой кратности $i \le n$ (одиночная, двойная, тройная и т. д.), для обнаружения некоторой ошибки требуется корректирующий код с минимальным расстоянием $d_{\min} = i + 1$. Следовательно, обнаружение одиночной ошибки можно обеспечить ценой минимальной избыточности – добавлением к слову всего одного контрольного разряда (контроль четности). Возникновение одиночной ошибки (или же ошибки с нечетным изменением количества разрядов) приводит к нарушению нечетности веса всей комбинации.

Таким образом, избыточность кодовой комбинации на один разряд позволяет обнаруживать все нечетные групповые ошибки и, как частный случай, одиночную ошибку.

В местах возможного возникновения кратковременных сбоев или выхода из строя отдельных элементов применяются так называемые коды с обнаружением ошибок, а для особо ответственных схем – коды с автоматическим исправлением ошибок. В таких кодах (обычно двоичных) имеются дополнительные контрольные разряды.

Идея двоичного кода с исправлением одиночной ошибки состоит в следующем. Пусть исходный (незащищенный) код имеет *m* двоичных разрядов. Выберем $n > m$ так, чтобы $\frac{2^n}{n+1} \ge 2^m$. Полученный защищенный код будет иметь $k = n - m$ контрольных разрядов, причем $2^k \ge m + k + 1 = n + 1$. Каждому из *n*-разрядов присваивается номер от 1 до *n*. Далее для любого значения защищенного (*n*-разрядного) кода составляется *k* контрольных сумм $S_1, S_2, \ldots, S_k$ по модулю 2 значений специально выбранных разрядов этого кода. Для S_i выбираются разряды, для которых двоичные коды номеров имеют в *i*-м разряде единицу. Для суммы S_1 это будут, очевидно, разряды с номерами 1, 3, 5, 7, …, для суммы S_2 – разряды с номерами 2, 3, 6, 7, 10, 11, … и т. д.

При любом исходном коде контрольные разряды могут быть выбраны так, чтобы все контрольные суммы были равны нулю. Проверочный (двоичный) код

$S = S_k \ldots S_1$ при этом будет нулевым. При таком условии защищенный код называется *правильным*. При возникновении одиночной ошибки в этом коде в любом (безразлично, основном или контрольном) разряде проверочный код будет отличен от нуля. При этом в силу способа выбора контрольных сумм значение проверочного кода S будет представлять собой двоичный код номера разряда защищенного кода, в котором возникла ошибка. Для исправления этой ошибки достаточно изменить значение указанного разряда на противоположное.

Примером корректирующих кодов являются код Хемминга и код Рида – Соломона.

Код, в котором в качестве контрольных берут разряды 1, 2, 4, …2^n, называется *кодом Хемминга*. Код Хемминга может либо исправлять одиночные ошибки (при гипотезе, что ошибки более высокой кратности невозможны), либо обнаруживать любые ошибки, не кратные трем. Тройные ошибки считаются маловероятными, и поэтому иногда говорят, что *код Хэмминга позволяет обнаруживать одиночные и двойные ошибки*.

Покажем на примере, как ликвидируется одиночная ошибка с помощью кода Хемминга при передаче четырехразрядного двоичного слова X = 0110 (m = 4).

Построение кода выполняется в три этапа:

1 Вычислим необходимый размер (n) разрядной сетки защищенного слова и количество контрольных разрядов (k). Минимальное значение n, при котором выполняется неравенство $\frac{2^n}{n+1} \geq 2^m = 2^8$, равно 7, $k = 7 - 4 = 3$.

2 Определим номера контрольных разрядов в защищенном слове: $2^0 = 1$, $2^1 = 2$, $2^2 = 4$.

3 Построим защищенное слово в семиразрядной сетке. Для этого запишем исходное слово в порядке следования его разрядов, пропуская места контрольных разрядов:

1	2	3	4	5	6	7
001	010	011	100	101	110	100
		0		1	1	0

Определим значения защищенных разрядов исходя из равенства сумм S_i нулю. В сумму S_1 должны войти значения разрядов, расположенных на нечетных местах: 1, 3, 5, 7 = 001, 011, 101, 111. Чтобы сумма S_1 была равна нулю, входящий в нее 1-й контрольный разряд должен быть равен единице, т. е. $S_1 = 1 \oplus 0 \oplus 1 \oplus 0 = 0$.

Аналогично отыскиваются и другие контрольные разряды.

В S_2 войдут разряды (2, 3, 6, 7) = (010, 011, 110, 111), $S_2 = 1 \oplus 0 \oplus 1 \oplus 0 = 0$.

В S_3 войдут разряды (4, 5, 6, 7) = (100, 101, 110, 111), $S_3 = 0 \oplus 1 \oplus 1 \oplus 0 = 0$.

По результатам вычисления контрольных разрядов и значениям разрядов передаваемого слова строим защищенное слово:

1	1	0	0	1	1	0

В месте приема переданного слова снова вычисляются контрольные суммы S_1, S_2, S_3, но уже при известных значениях заштрихованных контрольных разрядов. При равенстве всех сумм нулю считается, что передача произошла правильно. Предположим, что в процессе передачи произошло искажение только одного разряда (неважно какого: контрольного или основного). Например, предположим, что неправильно передан шестой разряд. Это означает, что вместо единицы в 6-м разряде появился нуль. Тогда после вычисления контрольных сумм получим $S_1 = 0$, $S_2 = 1$, $S_3 = 1$. Располагаем эти суммы в порядке убывания их номеров и получаем $S = S_1S_2S_3 = 011$. Это двоичное число соответствует номеру 6. Тогда можно автоматически исправить ошибку в шестом разряде, заменив пришедший нуль на едуницу. После исправления ошибки контрольные биты отбрасываются.

Коды Рида – Соломона (*Reed – Solomon code, RS-code*) позволяют обнаруживать и исправлять ошибки в блоках данных. Элементами кодового вектора являются не биты, а группы битов (блоки, в частности, байты).

Кодирование информационного блока может быть выполнено разными способами, при этом исходный блок будет либо просто дополнен некоторым проверочным блоком (*систематическое кодирование*), либо перекодирован (*несистематическое кодирование*).

В кодах Рида – Соломона сообщение представляется в виде набора символов некоторого алфавита, в качестве которого используются элементы поля Галуа. *Поле Галуа* является конечным полем (обычно обозначается *GF*(*N*), где *N* – размерность поля, равная количеству его элементов). Арифметические действия над элементами конечного поля дают результат, который также является элементом этого поля.

Количество чисел в поле должно являться простым числом в любой натуральной степени, однако в случае простых кодов Рида – Соломона размерность поля – простое число в степени 1. Например, для поля Галуа размерностью 7, т. е. *GF*(7), все математические операции будут происходить с числами 0, 1, 2, 3, 4, 5, 6.

Сложение, вычитание, умножение и деление в полях Галуа выполняется по модулю N.

Примеры:

$4 + 5 = 9 \bmod 7 = 2$;

$3 – 5 \bmod 7 = 5$;

$3 \cdot 5 \bmod 7 = 1$;

$2^4 \bmod 7 = 2$;

$5 : 6 \bmod 7 = 2$ (верно, так как $6 \cdot 2 \bmod 7 = 5$).

Полезное свойство обнаруживается в полях Галуа при возведении в степень. М можно заметить, что значения степеней чисел 3 либо 5 в выбранном поле Галуа *GF*(7) – это все элементы текущего поля Галуа, кроме 0.

Например:

$3^0 = 1$, $3^1 = 3$, $3^2 = 2$, $3^3 = 6$, $3^4 = 4$, $3^5 = 4$, $3^6 = 1$;

$5^0 = 1$, $5^1 = 5$, $5^2 = 4$, $5^3 = 6$, $5^4 = 2$, $5^5 = 3$, $5^6 = 1$.

Такие числа называются *примитивными элементами*. В кодах Рида – Соломона обычно используется самый большой примитивный элемент выбранного поля Галуа. Для $GF(7)$ он равен 5.

Данные обрабатываются порциями по m бит, которые принято называть символами. Как правило, порция представляет собой байт, т. е. $m = 8$. При построении кода Рида – Соломона задается пара чисел n, k, где n – общее количество символов (длина блока), а k – количество информационных символов, остальные $n – k$ символов представляют собой избыточный код, предназначенный для восстановления ошибок (рисунок 2.1). Такой код будет иметь расстояние Хэмминга $D = n – k + 1$.

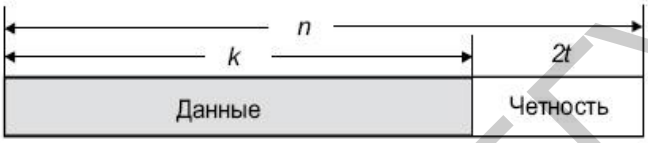


Рисунок 2.1 – Структура кода Рида – Соломона

В соответствии с теорией кодирования код, имеющий расстояние Хемминга $D = 2t + 1$, позволяет восстанавливать t ошибок. Таким образом, если в кодовое слово случайно внести $t = (n – k) : 2$ ошибок (т. е. просто произвольно заменить значения t символов любыми значениями), то окажется возможным обнаружить и исправить эти ошибки.

Пусть $t = 1$, $m = 8$. Длина блока равна $n = 2^8 – 1 = 255$ символов или 2040 бит. Длина контрольной части – 2 символа, или 16 бит. Код обнаруживает любой пакет ошибок, длина которого не более 16 бит, и исправляет ошибки в пределах одного байта.

Исходное сообщение при кодировании Рида – Соломона представляется полиномом $p(x)$ степени $k – 1$, имеющем k коэффициентов.

Порождающий многочлен Рида – Соломона $g(x)$ строится следующим образом:

$$g(x) = (x + a^1)(x + a^2)\ldots(x + a^{D-1}),$$

где a – примитивный элемент.

В несистематическом коде закодированное сообщение $C(x)$ получается как произведение исходного сообщения на порождающий многочлен:

$$C(x) = p(x) \cdot g(x).$$

Систематический код строится аналогично *CRC*, когда $p(x)$ сдвигается на k коэффициентов влево, а затем прибавляется остаток от его деления на $g(x)$:

$$C(x) = p(x) \cdot x^{n-k} + p(x) \cdot x^{n-k} \bmod g(x).$$

Пример – Пусть нам нужно передать кодовое слово $X = (3,1)$ с возможностью исправить две ошибки ($k = 2, t = 2$). Для этого нужно взять $2t = 4$ избыточных символа. Полученное значение $X = (3, 1, 0, 0, 0, 0)$ в виде полинома имеет вид $X = 3 \cdot x^0 + 1 \cdot x^1 + 0 \cdot x^2 + \ldots = 3 + x$. В поле Галуа $GF(7)$ с примитивным элементом 5 получаем код $c(4, 1, 0, 2, 5, 6)$, где $c_i = C(z^i) = 3 + 1 \cdot z^i$. Например, $c_2 = C(z^2) = 3 + 1 \cdot z^2 = 0$.

Закодированное сообщение $C(x)$ без остатка делится на порождающий многочлен g(x):

$$C(x) \bmod g(x) = 0.$$

В случае если закодированное сообщение будет изменено, то это равенство окажется нарушенным.

Декодировщик последовательно вычисляет синдромы ошибок, строит соответствующий полином ошибок и находит корни данного полинома (они указывают положение искаженных символов в кодовом слове), по которым определяет характер ошибок и исправляет их.

Устранение ошибок с помощью корректирующих кодов (такое управление называют *Forward Error Control*) реализуют в симплексных каналах связи. В дуплексных каналах достаточно применения кодов, обнаруживающих ошибки (*Feedback or Backward Error Control*), так как сигнализация об ошибке вызывает повторную передачу от источника.

2.2 Защита информации и электронная подпись документов

Защита информации и создание безбумажных систем документирования стали тесно переплетаться на основе общих средств используемого математического аппарата.

С каждым годом повышается роль средств, обеспечивающих информационную безопасность различных технологий, использующих компьютеры. Математической основой этих технологий является теория криптографии (тайнописи). Ее истоки уходят в древние времена. На первых порах часто использовались довольно простые шифры замены знаков (каждого в тексте или группы) путем подстановки другого знака методом, который был известен получателю и допускал обратную операцию. Широко известен среди них метод Цезаря, который задавал в каждом сообщении шаг сдвига для буквы в тексте: вместо истинной буквы записывалась другая, отстоящая от нее в алфавите на заданный шаг в закольцованном алфавите. Например, в алфавите {*A*, B, C, Д, E, *F*, *R*}, истинное сообщение *ARFA*, закодированное с шагом 3, будет выглядеть как ДСВД.

Секрет такого шифра в состоянии раскрыть даже школьник. Поэтому теория кодирования пошла путем усложнения трудностей в прочтении закодированных текстов таким образом, чтобы, даже владея методом кодирования, но, не имея к нему ключа (в нашем примере это шаг сдвига), с помощью современной техники за практически разумное время (жизнь человека) нельзя было найти ключ к чтению текстов.

Компьютерный этап криптографии начался с трудов американского математика К. Шеннона. Он выделил два общих принципа, использующихся в практическом шифровании:

- рассеивание (влияние одного знака на шифротекст);
- перемешивание (создание трудностей в использовании статистических свойств из-за повтора знаков или характера сообщений, например, стандартные донесения с датами, наименованиями типов вооружений и их количество и т. п.).

Практические трудности здесь стали заключаться в том, что надо не только затруднить распознание (раскрытие) истинного сообщения, но и само шифрование сделать достаточно легким.

Таким образом, для решения этих задач пришли к идее сами ЭВМ использовать в шифровании и дешифровании текстов под управлением человека, который остался лишь носителем небольшой секретной информации (ключей).

Постепенно созрели идеи и о необходимости использования государственных стандартов шифрования, чтобы уверенно и безопасно пользоваться программами и алгоритмами, прошедшими экспертизу в специальных государственных органах. В мире первым таким стандартом стал *DES* (*Data Encryption Standart*, США). Он опирался на открытый для всех алгоритм шифрования, основанный на реализации принципов рассеивания и перемешивания. В нем открытый текст, криптограмма и ключ представлялись в виде двоичных последовательностей длиной соответственно 64, 64 и 56 бит. Спустя более 30 лет после его опубликования с помощью специального компьютера «Большой взлом» (*Deep Crack*) за 56 часов при переборе $18 \cdot 10^{16}$ комбинаций удалось осуществить взлом контрольной шифровки. В связи с этим на смену *DES* пришел *AES* (*Advanced Encryption Standart* – улучшенный стандарт шифрования), в котором для перебора всевозможных ключей (2^{256}) потребуется столько операций, которые практически нельзя осуществить даже на специальном комплексе из многих суперкомпьютеров за приемлемое время.

Стандарты аналогичного типа стали использоваться в качестве ГОСТов и в других странах (например, в России). Они относятся к классу систем с симметричным алгоритмом шифрования, т. е. один и тот же ключ используется для шифрования и дешифрования текста. Тут возникает другая сложность: как передать адресату ключ по открытым каналам связи без риска его перехвата. Поэтому дальнейшее усовершенствование криптосистем пошло в направлении создания двухключевых систем: их первый ключ публикуется для применения всеми пользователями при расшифровании данных, а для их шифрования имеется секретный второй ключ, который нельзя получить из первого, что сделало сам зашифрованный текст равнозначным подписанному. Суть математических

преобразований для этих целей опирается на теорию функций с «лазейкой», когда $y = f(x)$ легко вычисляется, а обратная функция $x = f(y)$ − невычислимая без знания дополнительной информации (лазейки). Простейший пример такого типа представляет вычисление целочисленной функции $y = x \bmod n$, когда ее результатом является остаток от деления целого числа на целочисленный модуль n. Например, $y = 12 \bmod 7 = 5$, но обратная операция нахождения x как функции от остатка 5 не выполняется однозначно (один и тот же остаток при делении на 7 могут дать 5, 12, 19, …). Неизвестное x можно вычислить лишь при знании «лазейки», например, номера числа в ряду 5, 12, 19, … .

Покажем реализацию одноключевой системы такого типа с открытой передачей ключей на основе функций с лазейкой. Для открытого распространения ключей Диффи и Хелман предложили использовать функцию

$$F(x) = a^x \bmod p,$$

где p – большое простое число;

x – произвольное натуральное число из множества $\{1, 2, ..., p-1\}$;

a – целое число из множества {2, 3, …, p, для которого выполняется требование, чтобы все степени a от 1 до $(p – 1)$ в произвольном порядке по модулю p дали все числа из множества $\{1, 2, ..., (p-1)\}$.

Например, при модуле $p = 7$ можно выбрать $a = 3$:

$f(1) = 3^1 (\bmod 7) = 3, \quad f(2) = 3^2 \bmod 7 = 2, \quad f(3) = 3^3 (\bmod 7) = 6, \quad f(4) = 3^4 (\bmod 7) = 4,$

$f(5) = 3^5 (\bmod 7) = 5, \quad f(6) = 3^6 (\bmod 7) = 1).$

Предполагается, что всем пользователям сети известны a и p. Пользователь i случайным образом выбирает число x_i (свою лазейку), т. е. секретное число, из множества $\{1, 2, ..., (p-1)\}$, известное только ему. Далее он вычисляет $y_i = a^{x_i} (\bmod p)$ и помещает его в открытый для доступа всех пользователей сети справочник. При желании установить секретную связь с пользователем j он берет из справочника его число y_j и с помощью своего секрета x_i для обмена сообщениями с j вычисляет ключ $Z_{ij} = (y_i)^{x_i} (\bmod p)$. После установления контакта аналогичную работу проделывает пользователь j, который с помощью своего секретного числа x_j вычисляет $Z_{ji} = (y_i)^{x_j} \bmod p$. Ограничения, наложенные на выбор a, обеспечивают получение равенства $Z_{ij} = Z_{ji}$, т. е. одинаковых ключей для обмена сообщениями. В самом деле $Z_{ij} = y_j^{x_i} \bmod p = (a^{x_j})^{x_i} \bmod p = a^{x_j x_i} \bmod p$ и $Z_{ji} = a^{x_i x_j} \bmod p$.

Пример – $p = 7, a = 3, x = \{1, 2, 3, 4, 5, 6\}$.

$x_i = 3$ (секрет i), $y_i = 3^3 \bmod 7 = 6$.

$x_j = 4$ (секрет j), $y_j = 3^4 \bmod 7 = 4$.

$$Z_{ij} = 4^3 \bmod 7 = 1.$$

$$Z_{ij} = 6^4 \bmod 7 = 1296 \bmod 7 = 1.$$

Цифра 1 может означать некоторую функцию, которая используется при кодировании, страницу в заранее разосланных пользователям материалах и т. д.

Недостаток описанной криптосистемы с открытым распространением ключей состоит в том, что она требует абсолютного доверия партнеров по связи друг к другу, так как в этой одноключевой системе они могут изменять переданный текст. Поэтому она непригодна, например, для не доверяющих друг другу удаленных абонентов. Вычисление остатков *x* при делении целых чисел на модуль *y* можно выполнять с помощью функции mod (*x*, *y*).

Решение задач, связанных с признанием электронной подписи, идентификации и аутентификации, тесно связаны с системой шифрования *RSA*, хэш-функциями и исследованиями по развитию такого рода механизмов.

Алгоритм *RSA* – компонент ряда криптосистем для цифровой подписи. Алгоритм *RSA* обеспечивает высокую степень защиты, поскольку в нем в качестве модуля используется произведение двух больших простых целых чисел.

Рассмотрим, как построить двухключевую систему с использованием алгоритма *RSA* и выполнить в ней операцию шифрования и дешифрования трех первых букв фамилии студента (при количестве букв меньше трех, недостающие буквы берутся из имени). Пара простых чисел *P* и *Q* выбирается из диапазона ближайших к количеству букв в фамилии и имени студента.

Например, Петров Владимир, *P* (5 или 7), *Q* (7 или 11). Методом испытаний подбирается также ближайшая пара чисел *E* и *D*.

В нашем случае это могут быть *P* = 5, *Q* = 7.

В случае неудачных сочетаний из названного диапазона берутся рядом другие ближайшие простые числа, например, *P* = 5, *Q* = 13.

В системе *RSA* каждый пользователь имеет свой ключ шифрования. Ключи дешифрования известны всем, а шифрующий ключ держится в секрете. Криптографические системы типа *RSA* подходят для реализации цифровой подписи, применяемой в системах электронных платежей и при передаче сообщений с помощью устройств телесвязи.

К недостаткам системы *RSA* и аналогичных ей относят ее существенно более низкое быстродействие и потребность в более длинных ключах. Наиболее эффективные реализации *RSA* характеризуются скоростью шифрования порядка нескольких тысяч бит в секунду. Тогда как аналогичные реализации более простых систем шифруют несколько миллионов бит в секунду. В связи с этим наиболее целесообразным применением *RSA* считается организация обмена секретными ключами, необходимыми для обеспечения безопасности в сетях связи.

Основная проблема для системы *RSA* – генерация соответствующей пары ключей.

Для генерации используется следующая процедура:

1 Выбрать 2 простых числа *P* и *Q*.

2 Найти произведение $N = P \cdot Q$ и число $L = (P - 1)\cdot(Q - 1)$.

3 Выбрать случайное число D – такое, что оно должно быть взаимно простым с числом L.

4 Определить другое число E – такое, что $(E \cdot D) \bmod L = 1$.

5 Как только все числа найдены, мы имеем секретный ключ E и открытый ключ – пару чисел D и N.

Теперь при шифровании сообщения его разбивают на блоки M. В результате шифрования для каждого блока M получим число

$$C = (M^E) \bmod N.$$

При дешифрации получаем расшифрованные (исходные) блоки:

$$M^* = (C^D) \bmod N.$$

Рассмотрим это на примере алфавита из букв {Л,О,Я} = {1, 2, 3} для передачи текста «ОЛЯ» (или 2, 1, 3). Цифровые обозначения букв или блоков обязательны, так как метод основывается на обработке натуральных чисел:

1 Выберем $P = 3$ и $Q = 11$.

2 Найдем $N = P \cdot Q$, $N = 33$; $L = (P - 1) \cdot (Q - 1)$, $L = 20$.

3 Выберем D взаимно простое с L: $D = 3$.

4 Выберем E такое, что $(E \cdot D) \bmod L = 1$, например, $E = 7$; действительно, $(7 \cdot 3) \bmod 20 = 1$.

5. Тогда открытый ключ $\left.\begin{matrix} D = 3, \\ N = 33 \end{matrix}\right\}$ – секретный $E = 7$.

Производим шифрацию:

$$C1 = (M1^B) \bmod N: \quad \begin{matrix} C3 = (3^7) \bmod 33 = 9, \\ C2 = (1^7) \bmod 33 = 1, \\ C1 = (2^7) \bmod 33 = 29. \end{matrix}$$

Получается зашифрованный текст (29, 1, 9).

Расшифровка:

$M1^* = (C1^D) \bmod N$: $(29^3) \bmod 33 = 2$,

$M2^* = (C2^D) \bmod N$: $(1^3) \bmod 33 = 1$,

$M3^* = (C3^D) \bmod N$: $(9^3) \bmod 33 = 3$.

В результате мы получили исходный текст: 2, 1, 3.

Остается только добавить, что для получения достаточно стойкой шифровки необходимо брать очень большие простые числа.

Выполнение соотношения $(ED) \bmod L = 1$ позволяет использовать этот факт для проверки подлинности подписи без знания секретного ключа E с помощью аппарата хэш-функций.

В практической работе необходимо идентифицировать автора электронного документа и предприятие не по особенностям подписи и печати (например, по образцам подписей и печатей в банковской карточке клиента), а по наличию у него электронного ключа для подписывания документов. В этом случае конкретное число-подпись под данным документом в фиксированное время, может сделать только законный обладатель ключа (E).

Процедура электронной подписи включает в себя два этапа:

– подписывание (вычисление параметров подписи, зависящих от текста конкретного документа, один из которых (E) хранится в секрете);

– проверка получателем с помощью несекретных параметров (D, N) подлинности сообщения (подписи).

Сообщение шифруется по алгоритму *RSA*, где E подбирается и известно только отправителю, а D, N знает и получатель. Получатель должен иметь возможность с помощью открытого ключа проверить подлинность сообщения. Для этой цели в сообщение добавляется еще одно число, которое является результатом вычисления хэш-функции $h(T)$, зависящей от текста T.

Обеспечение подлинности сообщений. К хэш-функции предъявляется ряд требований:

- невозможность (или возможность за очень длительное время) найти по значению $h(T)$ само T (т. е. требуется построить практически необратимую функцию);

- для заданного T нельзя найти такое T', чтобы $h(T) = h(T')$;

- вообще нельзя найти пару различных слов T и T' такую, что $h(T) = h(T')$;

- сообщение T (например, текст договора, платежного поручения и т. п.) по заданной функции сжимается в целое число $m = h(T)$, причем $1 < h(T) < N$.

Число m позволяет с помощью открытого ключа констатировать подлинность документа.

С этой целью автор документа с помощью своего секретного ключа E получает второй параметр подписи $S = (m^E) \bmod N$. Параметры m и s подставляются в текст сообщения на место подписи и печати. Все сообщение по телекоммуникационным каналам передается получателю. Он проверяет правильность цифровых параметров m и S, исходя из знания функции $h(T)$, полученного символьного объема зашифрованного сообщения (T_1) и «лазейки» для вычисления $h(T)$, $h(T_1)$.

Проверка параметра S производится путем идентификации условия:

$$(S^D) = m \bmod N.$$

Математически доказано, что результат проверки m и s будет положительным в том случае, когда в их формировании использовался секретный ключ E, соответствующий открытому ключу D. Вероятность расшифровки секретного ключа E по открытым параметрам S, m, D и N считается ничтожно малой из-за

затрат времени на решение задачи взлома системы по сравнению со временем полезного действия сообщения.

Покажем в упрощенном варианте проверку подписи сообщения T = ОЛЯ с дополнением его параметрами m и S. В качестве функции хэширования $h(T)$ возьмем произведение сумм из двух элементов для каждого шифруемого знака: его кода по ходу текста с простым числом (2, 3, 5, 7, …) по порядку следования. В нашем случае их позиции О = 1, Л = 2, Я = 3, а коды Л = 1, 0 = 2, Я = 3, т. е. П = (2 + 2)(1 + 3)(3 + 5) = 128. Чтобы получить m, вычисляем его так:

$$m = 128 \bmod 33 = 29.$$

Можно убедиться, что эта функция удовлетворяет требованиям к ней по крайней мере для любого сообщения из трех разных букв. Это обнаруживается при вычислении всех возможных шести разнобуквенных сообщений T: ОЛЯ – 29, ОЯЛ – 12, ЛОЯ − 21, ЛЯО − 27, ЯОЛ − 18, ЯЛО − 8, т. е. нет равных $h(T)$.

Следует заметить, что с ростом длины сообщения может оказаться, что такая функция не удовлетворяет требованию, когда два сообщения разной длины имеют одинаковое значение, т. е. $h(T) = h(T')$. Чтобы избежать такого явления, можно разбивать сообщения на блоки заранее ограниченной длины. Поэтому выбор хорошей функции $h(T)$ является очень трудной задачей, и ее решением занимаются специалисты, которые разрабатывают стандарты шифрования. В описанном нами примере ограничимся лишь демонстрацией процедуры признания подписи.

Текст T = ЛОЯ, который получает партнер, позволяет проверить подлинность подписи по параметрам m, S и D = 3 (открытая часть ключа), S приходит с текстом отправителя ($S = m^E \bmod N = 21^7 \bmod 33 = 21; \quad m = 21 < 33$). Проверка подлинности сообщения:

$$S^D = m \bmod N; \quad 21^3 = 21 \bmod 33,$$

т. е. результат проверки положителен и подпись подлинна.

Кроме того, если получатель тоже знает, как вычислить функцию хэширования от текста, то по прочитанному тексту он может ее вычислить.

В настоящее время в Республике Беларусь выпущен предварительный стандарт СТБ П34, 101.25-2008 для электронной подписи, в котором алгоритм *RSA* − один из трех рекомендуемых для применения.

Защита от несанкционированного доступа к информации, циркулирующей или хранящейся в различных, часто удаленных друг от друга узлах сети, является одной из самых сложных. Это связано с тем, что необходимо решать не только задачу о допуске удаленного пользователя к разрешенным ему объектам для доступа в рамках его полномочий, но и определении личности пользователя, чтобы вместо него не работал за терминалом сети или самовольно подключенным терминалом злоумышленник. Для решения такого рода задач со-

зданы специальные инструменты – *сетевые протоколы*. Их составной частью являются элементы для идентификации и аутентификации пользователей.

Идентификация пользователя, как правило, сводится к определению его полномочий на доступ к данным и устройствам по фамилии и паролю. Аналогичная задача может решаться и по отношению к удаленному терминалу на предмет его законного подключения к сети.

Аутентификация является более сильной формой опознания личности при совершении ответственных операций, так как в современных системах требуется предъявлять физические характеристики пользователя (палец, ладонь, произношение некоторых из заданных слов и т. п.).

2.3 Назначение периферийных устройств и интерфейсов

Классификация периферийных устройств и интерфейсов. В составе ПЭВМ (или узла более сложной системы) можно выделить вычислительное ядро и его периферию. Ядро обычно состоит из АЛУ, выполняющего также некоторые из задач управления, и ОЗУ. Все устройства, не входящие в вычислительное ядро (ядра), относятся к *периферийным*. Они могут располагаться снаружи/внутри корпуса ЭВМ, а также входить в состав основных микросхем системы.

Основная задача периферийного устройства (ПУ) – поставка данных на обработку и хранение, а также вывод их за пределы вычислительного ядра. Данная задача охватывает процессы преобразования информации некоторой физической природы в электрические сигналы, пригодные для хранения и обработки ядром системы, такие как оцифровка и преобразование данных в электрическую форму (из оптической, механической, электромагнитной и т. д.), сохранение на внешних носителях, изготовление «твердой копии» на бумаге, передача по *каналам связи*, отображение в графической форме на экране и т. д. Соответственно, основными ПУ являются *устройства ввода/вывода* и *хранения данных*. Можно также выделить отдельный класс устройств *управления и обслуживания системы*, в том числе сетевые и коммуникационные устройства, которые по назначению неправомерно относить к периферийным, но по принципу действия они являются периферийными.

Средства (аппаратные и программные), используемые для соединения двух компонентов или систем, называются *интерфейсом*. Второе понятие, которое тесно связано с понятием интерфейса (иногда они отождествляются), – это *компьютерная шина*, *магистраль*, т. е. подсистема, которая передает данные между функциональными блоками и представляет собой совокупность сигнальных линий, которые имеют определенные электрические характеристики и протоколы передачи информации.

По *способу кодирования и передачи данных интерфейсы делятся на следующие*:

- параллельные, характеризующиеся разрядностью (количеством бит одного машинного слова, передаваемых в один момент времени);

- последовательные, характеризующиеся количеством агрегированных каналов передачи данных (количеством бит различных машинных слов, передаваемых одновременно, не обязательно синхронно и с одной скоростью).

По *способу синхронизации источников или получателей данных* основными режимами передачи информации являются *асинхронная и синхронная передачи данных*, которым соответствуют *асинхронные и синхронные протоколы*. Данные передаются последовательно бит за битом, биты образуют байты (определенные символы), а байты образуют блоки (кадры) передаваемых данных.

При *синхронной передаче* поддерживаются постоянные интервалы между очередными квантами информации в процессе передачи всего сообщения. Для этого могут использоваться специальные тактовые синхросигналы на отдельной сигнальной линии либо специальные последовательности бит (символы синхронизации, формируемые передатчиком.

Передачу называют *асинхронной*, если интервал между передачей кадров непостоянен (информационные сигналы могут появляться в произвольный момент времени). Для выделения кадра в его начале и конце записывают старт-стопные последовательности бит.

При обмене данных на физическом уровне единицей информации является бит, поэтому средства физического уровня всегда поддерживают побитовую синхронизацию между приемником и передатчиком.

В рамках этих двух основных режимов может обеспечиваться выполнение транзакций с *адаптивной* настройкой скорости передачи как приемников, так и источников, а также изохронные транзакции (с гарантированной запрошенной пропускной способностью).

По направлению передачи интерфейсы бывают:

- однонаправленные (симплексные);
- двунаправленные (дуплексные);
- с возможностью изменения направления передачи (полудуплексные);
- мультиплексные (в случае связи нескольких компонентов в каждый момент времени связь может быть осуществлена между парой абонентов в любом, но единственном направлении, используется тогда, когда для передачи данных применяется только одна линия передачи).

Современные интерфейсы обычно обеспечивают дуплекс за счет работы двух встречно направленных симплексных каналов. При этом зачастую в одну сторону передаются данные, а в другую – информация о состоянии ПУ и управление передачей.

По организации связей (топологии) интерфейсы подразделяют на следующие:

- магистральные, или шинные;
- радиальные, или звездообразные;
- кольцевые;
- древовидные, или иерархические;
- радиально-магистральные.

По физическому явлению, используемому для кодирования информации, интерфейсы бывают:

- электрические (ток или напряжение);
- оптические (оптические сигналы, оптическое волокно);
- беспроводные (радио, лазерное или инфракрасное излучение, ультразвук).

Сигналы, протоколы, транзакции, арбитраж. Чтобы охарактеризовать конкретную шину, нужно описать:

- совокупность сигналов (сигнальных линий) и их электрические характеристики;
- физические и механические характеристики шины;
- используемые сигналы арбитража, состояния, управления и синхронизации;
- правила взаимодействия подключенных к шине устройств (протокол шины).

Различают сигналы линейные и дифференциальные.

Операции на шине называют *транзакциями.*

В рамках транзакции определены два объекта – *инициатор обмена* и *целевое устройство*. Устройство, ставшее инициатором обмена и взявшее на себя временное управление шиной, называется *ведущим* (*Bus Master*), а устройство, которое осуществляет операции по команде ведущего, – *ведомым* (*Bus Slaver*). В рамках одной физической шины в конкретный момент может происходить только одна транзакция. Для предотвращения одновременной активности нескольких ведущих в любой шине предусматривается процедура допуска к управлению шиной только одного из претендентов (*арбитраж*). В то же время некоторые шины допускают широковещательный режим записи, когда информация одного ведущего передается сразу нескольким ведомым.

Если физических шин несколько, то транзакции на них могут выполняться одновременно (*Peer Concurrency*), при условии, что пути прохождения данных не пересекаются.

Основные виды транзакций:

- конфигурационные;
- чтения;
- записи.

Транзакция включает в себя две части:

- посылку адреса;
- прием (или посылку) данных.

Транзакции бывают одиночные и пакетные.

Исполнителем может быть как инициатор, так и целевое устройство, а также другое устройство, участвующее в выполнении транзакций. К примеру, мост в транзакции чтения должен принять и переслать результаты инициатору, при этом он может:

- отложить транзакцию (записать ее в буфер). Этот вариант называется *отложенной транзакцией* (*delayed transaction*), он заставит инициатора через некоторое время повторить данную транзакцию. За это время мост должен выполнить заказанную транзакцию – принять результат чтения. Это позволяет синхронизировать медленную шину с более быстрой: транзакции на более медленную шину заносятся

в буфер, и более быстрая шина освобождается для проведения других транзакций, пока медленная шина все еще занята отложенными транзакциями;

- успешно завершить транзакцию. Такой вариант, называемый *отправленной записью* (*posted write*), возможен только для операций записи в память. Реальная запись произойдет позже, когда мост сумеет получить управление для записи.

В случае *расщепленных транзакций* (*Split*) выполняется обмен ролями: целевое устройство подает сигнал *Split Response* (расщепление), внутренне исполняет команду, а потом инициирует собственную транзакцию (команда *Split Completion*) для пересылки данных или сообщения о завершении инициатору исходной (расщепленной) транзакции. Транзакция называется расщепленной, поскольку произвольное количество других пакетов или транзакций могут использовать шину между запросом и ответом. Шину, обеспечивающую расщепление транзакций, иногда называют *шиной соединения/разъединения* (*connect/disconnect*) или *конвейерной шиной* (*pipeling bus*). Существенное увеличение пропускной способности по сравнению с шиной с коммутацией цепей стало возможным за счет разделения транзакции на две логические части: запроса шины и ответа.

Протокол – строго заданная процедура или совокупность правил, определяющих способ выполнения определенного класса функций соответствующими системами вычислительной техники.

Каждому потенциальному ведущему присваивается определенный уровень приоритета, который может оставаться неизменным (*статический*, или *фиксированный приоритет*) либо изменяться по какому-либо алгоритму (*динамический приоритет*).

Арбитраж запросов на управление шиной может быть организован по *централизованной* или *децентрализованной схемам*. Выбор конкретной схемы зависит от требований к производительности и ограничений по стоимости. При централизованном арбитраже в системе имеется специальное устройство – центральный арбитр, которое ответственно за предоставление доступа к шине только одному из запросивших доступ ведущих устройств. В зависимости от того, каким образом ведущие устройства подключены к центральному арбитру, возможны *параллельные* и *последовательные схемы* централизованного арбитража.

При децентрализованном (или *распределенном*) арбитраже единый арбитр отсутствует. Вместо этого каждый ведущий содержит блок управления доступом к шине, и при совместном использовании шины такие блоки взаимодействуют друг с другом, разделяя между собой ответственность за доступ к шине. По сравнению с централизованной схемой децентрализованный арбитраж менее чувствителен к отказам претендующих на шину устройств. В целом схемы децентрализованного арбитража потенциально более надежны, поскольку отказ контроллера шины в одном из ведущих не нарушает работу с шиной на общем уровне. Тем не менее должны быть предусмотрены средства для обнаружения неисправных контроллеров. Основной недостаток децентрализован-

ных схем заключается в относительной сложности логики арбитража, которая должна быть реализована в аппаратуре каждого ведущего.

Стек протоколов. В соответствии с эталонной моделью *OSI* (*Open System Interconnection*), разработанной в 1984 году Международной организацией по стандартизации (*International Organization of Standardization, ISO*), взаимодействие компонентов сети представляется на семи функциональных уровнях:

- прикладном;
- представительном;
- сеансовом;
- транспортном;
- сетевом;
- канальном;
- физическом.

Связь между одноименными уровнями различных компонентов сети является виртуальной (логической) на базе общих протоколов для обмена данными. В действительности взаимодействие осуществляется между смежными уровнями одного компонента с использованием интерфейсов прикладных программ *API* (*Application Programming Interface*). Информация на компьютере-отправителе должна пройти через все уровни. Затем она передается по физической среде до компьютера-получателя и опять проходит сквозь все слои, пока не достигнет того же уровня, с которого она была послана на компьютере-отправителе.

Прикладной уровень (*Application layer*) обеспечивает прикладным процессам средства доступа к области взаимодействия. Он представляется набором разнообразных протоколов, с помощью которых пользователи сети получают доступ к разделяемым ресурсам, таким как файлы, принтеры или гипертекстовые *web*-страницы, а также организуют свою совместную работу.

Функции *представительного уровня* (*Presentation layer*) – представление данных, передаваемых между прикладными процессами, в нужной форме. Этот уровень обеспечивает то, что информация, передаваемая прикладным уровнем, будет понятна прикладному уровню в другой системе.

Сеансовый уровень (*Session layer*) – обеспечение управления транзакциями для того, чтобы фиксировать, какая из сторон является активной в настоящий момент, а также предоставляет средства синхронизации. На сеансовом уровне определяется, какой будет передача между двумя прикладными процессами: полудуплексной или дуплексной.

Транспортный уровень (*Transport Layer*) предназначен для передачи пакетов через коммуникационную сеть. На транспортном уровне определяется адресация физических устройств, передаваемые данные разбиваются на блоки. Работа транспортного уровня заключается в том, чтобы обеспечить приложениям или верхним уровням модели (прикладному и сеансовому) передачу данных с той степенью срочности и надежности, которая им требуется.

Сетевой уровень (*Network Layer*) обеспечивает организацию каналов связи и выбор маршрута передачи данных.

Единицей информации *канального уровня* (*Data Link*) являются кадры (*frame*). Кадры – это логически организованная структура, в которую можно помещать данные. Задача канального уровня – обеспечить корректность передачи каждого кадра от сетевого уровня к физическому уровню. На канальном уровне обеспечивается идентификация устройства, т. е. определение типа устройства (конечное, экспандер) и адреса порта, производится согласование скоростей путем добавления в передаваемый поток специальных заполнителей, не несущих информативности. Канальный уровень также обеспечивает управление соединением.

Физический уровень (*Physical Layer*) предназначен для сопряжения с физическими средствами соединения. *Физические средства соединения* – это совокупность физической среды, аппаратных и программных средств, обеспечивающая передачу сигналов между системами. Физический уровень получает пакеты данных от вышележащего канального уровня и преобразует их в оптические или электрические сигналы физической среды передачи данных, соответствующие нулю и единице бинарного потока. Эти сигналы посылаются через среду передачи на приемный узел. Механические и электрические/оптические свойства среды передачи определяются на физическом уровне и включают:

- тип кабелей и разъемов;
- разводку контактов в разъемах.

На уровне физического интерфейса определяется логическое кодирование (8*B*10*B*), «внеполосная» сигнализация, согласование скоростей. Внеполосная сигнализация используется для обнаружения факта подключения устройства, определения его типа и согласования скорости интерфейса.

Аппаратная и программная поддержка интерфейсов. *Аппаратные средства* поддержки работы ПУ включают:

- контроллеры;
- адаптеры;
- мосты;
- повторители;
- согласующие устройства.

Контроллер – это специализированная микросхема, предназначенная для выполнения функций контроля и управления передачей данных по интерфейсу. Контроллер осуществляет управление работой ПУ и обеспечивает взаимосвязь этого устройства с компонентами ВС. Современные контроллеры являются интеллектуальными в том смысле, что, кроме непосредственного управления обменом, выполняют ряд дополнительных функций: контроль правильности данных и их корректировки, принятие решений в условиях неопределенности по результатам поступившей информации и др. То есть они содержат микропроцессор. Для поддержки такого рода контроллеров служит стандарт интеллектуального ввода-вывода *I2O* (*Intellectual Input/Output*). Его задача – определить унифицированный интерфейс между операционной системой и устройствами ввода/вывода.

Адаптер предназначен для преобразования сигналов интерфейса для подключения к определенной шине устройств и измерительной аппаратуры, использующих другой интерфейс. Адаптеры позволяют создать виртуальные порты, обмен данными с которым выполняется по интерфейсу подключаемых приборов, причем пользователю не требуется никаких знаний об устройстве и работе основного интерфейса.

Мостом называется устройство для управления шинами – он соединяет две отдельные сети или два сегмента одной сети, использующих одинаковые протоколы. Центральное место для обеспечения баланса производительности ВС по мере роста быстродействия микропроцессоров занимают специализированные наборы мостов и контроллеров – чипсеты. Как правило, чипсет состоит из двух (трех) микросхем, одна из которых, находящаяся на верхнем уровне иерархии шин интерфейсов, называется «северным мостом» (системным контроллером), другая – «южным мостом» (контроллером ввода-вывода). Северный мост управляет передачей данных между процессором, оперативной памятью, графическим контроллером и южным мостом. Южный мост организует связь с контроллерами устройств ввода-вывода и предназначен для работы с низкоскоростными интерфейсами.

Повторитель восстанавливает уровень и форму сигнала, которые изменяются, проходя по линии передачи. Они используют для увеличения расстояния, на которое требуется передать сигнал, а также для увеличения нагрузочной способности (коэффициента разветвления) передатчика интерфейса.

Согласующие устройства преобразуют сигналы физических соединительных линий в сигналы интерфейса, что дает возможность непосредственного подключения и эксплуатации различных датчиков и исполнительных механизмов. К согласующим блокам интерфейса относят также активные и пассивные терминаторы – поглотители энергии (обычно резисторы) на конце проводных длинных линии, сопротивление которых равно их волновому сопротивлению.

Программные средства доступа к ПУ строятся на основе следующих принципов:

- разбиение транзакций на несколько уровней, причем нижние учитывают особенности аппаратуры, а верхние обеспечивают удобный интерфейс для пользователей;
- независимость от устройств;
- единые правила именования ПУ;
- обработка ошибок транзакций как можно ближе к аппаратуре. Если контроллер обнаруживает ошибку чтения, то он должен попытаться ее скорректировать. Если же это ему не удается, то исправлением ошибок должен заняться драйвер устройства. И только если нижний уровень не может справиться с ошибкой, он сообщает об ошибке верхнему уровню;
- использование различных способов обмена. К примеру, операционная система (ОС) выполняет операции ввода/вывода асинхронно, но представляет их для пользовательских программ в синхронной форме;

- обеспечение работы как с разделяемыми устройствами, к которым возможен доступ нескольких процессов, так с выделенными.

Поэтому программное обеспечение ввода-вывода делится на четыре слоя:

- независимый от устройств слой операционной системы;
- обработка прерываний;
- драйверы устройств;
- пользовательский слой программного обеспечения.

Программирование доступа к ПУ в общем случае является нетривиальной задачей, даже если не касаться особенностей работы с ПУ, связанных с архитектурой ОС (которая в общем случае реализует виртуализацию ПУ через систему драйверов). Единого интерфейса программирования (*API*) для работы с ПУ не существует, зачастую даже стандартный интерфейс для определенного типа устройств разрабатывается не сразу.

Ранее разработчики ПО полагались на *API*, предоставляемый системным *BIOS* (*Basic Input/Output System,* базовая система ввода-вывода − это часть программного обеспечения ПЭВМ, поддерживающая управление адаптерами внешних устройств, экранные операции, тестирование, начальную загрузку и установку ОС) или *BIOS* самого устройства, а в сложных случаях прибегали к «ручному» программированию устройства. Однако в многозадачных средах такой подход не работает – требуется обеспечить множественный доступ к одному и тому же устройству. Реализуется это либо программно, через драйверы, либо через интеллектуальный хост-контроллер, функции которого распределены между «железом» и драйверами.

Различают четыре основных метода доступа: *последовательный, прямой, произвольный и ассоциативный.* С каждым из них связана своя организация памяти. В ЭВМ используются *три основных способа организации передачи данных между памятью и периферийными устройствами*:

- программно-управляемая передача, инициируемая процессором;
- программно-управляющая передача, инициируемая запросом прерывания от периферийного устройства;
- прямой доступ к памяти (ПДП, *DMA − Direct Memoryn Access*).

Модуляция и кодирование данных. В настоящее время многие источники информации являются цифровыми. Исходные данные путем логического кодирования преобразуются в битовые последовательности, а потом передаются методами физического кодирования, либо используются различные способы их модуляции для хранения по тому или иному физическому принципу.

Электрический принцип кодирования. При цифровом кодировании для представления логических нуля и единицы используют *значение потенциала сигнала* (*потенциальный код) или импульсы определенной полярности (биполярный импульсный код)* либо *перепады потенциала* (*импульсный код манчестерский)*.

Потенциальное кодирование или код без возвращения к нулю (*Non Return to Zero, NRZ*) представляет собой последовательность однополярных импульсов со скважностью $q = 1$ (при передаче последовательности единиц, сигнал не воз-

вращается к нулю в течение такта. То есть смена сигнала происходит при передаче единицы, а передача нуля не приводит к изменению напряжения). Когда используется потенциальный код с инверсией при единице (*NRZI*) или дифференциальный *NRZ*, то состояние меняется в начале битового интервала для единицы и не меняется при нуле (передается потенциал, который был установлен на предыдущем такте). Он удобен в тех случаях, когда наличие третьего уровня сигнала весьма нежелательно, например, в оптических кабелях, где устройство распознает только два сигнала – свет и темноту.

Принцип дифференциальной передачи данных заключается в передаче одного сигнала по двум проводам. Причем по одному проводу идет оригинальный сигнал, а по другому – его инверсная копия. Таким образом, между двумя проводами пары всегда есть разность потенциалов, именно этой разностью потенциалов и передается сигнал. Такой способ передачи обеспечивает высокую устойчивость к синфазной помехе (минимизирует электромагнитное излучение и позволяет упростить чтение переданных сигналов).

Базовыми методами модуляции являются:

- частотная модуляция (*Frequency Modulation, FM* или *Frequency-Shift Keying, FSK*);
- амплитудная модуляция (*Amplitude-Shift Keying – ASK*);
- фазовая модуляция (*Phase-Shift Keying, PSK*).

Широко используются различные их модификации и комбинации, ориентированные на повышение плотности и помехоустойчивости с учетом области применения:

- модифицированная частотная модуляция;
- квадратурная амплитудная модуляция;
- модуляция с ограниченной длиной последовательности (*Run Length Limited, RLL*);
- метод на основе цифровой обработки сигналов (*Partial-Response, Maximum-Likelihood, PRML – групповой отклик, максимальная достоверность* или *частичное определение, максимальное правдоподобие).*

Широтно-импульсная модуляция, или ШИМ (*Pulse Width Modulation, PWM*) – это операция получения изменяющегося аналогового значения посредством цифровых устройств. С помощью задания скважности можно менять среднее напряжение на выходе ШИМ. Другими словами, получаем аналоговый сигнал цифровыми методами.

Ниже рассматриваются способы формирования сигналов в различных физических средах хранения и передачи данных.

Формирование аудиосигналов. Звук представляет собой колебания физической среды (обычно воздуха) частотой приблизительно 20 – 20 000 Гц. Обработка и передача звука основаны на преобразовании этих колебаний в электрический сигнал, последующей его (аналоговой или цифровой) обработки и выводе вновь в виде колебаний физической среды.

Запись произвольного звука осуществляется путем оцифровки аналогового сигнала, представляющего собой электрическую копию звукового давления (датчик

звукового давления – микрофон). Преобразование аналогового сигнала в цифровую форму выполняет аналого-цифровой преобразователь (АЦП), служащий для дискретизации сигнала по времени (частота оцифровки) и квантования по уровню (собственно цифровое представление сигнала) – *импульсно-кодовая модуляция* (*Pulse Code Modul*ation, *PCM*).

Для сокращения потока данных используются иные (отличные от *PCM*) методы кодирования аналогового сигнала. При так называемом μ -*кодировании* аналоговый сигнал преобразуется в цифровой код, определяемый логарифмом величины сигнала (а не его линейным преобразованием). Недостаток метода – необходимость иметь априорную информацию о характеристиках исходного сигнала. При *дифференциальной импульсно-кодовой модуляции* (*Differential Pulse Code Modulation*, *DPCM*) сохраняется только разность между текущим и предшествующим уровнями сигнала (разница требует для цифрового представления меньшего количества бит, чем полная величина амплитуды). При *дельта-модуляции* (*Delta Modulation, DM*) каждая выборка состоит всего из одного бита, определяющего знак изменения исходного сигнала (увеличение или уменьшение); дельта-модуляция требует повышенной частоты сэмплинга.

Наибольшее распространение при записи звука получила *адаптивная импульсно-кодовая модуляция* (*Adaptive Pulse Code Modulation, ADPCM*), использующая восьми- или четырехразрядное кодирование для разности сигналов.

Существуют два пути решения синхронизации устройств чтения/записи:

- специальный сигнал синхронизации;
- синхросигнал объединен с сигналом данных.

Из возможных передаваемых символов выбираются такие, которые за счет достаточного количества фронтов без задающего генератора обеспечивают синхронизацию отправителя и получателя по границам битов. Эта проблема решается представлением байта словами большей разрядности (например, кодирование 8/10 или 8/14). Очевидно, в силу применения такой кодировки суммарная пропускная способность канала уменьшается). Основной особенностью схемы кодирования 8/10 является то, что количество последовательно передаваемых нулей (или единиц) не должно превышать четырех.

Передача нулей и единиц осуществляется в виде изменения величины подаваемого напряжения. Поэтому промежуток между переходными напряжениями, которые подаются передатчиком, получается достаточно сбалансированным, с более устойчивым и регулярным потоком импульсов. Нагрузка схемы становится более постоянной, что приводит к повышению ее надежности. Это свойство актуально для лазерных оптических передатчиков. От данного соотношения зависит их нагрев и при колебании степени нагрева увеличивается количество ошибок приема. Применяется в гигабитной сети на оптоволокне.

Формирование звука при выводе осуществляется *цифроаналоговым преобразователем* (ЦАП) либо *синтезатором*, которые есть почти на всех звуковых картах. Наиболее часто применяют цифровой *FM*-синтез звука путем использования генераторов огибающей, управляющих амплитудой отдельных *VOC*-генераторов (*Voltage-Controlled Oscillator*). В цифровом *FM*-синтезе каж-

дый из таких генераторов называется *оператором*, в нем выявляются два базовых элемента:

- *фазовый модулятор* – задает частоту (высоту) звука;
- *генератор огибающей* – задает амплитуду (громкость) звука.

Кроме *FM*-синтеза в высококачественных звуковых картах используется табличный, или *WT*-синтез (*Wave Table*). Такие устройства именуются также синтезаторами выборок. Идея применения *WT*-синтеза состоит в использовании специальных алгоритмов, позволяющих по одному лишь характерному тону (выборке) музыкального инструмента воспроизвести все остальные тона (фактически восстановить его полное звучание). Известен *WF*-метод (*Wave Form*) генерации звучания, основанный на преобразовании звуков в сложные математические формулы и дальнейшем применении этих формул для управления мощным процессором с целью воспроизведения звука; от *WF*-синтеза ожидают еще большей (относительно *FM*- и *WT*-технологий) реальности звучания музыкальных инструментов при ограниченных объемах звуковых файлов.

Для работы со звуком ПЭВМ содержит следующие *обязательные аналоговые порты*:

- линейный стереовход, который предназначен для подключения источника аналогового сигнала (стерео- или моно-) с линейного выхода внешних аналоговых устройств, например, аудиоплеера, радиоприемника, видеомагнитофона и пр.;
- стереовход *CD Audio* для подключения *CD*-привода;
- вход для подключения микрофона;
- дополнительный линейный вход (AUX-In) на плате, на который передается аналоговый звук с карт *FM*- или *TV*-тюнера, или других ПУ, например, второго *CD*/*DVD*-привода;
- выход на головные телефоны;
- линейные выходы для вывода звука на активные акустические колонки, усилитель или линейный вход любого внешнего устройства (например, магнитофона).

Для подключения к ПЭВМ звукового оборудования, работающего в составе различных аппрататно-программных комплексов (например, охранной сигнализации, ультразвукового контроля скорости, систем звукового мониторинга и др.) используется интерфейс *UART – Universal Asynchronous Receiver*/*Transmitter*, универсальный асинхронный приемопередатчик, *USB*, *FireWire* и беспроводные интерфейсы.

Звук может быть получен при помощи внешнего синтезатора, управляемого от компьютера, при использовании:

- при использовании *MIDI*-порта, который содержится в *Super I*/*O* и практически на всех звуковых картах, обладающих многими дополнительными возможностями обработки звука (распознавание речи, реверберация, спецэффекты подобные трехмерному звучанию и др.). *MIDI* (*Musical Instrument Digital Interface*) – спецификация цифрового интерфейса, включающая стандарт на программную и аппаратную части для организации локальной сети электронных инструментов;

- специальных карт-адаптеров, оснащенных *цифровыми сигнальными процессорами* (*Digital Signal Processor*, *DSP*).

Формирование сигнала на поверхности магнитного диска. Для хранения данных на жестких дисках использован хорошо известный принцип упорядочивания направления намагничивания частиц ферромагнетиков под действием внешнего магнитного поля. В качестве среды записи и хранения информации в жестких дисках выступают ферромагнетики, отличительной особенностью которых является наличие микроскопических однородно намагниченных объемов вещества, называемых доменами. Один бит магнитной информации – это один магнитный домен ферромагнитного материала, направление вектора намагниченности в котором может быть изменено внешним полем.

При записи в ячейках формируется последовательность зон смены знака, зависящая от способа кодирования информации. При кодировании один бит или группа битов заменяется несколькими колебаниями напряженности магнитного поля. Таким образом решается проблема синхронизации и отделения битов друг от друга. Задача элемента чтения – обнаружить изменения направления намагниченности участков диска.

Участок дорожки записи, на котором может быть записана одна зона смены знака, называется ячейкой переходов (*transition cell*) или просто битовой ячейкой. Геометрические размеры такой ячейки зависят от тактовой частоты сигнала записи и скорости, с которой перемещаются относительно друг друга головка и поверхность диска.

Чтобы решить проблему синхронизации и отделения битов друг от друга, применяется специальное кодирование информации при записи на магнитный носитель. При кодировании один бит или группа битов заменяется несколькими колебаниями напряженности магнитного поля.

При *FM*-кодировании ячейка должна начинаться с зоны смены знака, которая выполняет роль заголовка (синхросигнала). Затем в зависимости от значения бита данных следует (или не следует) переход: 1 = *TT*, 0 = *TN*, где *T* и *N* – присутствие и отсутствие изменения направления напряженности магнитного поля соответственно.

При *MFM*-кодировании изменение магнитного поля происходит только в случае, когда несколько нулей идут подряд: 1 = *NT*, (0)0 = *TN*, (1)0 = *NN*.

RLL-кодирование обеспечивает такую закодированную последовательность, что длина поля записи (количество бит между переходами от нуля к единице или от единицы к нулю) ограничена определенным диапазоном [$d + 1$; $k + 1$]. Кодирование происходит так, чтобы расстояние между зонами смены знаков было не слишком маленьким, но и не слишком большим (с учетом разрешения элемента чтения/записи и синхронизации). Параметры d и k задаются модификацией алгоритма, который обозначается *RLL d, k*. Изменяя эти параметры, можно получать различные методы кодирования.

Пример – *RLL* 2,7 – 8 бит данных перекодируются в 16 так, чтобы в последовательности встречалось не менее двух и не более семи нулей. Таблица

перекодировки различных последовательностей битов в серии зон смены знака (*IBM*) выглядит следующим образом:

10 *NTNN*;
11 *TNNN*;
000 *NNNTNN*;
010 *TNNTNN*;
011 *NNTNNN*;
0010 *NNTNNTNN*;
0011 *NNNNTNNN*.

При *PRML*-кодировании контроллер анализирует поток данных с головки посредством фильтрации, обработки и алгоритма определения (элемент частичного определения), а затем предсказывает последовательность битов, которые этот поток данных наилучшим образом представляет (элемент максимального правдоподобия). Контроллер *PRML* применяет большую тактовую частоту дискретизации при переводе аналогового сигнала в цифровой. Он рассматривает не один всплеск, а целый временной интервал, описывающий считанный сигнал. Далее контроллер сравнивает полученные результаты и подбирает наиболее похожий набор данных. Для этого используются следующие *основные допущения*:

- форма считанного сигнала отдельного перехода полярности известна;
- суперпозиция сигналов соседних переходов линейная.

Формирование сигнала на поверхности магнитной ленты. Магнитная лента представляет собой гибкую ленту из того или иного материала, на которую с одной стороны нанесено покрытие из ферромагнетика. Применение вакуумного напыления позволяет надежно фиксировать покрытие (без адгезивного материала), что повышает устойчивость ленты к износу. Запись данных на ленту выполняется в виде дорожек, разделенных зазорами.

Практически все современные устройства записи на ленту поддерживают аппаратное сжатие данных в «прозрачном» режиме. В ходе эволюции ленточных устройств разработчики перепробовали большое количество различных алгоритмов сжатия без потерь, лучшие из них обеспечивают среднестатистическое сжатие не менее 3:1.

Последовательная схема сжатия LZ (Зива – Лемпеля, 1977). Используется обратимое преобразование блока данных к виду, который позволяет эффективно сжать данные с помощью простых алгоритмов. Преобразование имеет целью сгруппировать символы так, чтобы вероятность появления последовательностей идентичных символов значительно возросла. Такой текст может быть легко сжат посредством локально-адаптивных алгоритмов в сочетании с кодировкой Хафмана и арифметической кодировкой.

Арифметическое кодирование IDRC (*Improved Data Recording Capability*). Идея сжатия заключается в представлении входного потока информации число, находящееся в полуинтервале [0, 1). Для нахождения такого числа необходима информация об относительной частоте P_i каждого вхождения i-го символа в этот поток.

Большинство ленточных приводов благодаря большой площади головки выполняет чтение одновременно с записью, чтобы гарантировать отсутствие изначально нечитаемых участков. Если отмечено наличие ошибки записи, привод обычно повторяет дефектный участок еще раз. Формат, размер блоков, количество контрольных кодов зависят от конкретной реализации.

В современных устройствах данные защищаются кодами Рида – Соломона, причем многоуровневыми. Повышение надежности записи обеспечивается благодаря использованию пакетной, а не дорожечной записи. Каждый пакет защищен четырехуровневым кодом Рида – Соломона, что обеспечивает исключительную устойчивость к дефектам ленты. Поточный способ записи при постоянной скорости движения ленты обеспечивается упреждающим кэшированием и адаптивным механизмом выбора скорости движения ленты.

Для повышения емкости постоянно увеличивалось количество дорожек, от модуляции *MFM* разработчики перешли к *RLL* и *PRLM* и использованию адаптивных алгоритмов сжатия.

В настоящее время *ленточные библиотеки* обеспечивают высокую надежность, масштабируемость, управляемость и защиту данных в гетерогенных сетях хранения данных (*Storage Area Network, SAN*). Встроенное аппаратное шифрование *AES* 256 обеспечивает высокий уровень защиты данных со сжатием и соответствие нормативным требованиям при выполнении резервного копирования конфиденциальной информации

Виртуальная ленточная библиотека (*Virtual Tape Library, VTL*) – выделенное вычислительное оборудование, которое эмулирует накопители физической ленточной библиотеки. *VTL* обычно состоят из трех компонентов:

- компьютерного оборудования;
- программного обеспечения;
- *RAID* массива жестких дисков.

Виртуальные ленточные библиотеки дополняют решения для хранения данных на физических лентах и обеспечивают два уровня хранения: первичная резервная копия, которая размещается на жестком диске, вторичная – на магнитной ленте. Внедрение *VTL* позволяет сократить время операций копирования и восстановления, не требуя изменения имеющихся процессов или рабочих нагрузок.

Формирование сигнала на поверхности оптического диска. В основе технологии лежит принцип регистрации интенсивности лазерного луча, отраженного от рельефной (или неоднородной с точки зрения оптики) поверхности.

В *магнитооптических дисках* используются лазерная и электромагнитная головки.

В процессе записи локальные участки диска нагреваются лазером высокой мощности до точки Кюри (обычно 200–300 °С), когда они становятся восприимчивыми к воздействию внешнего магнитного поля.

В процессе чтения дорожки облучаются лазером низкой мощности, прошедшим через поляризатор. Согласно явлению Керра намагниченные участки поверхности способны поворачивать плоскость поляризации луча в ту или

иную сторону. Факт поворота фиксируется фотоприемником, расположенным на оптической головке.

Особенность формирования сигнала на поверхности *оптического диска* учтена при выборе методики модуляции и кодирования информации.

Модулирование степени отражения лазерного луча выполняется тремя способами:

- варьированием рельефа отражающего слоя (штампованные диски);
- варьированием прозрачности органического слоя, покрывающего слой отражающего материала (записываемые/перезаписываемые диски);
- созданием эффекта голограммы за счет интерференции нескольких лучей, сфокусированных на разной глубине (голографические диски).

На сегодняшний день существуют *три поколения оптических дисков для хранения данных*:

- *CD – Compact Disc*;
- *DVD – Digital Versatile Disc*;
- *BD – Blu-ray Disc*.

На *CD*-диске данные хранятся в виде выборок (сэмплов) цифрового стереозвука 16 бит, полученные с частотой 44,1 кГц. Хранение данных общего назначения выполняется в том же формате, только вместо выборок используются слова (16 бит) данных. Применяется кодирование 8/14 (*Eight to Fourteen Modulation, EFM*), перемеживание (чередование, скремблирование) и двухуровневый подсчет контрольных сумм по схеме Рида – Соломона – так исправляются ошибки бит на физическом уровне и в данных фрейма (кадра).

Физическая структура *DVD*-диска общая для всех типов информации, хранимой на нем. Данные о типе и содержании мультимедиа-контента хранится в файловой системе и внутренней структуре файлов, а не на физическом уровне. Единица хранения данных – сектор с 2048 байтами полезной нагрузки.

Для нейтрализации возможных дефектов используется двухуровневая схема избыточности с добавлением контрольных сумм: используются *ECC*-коды для идентификатора и для данных, а также применен иной способ кодирования *EFM+* – 8/16 при *RLL* 2.10.

Для обеспечения корректной синхронизации и подстройки частоты требуется добавление битовых последовательностей специального вида, не встречающихся в данных. Возникновение повторяющихся последовательностей битов мешает системе слежения за дорожкой. Их нужно исключить путем перемешивания – скрэмблирования. Тем самым нейтрализуются протяженные дефекты.

Технология *BD* вводит новый метод модуляции – 1.7 *PP* (*Parity Preserving – Prohibit RMTR – repeated minimum transition run length*) с минимальной длиной пита в 2*T* (так называемый *d* = 1 код). Данный метод кодировки предусматривает:

- преобразование 8/14 по методу *RLL* 1.7;
- кодирование *NRZI*;
- сохранение четности количества переходов (для достижение статистически нулевого постоянного тока) путем вставки в код единичных бит);

- исключение повторяющихся последовательностей минимальной длины (0101010…).

Для защиты данных используется перекрестный *ECC*-код для блока в 64 Кбайт (вдвое больше, чем у *DVD*), что обеспечивает восстановление при ошибках из-за дефектов большой протяженности.

Применение лазера с длиной волны 405 нм – логичный шаг в сторону повышения плотности, поскольку минимально достижимый размер пятна лазера прямопропорционален длине волны излучения. При этом 400 нм является теоретически достижимым пределом, так как при дальнейшем уменьшении длины волны проявляются квантовые артефакты потери прозрачности некоторых оптических сред, в том числе поликарбоната, из которого изготавливают оптические диски.

Формирование инфракрасных сигналов. Для формирования используется источник света с длиной волны 850–900 нм. *Устройство инфракрасного интерфейса* (*Infra-red Data Association, IrDA*) подразделяется на два основных блока:

- преобразователь (модули приемника-детектора и диода с управляющей электроникой);
- кодер-декодер.

Блоки обмениваются данными побайтно по электрическому интерфейсу, в котором в том же виде транслируются через оптическое соединение, за исключением того, что здесь они пакуются в кадры простого формата – 8 бит данных дополняются одним старт-битом в начале и одним стоп-битом в конце данных. Обмен данными происходит асинхронно, для обнаружения ошибок применяется *CRC*.

Сам порт *IrDA* основан на архитектуре коммуникационного *COM*-порта ПЭВМ, который использует универсальный асинхронный приемопередатчик (*UART*).

Связь в *IrDA* полудуплексная, так как передаваемый инфракрасный луч неизбежно засвечивает фотодатчик приемника. Стандарт *IrDA* требует, чтобы все последовательные биты кодировались таким образом: логический нуль передается одиночным импульсом длиной от 1,6 мс до 3/16 периода передачи битовой ячейки, а логическая единица передается как отсутствие импульса.

Формирование сигнала в радиопротоколах. При формировании радиосигналов используется аналоговая модуляция, при которой информация кодируется изменением амплитуды, частоты или фазы синусоидального сигнала.

Одна из основных проблем построения беспроводных систем – это решение задачи доступа многих пользователей к ограниченному ресурсу среды передачи. Существует несколько базовых методов доступа (их еще называют методами уплотнения или мультиплексирования), основанных на разделении между станциями таких параметров, как пространство, время, частота и код. Задача уплотнения – выделить каждому каналу связи пространство, время, частоту и/или код с минимумом взаимных помех и максимальным использованием характеристик передающей среды.

Пространственное уплотнение основано на разделении сигналов в пространстве, когда передатчик посылает сигнал, используя код *c*, время *t* и частоту *f* области s_i. То есть каждое беспроводное устройство может вести передачу данных только в границах определенной территории, на которой любому другому устройству запрещено передавать свои сообщения. Характерный пример – системы сотовой телефонной связи.

При *уплотнении с частотным разделением* (*Frequency Division Multiplexing, FDM*) каждое устройство работает на определенной частоте, благодаря чему несколько устройств могут вести передачу данных на одной территории. Это один из наиболее известных методов, так или иначе используемый в самых современных системах беспроводной связи.

Уплотнение с временным разделением (*Time Division Multiplexing, TDM*) основано на распределении каналов по времени, т. е. каждый передатчик транслирует сигнал на одной и той же частоте области *s*, но в различные промежутки времени (как правило, циклично повторяющиеся) при строгих требованиях к синхронизации процесса передачи.

Формирование изображений. Изображение, как и звук, и радио, имеет электроволновую природу, т. е. представляет собой совокупность значений электромагнитного излучения. Для описывается цвета разработано много моделей.

Способы задания цвета:

- колориметрический – цвет описывается как точка в некоторой системе координат (в цветовом пространстве);
- системой спецификаций – когда каждой точке задается определенный цвет.

В зависимости от способа регистрации различают следующие модели цвета:

- интуитивные (пространство *HSL*, где *Hue* – цветовой тон, *Saturation* – насыщенность и *Lightness* – яркость);
- аддитивные (цветовое пространство *RGB*);
- субтрактивные (пространство *CMYK*, где *Cyan* – голубой, *Magenta* – розовый, *Yellow* – желтый и *Key color* – черный. Модели такого типа применяются, когда имеют дело с цветом, возникающим за счет поглощения части спектра, т. е. вычитания цветовых потоков).

Создание графического объекта состоит из трех этапов:

1 *Ввод изображения*. Используются различные устройства ввода – сканеры, дигитайзеры, фото- и видеокамеры и др., после чего создается трехмерная модель сцены и объектов на ней.

2 *Рендеринг*. Выполняется построение двухмерных проекций компьютерной модели в соответствии с выбранной физической моделью для визуализации.

3 *Вывод изображения*. Основные устройства вывода – это дисплей, принтер, проектор.

В *электронно-лучевых дисплеях* формирование изображений основывается на использовании внешнего фотоэффекта (фотоэлектронной эмиссии) – освобожденные электроны покидают облученное вещество, вылетая в пространство, и заставляют светиться экран дисплея.

Испускаемый пушками пучок электронов модулируется по интенсивности, фокусируется, разгоняется и направляется с помощью отклоняющей системы в заданную точку поверхности стеклянной колбы *электронно-лучевой трубки* (ЭЛТ, *Cathode Ray Tube, CRT*). Внутренняя поверхность колбы покрыта люминофором – материалом, способным излучать свет (кратковременно) при попадании электронов.

В *жидкокристаллических* (*ЖК*) *дисплеях* (*Liquid Crystal Display, LCD*) используется особое вещество, которое обладает кристаллической структурой (а значит, анизотропностью основных физических свойств), но при этом при комнатной температуре сохраняет жидкое состояние. Анизотропность свойств требуется для того, чтобы вещество было способно преобразовывать свойства светового излучения, т. е. работать как фильтр. Поместив вещество в отдельные ячейки, можно получить управляемые фильтры для пикселей. При этом для применения в ЖК-устройствах отобраны вещества, реагирующие на электрическое напряжение. Жидкое состояние необходимо для подвижности кристаллов. Под действием напряжения кристаллы меняют свою конфигурацию, сдвигаясь относительно друг друга. При этом меняется направление преобразования света – так мы получаем управляемый светофильтр.

В плазменных дисплеях (*панелях*) (*Plasma Display Panel, PDP)* используется явление свечения люминофора под воздействием ультрафиолетовых лучей, возникающих при электрическом разряде в ионизированном газе (плазме) плазменная панель представляет собой матрицу газонаполненных ячеек, заключенных между двумя параллельными стеклянными поверхностями. В качестве газовой среды обычно используется инертный газ неон или ксенон. Разряд в газе протекает между прозрачным электродом на лицевой стороне экрана и адресными электродами, проходящими по его задней стороне. Газовый разряд вызывает ультрафиолетовое излучение, которое, в свою очередь, инициирует видимое свечение люминофора. В цветных плазменных панелях каждый пиксель экрана состоит из трех идентичных микроскопических полостей, содержащих инертный газ и имеющих два электрода, спереди и сзади. Таким образом, каждый пиксель создан из трех ячеек, представляющих собой крошечные флуоресцентные лампы.

Проекторы отличаются типом устройства, формирующего первичное изображение, которое впоследствии с помощью лампы и оптической системы выводится в объектив.

Существуют четыре базовые технологии:

- технология электронно-лучевых трубок (*CRT*);
- жидкокристаллическая просветная технология (*LCD*);
- жидкокристаллическая отражающая технология (*Liquid Crystal on Silicon, LCOS* – жидкие кристаллы на кремнии, или *Direct Drive Image Light Amplifier, D-ILA* – зеркальная матрица на основе ЖК-технологии. Изображение по этой технологии формируется жидкими кристаллами, однако работают они не на просвет, а на отражение;

- микрозеркальная технология (*Digital Light Processing, DLP* – матрица управляемых микрозеркал).

Наиболее совершенной технологией считается *LCoS*, а наиболее распространенной – *DLP*.

Технология *DLP* обладает следующими характеристиками:

- хорошая контрастность, цветопередача и равномерность;
- высокое общее качество картинки;
- отсутствие «битых» пикселей;
- отсутствие эффекта старения.

Применение субпиксельных светофильтров в проекционных устройствах невозможно, поэтому для получения цветного изображения применяют *два основных подхода*:

1 *Использование трех раздельных матриц*, снабженных общими светофильтрами (обычно красным, зеленым и синим). Изображения от трех матриц (по трем цветовым каналам) смешиваются в оптической системе для получения полноцветной картинки. Данный подход реализуется в проекторах на основе *LCD* (3*LCD*).

2 *Разделение по цветовым каналам по времени*. Выходной световой поток проходит через сменяющиеся светофильтры, оформленные в виде колеса с четырьмя – шестью сегментами. За счет синхронизации с колесом матрица выдает несколько раздельных монохромных кадров, которые смешиваются в восприятии зрителя в один кадр. Этот подход реализуется в *DLP*-технологии, поскольку она и так основана на эффекте инерции человеческого зрения.

Печать. Краска на бумаге сама не способна испускать свет. Каждый красящий пигмент поглощает световой поток лишь некоторой части спектра и отражает некоторую часть попавшего на него света. Красная краска поглощает весь свет, кроме красной области, синяя – все цвета, кроме синего, и т. д.

Если белый цвет на мониторе является смесью всех основных цветов, то на бумаге белый – это отсутствие краски, а черный теоретически должен был бы формироваться из смеси всех красок, что в реальности не получается. Поэтому в используемой модели *CMYK* и введен отдельно черный цвет (компонент *K* – *key color*).

В современной полиграфии используются краски, определяемые взаимозависимыми цветами. Например, если взять фиолетовую краску, то она будет «убивать» зеленый цвет (расположенный напротив на цветовом круге), т. е. чем больше фиолетовой краски, тем меньше в картинке зеленого, при этом на другие цвета она почти не влияет. Аналогично синим цветом при субтрактивном описании управляет желтый пигмент, а красным – голубой.

Для получения качественных фотоснимков этих четырех цветов недостаточно, поэтому в струйных принтерах к четырем основным цветам добавлено еще несколько ярких оттенков цветов, например, для шестицветных принтеров применяют палитру *CMYKLcLm*, где *Lc* – светло-голубой, *Lm* – светло-розовый).

В некоторых принтерах имеется возможность контроля интенсивности светового потока и цифровой коррекции совмещения цветов и устранения дефектов из-за возможных деформаций линейки – наклона и изгиба.

Эмуляция, или язык описания страниц. Эмуляция – имитация работы одной системы средствами другой, без потери функциональных возможностей и искажений результатов, другими словами, это программный продукт, позволяющий понимать печатному устройству компьютер.

Язык описания страниц является специфическим языком программирования для управления принтером или другим устройством для создания изображений.

Язык *PostScript* (создан *Adobe* для лазерных принтеров) служит для описания векторной и растровой графики в стиле объектно-ориентированных языков программирования. Поддерживаются графические примитивы, масштабируемые шрифты, кривые Безье и другие элементы векторной графики. На принтер отправляется не изображение, а геометрические объекты. Для того чтобы напечатать текст определенным шрифтом, драйвер принтера должен указать последнему контур шрифта и его размер. Контур шрифта служит шаблоном для создания символов любого размера. Принтер генерирует изображение символа из его контура, а не загружает из памяти. Этот тип изображения, который генерируется индивидуально для каждой страницы, называется векторной графикой, в отличие от растровой графики, которая отправляется на принтер в виде готового набора точек.

Язык *PCL* (*Printer Command Language*, разработан *HP*) более объектно-ориентированный, имеет широкие средства управления шрифтами и объектами, близок к *API* программирования интерфейса *Windows* (*GUI*).

GDI – Graphic Device Interface – это библиотека определенных функций *ОС Windows* для осуществления вывода информации на графические периферийные устройства, такие как дисплеи или принтеры (*GDI*-принтер работает только с операционной системой *Windows* и использует ресурсы компьютера для обработки данных). В отличие от принтеров с мощным встроенным процессором, контроллер *GDI*-принтера всего лишь выводит информацию в буферную память принтера. Принимаемая программой печати информация представляет собой описание страницы, воспроизводящее уже подготовленные к печати графические примитивы – линии, текст и пр., для обработки которых и вызываются функции *GDI*.

3*D-принтер* – периферийное устройство, использующее метод послойного создания физического объекта по цифровой 3*D*-модели, а процесс трехмерной печати называется быстрым *прототипированием*, т. е. изготовлением прототипов моделей и объектов для их дальнейшей доводки. В инженерии такой подход способен существенно снизить затраты в производстве и освоении новой продукции.

В 3*D-принтерах* используются лазерная и струйная технологии. При лазерной технологии ультрафиолетовый лазер послойно засвечивает жидкий фотополимер, который затвердевает и превращается в достаточно прочный пластик, либо ла-

зер сплавляет порошок из металла или пластика, либо вырезает в каждом слое сечение создаваемого объекта. При струйной технологии используется застывание разогретого термопластика (или полимеризация фотополимерного пластика под действием ультрафиолетовой лампы) либо склеивание (или спекание) порошкообразного материала. Уже на этапе проектирования можно кардинальным образом изменить конструкцию узла или объекта в целом.

Сканеры используют оптический принцип – преобразование световой энергии в электрические сигналы (регистрация оптической прозрачности или отражающей способности элементов оригинала и преобразование их в электрические импульсы). Как правило, сканеры считывают данные оригинала поэлементно (обычно – построчно) и используют искусственное освещение. Вебкамеры и цифровые фото- и видеокамеры тоже можно отнести к сканерам, выполняющим мгновенное считывание всего оригинала

В сканерах используются три основных типа светочувствительных датчиков:

- ПЗС – прибор с зарядовой связью (*Charge-Coupled Device, CCD*);
- КМОП-сенсор (*Contact Image Sensor, CIS*);
- ФЭУ – фотоэлектронный умножитель (*Photomultiplier tube, PMT*).

ПЗС – специализированная аналоговая интегральная микросхема, состоящая из светочувствительных фотодиодов, выполненная на основе кремния. При подаче напряжения через поликремниевые затворы изменяются электрические потенциалы вблизи электродов.

До экспонирования обычно происходит сброс всех ранее образовавшихся зарядов и приведение всех элементов в идентичное состояние. Далее комбинация напряжений на электродах создает потенциальную яму, в которой могут накапливаться электроны, образовавшиеся в данном пикселе матрицы в результате воздействия света при экспонировании (явление внутреннего фотоэффекта). Чем интенсивнее световой поток во время экспозиции, тем больше накапливается электронов в потенциальной яме, соответственно, тем выше итоговый заряд данного пикселя.

Светочувствительные элементы КМОП также работают на основе внутреннего фотоэффекта – перераспределении под действием излучения электронов по состояниям в полупроводниках и диэлектриках. Их применение позволяет достичь чрезвычайно низкого уровня энергопотребления. Кроме того, в отличие от оснащенных люминесцентными лампами ПЗС-сканеров, модели на базе КМОП не требуют времени для прогрева и обеспечивают мгновенную готовность к работе независимо от состояния устройства.

ФЭУ – это электронная вакуумная лампа, способная усиливать зарегистрированный фотонный импульс. Принцип работы основан на вторичной электронной эмиссии, вызываемой падающим на кадод ФЭУ светом. Измерение полученного на аноде ФЭУ напряжения позволяет преобразовать цвет в цифровые данные.

В цветных сканерах применяется по три фотодатчика, по одному на каждый цвет.

3D-сканер – периферийное устройство, анализирующее физический объект и на основе полученных данных создающее его 3*D*-модель. 3*D*-сканеры делятся на два типа по методу сканирования:

- контактный (работа которого основывается на непосредственном контакте сканера с исследуемым объектом);
- бесконтактный.

Полученные методом сканирования 3*D*-модели в дальнейшем могут быть обработаны средствами САПР и использоваться для разработки технологии изготовления (*CAM*) и инженерных расчетов (*CAE*).

Для вывода 3*D*-моделей могут использоваться такие средства, как дисплей, проектор и 3*D*-принтер.

Спутниковая радионавигационная система или, как она еще называется, *глобальная система определения местоположения* (*Global Position System*, *GPS*) обеспечивает высокоточное определение координат и скорости объектов в любой точке земной поверхности, в любое время суток, в любую погоду, а также точное определение времени.

Принцип работы заключается в следующем. В приемнике измеряется время распространения сигнала от искусственного спутника Земли (ИСЗ) и вычисляется дальность «спутник-приемник» (радиосигнал, как известно, распространяется со скоростью света). Поскольку для определения местоположения точки нужно знать три координаты (координаты на плоскости и высоту), то в приемнике должны быть измерены расстояния до трех различных ИСЗ. Очевидно, при таком методе радионавигации (он называется беззапросным) точное определение времени распространения сигнала возможно лишь при наличии синхронизации временных шкал спутника и приемника.

Современные *GPS*-приемники имеют от пяти до двенадцати каналов, т. е. могут одновременно принимать сигналы от такого количества ИСЗ. Избыточные измерения (сверх трех) позволяют повысить точность определения координат и обеспечить непрерывность решения навигационной задачи.

В состав системы входят:

- созвездие ИСЗ (космический сегмент);
- сеть наземных станций слежения и управления (сегмент управления);
- собственно *GPS*-приемники (аппаратура потребителей).

Для того чтобы все спутники передавали одну частоту, *GPS*-сигналы модулируются специальным кодом. Этот код состоит 1023 нулей или единиц и известен как *C*/*A*-код. Он повторяется непрерывно и из-за своей уникальной структуры позволяет приемнику идентифицировать от каждого спутника его сигнал. Модули *GPS* имеют последовательный интерфейс (формат *NMEA*, сертифицирован Национальной морской авиацией) и интерфейсы оборудования *S-X*-02007-*C*. Большинство производителей часто предлагают специальные интерфейсы своих *GPS*-приемников.

Ввод крупноформатных изображений. Существует несколько вариантов построения технических средств аппаратно-программных комплексов ком-

бинированного ввода графических изображений – на основе крупноформатных дигитайзеров и малоформатных планшетных сканеров.

Использование дигитайзеров. Данные устройства представляют собой планшеты с указателями координат, которые позволяют, разместив на планшете, например, карту, оцифровать все интересующие пользователя объекты, последовательно перемещая указатель вдоль границ либо осевых линий объекта, и через малые промежутки, указывая точки, координаты которых будут автоматически введены в компьютер. Данный способ достаточно трудоемкий, трудно контролируемый по полноте и точности ввода.

Сканирование при помощи крупноформатных рулонных, планшетных или барабанных сканеров. Данный способ наиболее предпочтителен для пользователя. Однако высокая стоимость крупноформатных сканеров сдерживает их широкое применение. К тому же в силу своих конструктивных особенностей эти устройства также не лишены недостатков: громоздкость, невозможность ввода с нестандартных носителей, затруднительное использование в мобильных приложениях.

Аппаратно-программные комплексы пофрагментного сканирования строятся по принципу: вводимое графическое изображение размещается в крупноформатной двухкоординатной системе, затем подвижная малогабаритная платформа со сканирующей фотоприемной ПЗС-линейкой перемещается на траверсе в пределах рабочей области упомянутой координатной системы. Типичным представителем данного технического решения является планшетный сканер на основе плоттера со встроенной сканирующей головкой. При сканировании производится взаимное координатное перемещение сканирующей головки относительно жестко зафиксированного на рабочей поверхности плоттера графического изображения. Сшивка полос производится автоматически с помощью специализированного программного обеспечения.

Технология комбинированного ввода. Комбинированный ввод характеризуется возможностями пофрагментного сканирования крупноформатных изображений, их оцифровкой и комбинированным использованием этих технологий. Для ввода применяются свободно перемещаемые малогабаритные и относительно дешевые средства (сканеры, видеокамеры). Для координатной привязки сканеров и векторной дигитализации используют модификации крупноформатных дигитайзеров.

Комбинированный ввод обеспечивает более высокую точность и производительность по сравнению с «дигитайзерным» вводом; относительную дешевизну возможность избирательного ввода и привязки отдельных актуальных фрагментов изображения преемственность с существующими системами ввода с помощью дигитайзера или сканера.

Сшивка изображений. Большие изображения, как правило, представляются совокупностью кадров с некоторым перекрытием соседних кадров. Если используется оптико-электронная система ввода с ПЗС-матрицей, то каждый кадр представляется в виде матрицы пикселей, а изображение в целом – в виде матри-

цы кадров. Задача формирования полного изображения по совокупности кадров получила название сшивки кадров. Основной задачей при сшивке является определение оптимального совмещения соседних кадров в области перекрытия.

В ряде технических приложений совмещаемые кадры не имеют выделенных опорных точек, а известны лишь среднее значение перекрытия и некоторое допустимое отклонение от заданного значения. Примером таких изображений могут служить изображения топологических слоев интегральных схем, получаемых при восстановлении топологии в технологии обратного проектирования. Более того, даже при наличии реперов или опорных точек возможны смещения пикселей на изображениях из-за изменения масштаба, неперпендикулярности перемещений и др. В этом случае сшивка кадров изображений основана на анализе информации в областях перекрытия соседних кадров и определении степени отличия перекрывающихся областей (используется функция расстояния) или их подобия (используются корреляционные функции), которые бы с максимально возможной точностью и достоверностью позволяли локализовать фрагмент.

В двухмерном измерении евклидово расстояние между $p_1(x_1, y_1)$ и $p_2(x_2, y_2)$ определяется с помощью формулы

$$\| p_1 - p_2 \| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

В качестве эквивалента функции часто используют квадрат расстояния. Расстояние можно также измерять функцией

$$\| p_1 - p_2 \| = d_{abs}(p_1, p_2) = | x_1 - y_1 | + | x_2 - y_2 |.$$

Напомним четыре неотъемлемых свойства функции расстояния:

1) $d(x, y) = d(y, x)$;

2) $d(x, x) = 0$;

$d(x, y) \le d(x, z) + d(z, y)$ – кратчайшее расстояние между двумя точками;

3) для двух точек x и y функция расстояния должна быть вещественной, конечной и положительной: $0 \le d(x, y) < \infty$.

На практике функции расстояния могут дополнительно учитывать контраст, освещенность, шум и другие факторы, которые входят в формулы как слагаемые или коэффициенты:

$$p(x, y) = ap(x, y) + b + \varepsilon(x, y), \qquad (2.2)$$

где a – параметр контраста;

b – параметр средней освещенности;

$\varepsilon(x, y)$ – параметр шума.

Мера сходства может быть определена через функцию расстояния: считается, что пиксели схожи, если

$$d(p_1, p_2) \le \lambda,$$

где λ – некоторый установленный порог.

Если в качестве меры различия в некотором пикселе (k, l) будем брать среднеквадратичную ошибку

$$\varepsilon^2(k,l) = \sum_x \sum_y [p_1(x,y) - p_2(x+k, y+l)]^2 , \qquad (2.3)$$

которая минимизируется перебором всех допускаемых сдвигов эталона по заданной области контролируемого снимка, то приходим к корреляционной функции подобия. Считается, что в точке экстремума реализуется сходство, если

$$\varepsilon^2(k, l) \le \lambda,$$

где λ – некоторый установленный порог.

Из требования минимума ошибки ($\varepsilon^2(k, l) = 0$) приходим к выражению

$$\varepsilon^2(k, l) = \sum_x \sum_y [u(x,y)]^2 - \frac{\left[\sum_x \sum_y u_0(x,y)\, u(x,y)\right]^2}{\sum_x \sum_y [u(x,y)]^2},$$

где u – заданный фрагмент изображения;
u_0 – эталон, соответствующий фрагменту u.

Корреляционный подход характеризуется большой вычислительной сложностью. Используя специальные преобразования, можно ускорить вычисления корреляции. Хороший результат дает дополнительное использование фильтрации и автокорреляционных функций Фурье. Широко используется при построении наилучшего совмещения перекрывающихся областей способ, когда выбираются некоторые общие области, которые существенно меньше, чем заданные совмещаемые области. В рамках каждого из этих подходов возможно также использование алгоритмов параллельного совмещения для ускорения процесса вычисления.

2.5 Системные периферийные интерфейсы

По способу использования периферийные интерфейсы (периферийные шины) делятся на системные и специализиванные (для подключения накопителей данных, обеспечения работы звуковой и графических подсистем, устройств

печати и сканирования и др.). Выделяют также подмножество универсальных интерфейсов.

К *системным периферийным шинам* (интерфейсам) можно отнести: *PCI, PCI-Express, PCI-X, LPC*.

Шина PCI (*Peripheral Components Interconnect*) является синхронным параллельным электрическим интерфейсом с общей средой передачи данных (топология «шина»), состоит из мультиплексированных линий передачи адреса и данных (разделение по времени) и линий различных управляющих сигналов. В транзакции обязательно имеется одна фаза адресации и одна или несколько фаз передачи данных. Используется механизм *Bus Mastering*, который фактически заменяет механизм с выделенным контроллером *DMA*: каждое устройство самостоятельно осуществляет доступ к системной памяти, выполняя все функции контроллера *DMA*.

Основные характеристики шины PCI:

- разрядность (ширина) – 32 или 64 бита;
- тактовая частота – 33,3 или 66,6 МГц;
- адресация – 32 или 64 бита (не зависит от ширины шины);
- пропускная способность – от 133 до 533 Мбайт/с в зависимости от реализации;
- количество подключаемых устройств зависит от реализации, но не более 32 для одного физического сегмента шины;
- конфигурирование – все устройства, а также функции в пределах физического устройства имеют блок регистров размером 256 байт, доступный только через конфигурационный цикл транзакции;
- электрический интерфейс – 5 или 3,3 В.

Хост – источник команд и основной потребитель данных; для компьютера *x*86 это системное ядро, т. е. процессор и системная память. Хост подключен через главный мост (*Host bridge*), который является устройством *PCI* и действует от имени хоста. Хост занимается также распределением ресурсов и конфигурированием всех устройств *PCI*. Мосты играют роль арбитров, обрабатывая запросы от устройств на доступ к шине и отслеживая соблюдение протокола обмена.

Для контроля достоверности передачи информации применяется проверка четности адреса и данных. Для повышения производительности используются отложенные транзакции.

Шина PCI-X является модификацией *PCI* 64, направленной на улучшение ее ключевых характеристик, прежде всего пропускной способности и надежности за счет усложнения протокола обмена данными и увеличения тактовой частоты. Все транзакции по длине разделены на два типа:

- *пакетные (Burst)* – все команды, обращенные к памяти, кроме *Memory Read DWORD*;
- *одиночные* размером в двойное слово (*DWORD*) – остальные команды.

В *PCI-X* введено понятие последовательности (*Sequence*) – одной или нескольких логически связанных пакетных транзакций (чтение или запись в память), в рамках которых передается единый блок данных; *добавлена фаза ат-*

рибутов, следующая за фазой адресации перед фазами данных, в которой инициатор сообщает свой идентификатор.

Основные характеристики шины PCI-X:

- разрядность данных/адреса составляет 64 бита (допускается расщепление на четыре независимых 16-битных шины, что применяется во встраиваемых и промышленных системах;

- тактовая частота – 66, 100 и 133 МГц – зависит от количества устройств (66 МГц – для четырех устройств, 100 МГц – для двух, 133 МГц – для одного);

- максимальная пропускная способность возросла до 1064 Мбайт/с), так как применена технология удвоенной передачи данных (*Double Data Rate, DDR*), когда данные передаются на спаде и возрастании тактового импульса (2128 Мбайт/с), и учетверенной передачи данных (*Quad Data Rate, QDR*) (4256 Мбайт/с.);

- конфигурационное пространство расширенно до 4 Кбайт;

- напряжения питания и уровни сигналов – 3,3 или 1,5 В.

Для контроля достоверности передачи информации применяется механизм коррекции ошибок четности ECC, который позволяет исправлять только одиночные ошибки и обнаруживать большинство ошибок с большей кратностью.

В *PCI-X* отложенные транзакции заменены на расщепленные транзакции.

Шина PCI Express (*PCIe*) вместо шинного соединения использует схему объединенных через коммутаторы встречных симплексных каналов связи между устройствами и портами. Каждый канал является низковольтной дифференциальной парой сигналов (≤ 2.6 В). В целях масштабирования соединение может агрегировать несколько линий. Спецификация предусматривает следующие конфигурации соединения: *x*1, *x*2, *x*4, *x*8, *x*12, *x*16, *x*32. Данные по разным линиям передаются побайтно, общий поток делится на блоки, кратные количеству линий (*передача параллельная, но не синхронная*).

При разработке *PCI Express* особое внимание было уделено совместимости с *PCI* на уровне механизма конфигурирования, программного доступа и поддержки со стороны ОС и драйверов. При этом требовалось сохранить или уменьшить стоимость реализации при значительном улучшении всех характеристик, прежде всего пропускной способности.

Основные характеристики шины PCI Express:

- тактовая частота не декларируется, так как в технологии *PCI Express* не предусмотрен тактовый генератор, и устройства вправе начинать передачу в любой момент. Проблема отличить последовательность нулевых бит от простоя канала решается при помощи кодирования 9/10 или 128/130 в последних версиях шины;

- пропускная способность (скорость передачи) одного канала варьируется от 2,5 ГТ/с (гигатранзакций в секунду), что равно 2 Гбит/с (используется кодирование 8/10 или 250 Мбайт/с) до 16 ГТ/с, что равно 1969 Мбайт/с (используется кодирование 128/130 Мбайт/с);

- механизм конфигурирования аналогичен *PCI-X* (конфигурационное пространство 4 Кбайт), но для упрощения доступа к конфигурационным регистрам

предусмотрен механизм их отображения на пространство памяти, когда по заданному базовому адресу находится пространство для всех возможных устройств в рамках системной шины;

- напряжение питания и уровни сигналов – 3,3 или 1,5 В.

Для контроля достоверности передачи информации применяется 32-битный *CRC*-код.

Корневой комплекс – это аналог главного моста (*Host Bridge*) в шине *PCI* – отвечает за связь с процессором и системной памятью, а также за конфигурирование (подключение) устройств через коммутатор, который является аналогом моста дополнительных шин PCI.

В отличие от *PCI* протокол *PCI Express* условно разделен на три уровня (транзакций, канальный и физический), на каждом выполняется сборка и разборка пакетов и их обрамление необходимыми заголовками и контрольными суммами. Не все пакеты относятся к уровню транзакций, существуют пакеты только канального уровня, служащие для управления.

В *PCI Express* имеется поддержка дифференцированных классов по качеству обслуживания (*Quality of Service, QoS*), обеспечивающая *следующие возможности*:

- выделять ресурсы соединения для потока каждого класса (виртуальные каналы);
- конфигурировать политику по *QoS* для каждого компонента;
- указывать *QoS* для каждого пакета;
- создавать изохронные соединения.

Виртуальный канал представляет собой физически обособленные наборы буферов и средств маршрутизации пакетов, которые загружаются только обработкой трафика своего виртуального канала. На основе номеров виртуальных каналов и их приоритетов производится арбитраж при маршрутизации входящих пакетов. Каждый порт, поддерживающий виртуальные каналы, выполняет отображение пакетов определенных классов на соответствующие виртуальные каналы. При этом на один канал может отображаться произвольное число классов. По умолчанию весь трафик маркируется нулевым классом (*Traffic Class* 0, *TC*0) и передается нулевым виртуальным каналом. Виртуальные каналы вводятся по мере необходимости.

В протоколе *PCIe* для передачи пакетов по каналу связи используется механизм управления потоками на основе кредитов доверия. Принимающее устройство выдает кредиты доверия на основе объема буферной памяти, имеющейся у принимающего устройства. Передающему устройству запрещено начинать транзакции, для которых может потребоваться больше кредитов доверия, чем их заявило принимающее устройство. Для осуществления транзакции длина полезных данных, указываемая в заголовке запроса передающего устройства, должна точно совпадать с объемом передаваемых полезных данных и быть меньше или равной количеству кредитов доверия, имеющихся у принимающего устройства. Это может неоправданно ограничивать гибкость при передаче данных.

Шина LPC (*Low Pin Count*) является синхронной, параллельной, мультиплексированной. Разработана в 1992 году корпорацией *Intel* для замены шины *ISA* (*Industrial Standart Arhitecture*), сохранив программную совместимость для системных и периферийных устройств, входящих в архитектуру *IBM PC XT/AT*. Обычно используется для подключения единственного физического устройства – моста *Super I/O*, но также может поддерживать *IO*-контроллеры, *BIOS Firmware*, аудиокодеки. Не поддерживает общий механизм конфигурирования и *Plug&Play* ввиду специфики подключаемых *Legacy*-устройств (с фиксированными адресами, заложенными в архитектуре системы).

Основные характеристики шины LPC:

- топология – управляемая хостом шина, но чаще используется соединение «точка-точка»;
- разрядность (ширина) – 4 бита (для передачи одного байта требуется два такта);
- тактовая частота – 33,3 МГц;
- адресация памяти – 32 бит, портов – 16 бит;
- пропускная способность – от 16,67 Мбайт/с;
- количество подключаемых устройств зависит от реализации, но не более 32 для одного физического сегмента шины;
- конфигурирование выполняет процедура *POST* (*Power-On Self-Test* – самотестирование после включения), если *BIOS* поддерживает технологию *Plug&Play*, или ОС при загрузке. Блок конфигурационных регистров имеет размер 256 байт;
- электрический интерфейс совпадает с *PCI* – 3,3 В.

Интерфейс *LPC* использует ряд сигналов шины *PCI*, в частности, импульсы синхронизации. Контроллер *LPC* является мостом *PCI*. Интерфейс программно прозрачен и не требует каких-либо драйверов. Передача данных с помощью *DMA* выполняется под управлением хоста.

2.6 Специализированные периферийные интерфейсы

Специализированные интерфейсы подключения накопителей включают интерфейсы *ATA, Serial ATA* (*SATA*), *eSATA. SCSI, SAS, FC-AL*.

Интерфейс ATA (*Advanced Technology Attachment*) или *IDE* (*Integrated Drive Electronics*) предназначен для обмена данными с вынесенным на внешнее устройство контроллером накопителей (жестких дисков и оптических дисководов): передачи и приема данных, подачи команд, отслеживания ошибок, доступа к управляющим и статусным регистрам. Разработкой и стандартизацией интерфейса *ATA* занимается рабочая группа *T*13 института *INCITS*, входящего в организацию *ANSI*.

Исходный стандарт *ATA* имел следующие *основные характерискики*:

- 16-битная шина данных;
- тактовая частота – 8,33 МГц;

- адресация *CHS* (*Cylinder* : *Head* : Sector) задает сектор в виде трехзначного кода (номер цилиндра : номер гловки : номер сектора), имеющего размерность 16 + 4 + 8 бит, что при размере сектора 512 Кбайт ограничивает объем диска 137-ю гигабайтами (соответствует логической адресации *LBA*-28 (*Logical Block Address*) от 0 до 2^{28};

- пропускная способность – от 3,3 до 8,3 Мбайт/с;

- количество подключаемых устройств и конфигурирование – два устройства на шине, причем эти устройства должны были быть жесткими дисками, соединенными по схеме «ведущий/ведомый» (*master*/*slave*);

- электрический интерфейс имеет уровни *TTL* (высокий уровень – от 2,4 до 5,5 В, низкий – от –0,5 до 0,8 В);

- 40/44-контактный разъем и кабель;

- два базовых режима доступа к памяти – PIO и DMA.

Последующие разработки направлены на повышение производительности обмена данными, обеспечение достоверности передачи даных с примененем кодов *CRC* и *ECC*,поддержку большего количества винчестеров и накопителей большой емкости, включая ПУ, отличные от жестких дисков.

Стандарт *ATA*-2 (*Enhanced IDE, EIDE, Fast ATA*, *Fast ATA*-2 и *Ultra ATA*) содержит следующие основные дополнения к первоначальному стандарту *ATA*:

- пропускная способность в режиме *PIO* – от 4,1 до 16,7 Мбайт/с, в режиме DMA – свыше 20 Мбайт/с;

- подключение к одному контроллеру до четырех устройств;

- поддержка приводов *CD* и *DVD*, накопителей *LS*-120 и *ZIP*, магнитооптики, стримеров и т. п.);

- адресация *CHS*/*LBA* для дисков емкостью до 8,4 Гбайт.

Официальная поддержка *ATA*-2 прекращена в 2001 году.

Начиная с *ATA*-3 протоколы включают:

- *SMART* (*Self-Monitoring, Analysis and Reporting Technology*);

- пароли;

- средства управления энергопотреблением (*APM – Advanced Power Management*).

Начиная с *ATA*-4 подключен пакетный протокол (*ATAPI – ATA Packet Interface*) – стандарт, созданный для подключения *CD-ROM* к стандартному разъему *ATA*. Протокол дополнен:

- защитой передаваемых данных кодами *CRC*;

- возможностью создания защищенной области данных (доступной только ОС), данная возможность получила название *HPA – Host Protected Area*;

- распараллеливанием работы жестких дисков;

- оптимизацией очереди команд (минимизацией перемещений магнитной головки при выполнении всех команд, находящихся в очереди на выполнение);

- режимом передачи данных *Ultra-DMA*, который обеспечивает скорость до 33 Мбайт/с;

- новым восьмидесятижильным кабелем, обладающим повышенной помехозащищенностью.

В *ATA*-5/*ATAPI*-5 появляются новые режимы защиты и управления. Пропускная способность в режиме *DMA* увеличилась до 66,6 Мбайт/с (с обязательным использованием восьмидесятижильного кабеля).

В *ATA*-6/*ATAPI*-6 добавлены:

- управление уровнем шума;
- поддержка аудио- и видеопотоков;
- адресация *LBA*-48, позволяющая адресовать 2^{48} секторов, что обеспечивает поддержку дисков емкостью до 144,12 Пбайт (адресация *CHS* удалена);
- расширенные журналы *SMART*;.
- режим DMA с пропускной способностью до 100 Мбайт/с (для ее обеспечения необходим восьмидесятижильный кабель).

В *ATAPI*-7 включен последовательный интерфейс *Serial ATA* в качестве второго варианта физической/электрической реализации, пропускная способность в режиме DMA увеличилась до 133,3 Мбайт/с.

ATAPI-8 содержит аппаратное шифрование, датчик свободного падения, проверку чтением после записи, дополнительный протокол управления параметрами жесткого диска и чтения расширенной информации, флэш-память с возможностью управления ее энергосбережением.

Интерфейс Serial ATA (*SATA*) – последовательный интерфейс обмена данными с накопителями информации. *Его характеристики*:

- обеспечивает повышение скорости передачи;
- удешевление и улучшение кабелей и коннекторов;
- предоставляет выделенный интерфейс для каждого устройства;
- упрощает процесс конфигурирования, переход от шинной топологии к подключению «точка-точка».

SATA – это последовательный вариант реализации транспортного уровня интерфейса *ATA* (начиная с версии *ATAPI*-7) с топологией «звезда». Существуют три основных варианта *SATA*, отличающиеся друг от друга пропускной способностью: *SATA* I – 1,5 Гбайт/с, *SATA* II – 3 Гбайт/с, *SATA* III – 6 Гбайт/с. Данные кодируются по схеме 8/10 и скремблируются, после чего по физической линии передаются по методу *NRZ* (уровень сигнала соответствует передаваемому биту) по двум дифференциальным парам в обоих направлениях. Между канальным и прикладным уровнем имеется транспортный уровень, отвечающий за доставку кадров. На каждом уровне имеются свои средства контроля достоверности и целостности.

В *SATA* получилось легко реализовать механизм последовательного запуска двигателей винчестеров (*Staggered Spin-up*), что важно в тех случаях, когда их в системе много, и одновременный старт приводит к тому, что блок питания не может выдать требуемый номинал тока. Для оптимизации операций по чтению и записи используется техника накопления этих данных в промежуточном хранилище – кэше.

Контроллер *SATA* полностью эмулирует работу стандартного контроллера *ATA* для обеспечения совместимости с ПО (по умолчанию каждое подключенное устройство считается *Master*-устройством на отдельном канале). Воз-

можен также режим *Legacy*, когда каждое устройство становится либо *Master*, либо *Slave*.

Следует отметить следующие *перспективы интерфейса SATA*:

- предлагает более производительную альтернативу таким внешним интерфейсам, как *USB* и *FireWire*;

- для подключения внешних накопителей разработан *External SATA* (*eSATA*);

- расширенная версия *xSATA* позволяет использовать кабель длиной до восьми метров вместо двух.

SCSI (*Small Computer System Interface*) – специализированная внешняя параллельная шина, предназначенная для обмена данными по внешним каналам как между компьютерами, так и между компьютером и устройствами ввода/вывода и обработки данных. Изначально разрабатывался как расширение системной шины, но популярность приобрел в качестве периферийного. Шина *SCSI* учитывает специфику устройств различного класса: хранения данных, управления носителями данных, графического ввода (сканеры), вывода на твердый носитель (принтеры), коммуникации (модемы). Она позволяет использовать различные физические интерфейсы для передачи данных. Имеется возможность встраивать новые интерфейсы в виде нижнего уровня протокола (например, *FireWire*, *TCP*/*IP*).

SCSI не является «хост-центрическим». В рамках архитектуры *SCSI*-устройством являются как хост-адаптер (контроллер *SCSI* с точки зрения системы), так и контроллер ПУ. Причем количество хост-адаптеров в принципе не ограничено, что позволяет, например, организовать совместный доступ к ПУ нескольких систем. Все устройства равноправны, могут выступать как инициаторами, так и целью транзакций.

Существует много различных спецификаций данного интерфейса, отличающихся пиковой пропускной способностью, максимальным числом подключаемых устройств, максимальной длиной кабеля. Стандартизованы три основных варианта SCSI:

- SCSI-1 (1986 год) – имеет тактовую частоту 5 МГц, восьмибитную шину и максимальную скорость передачи – 1,5 Мбайт/с в асинхронном режиме и 5 Мбайт/с в синхронном режиме;

- SCSI-2 (1989 год) – имеет тактовую частоту 10 МГц, максимальную скорость передачи – до 10 Мбайт/с (вариант Fast SCSI с восьмибитной шиной) и до 20 Мбайт/с (вариант Wide SCSI с шеснадцатибитной шиной), максимальная длина кабеля равна трем метрам;

- SCSI-3 (1992 год) – имеет тактоаую частоту от 20 до 160 МГц, максимальную скорость передачи – до 20 Мбайт/с для восьмибитной шины и до 40 Мбайт/с для шеснадцатибитной шины, максимальная длина кабеля равна двенадцати метрам.

Последующие модификации (начиная с 1997 года) получили название Ultra SCSI, пропускная способность которых от 80 до 640 Мбайт/с, максимальная длина кабеля равна двенадцати метрам, рассчитаны на использование в мощных машинах-серверах и рабочих станциях.

Заметим, что реальная пропускная способность намного ниже из-за накладных расходов со стороны самого устройства SCSI, главного адаптера шины, оборудования хоста, драйвера ввода/вывода и операционной системы. Кроме того, собственно передача данных занимает лишь часть времени при выполнении команды, а передача команд осуществлялась в асинхронном режиме. В результате влияния всех факторов общие накладные расходы могут достигать 90 %.

Существует несколько электрических реализаций физического интерфейса *SCSI*:

- линейный (*SE – Single Ended*) – сигналы имеют уровень *TTL*, для передачи каждого сигнала используется отдельный проводник;
- высоковольтный дифференциальный (*HDV – High-Voltage Differential*) – пара сигналов в противофазе, уровни *TTL*;
- низковольтный дифференциальный (*LVD – Low-Voltage Differential*), уровни сигналов уменьшены до 0,9–1 В.

При этом разъемы и кабели используются одни и те же.

Для подключения жестких дисков применяется только *LVD*, так как он позволяет достигать высоких скоростей. Но хост-контроллер устанавливает тот режим, который гарантированно поддерживает самое старое устройство, по схеме *LVD* → *HVD* → *SE*. При этом устройства *LVD* обычно имеют возможность переключаться в режим *SE*, если того требует контроллер, теряя пропускную способность.

Асинхронная передача данных является обязательной для всех устройств *SCSI* и всех фаз передачи информации. Направлением передачи информации управляет целевое устройство (ЦУ) с помощью сигнала *I*/*O*:

I/*O* = 0 – передача Инициатор → ЦУ, *I*/*O* = 1 – передача ЦУ → Инициатор.

Передача каждого байта сопровождается взаимосвязанной парой сигналов *REQ* / *ACK*.

Инициатор фиксирует принимаемые данные по отрицательному перепаду сигнала *REQ*, ЦУ считает принимаемые данные действительными по отрицательному перепаду сигнала *ACK*.

Синхронная передача данных является опцией.

Дальнейшим развитием *SCSI* является разработка протокола, который базируется на *TCP*/*IP* (*iSCS*I – *Internet SCSI*), ориентированный на обеспечение взаимодействия распределенных систем хранения данных, серверов и клиентов.

Fibre Channel Arbitrated Loop (FC-AL) представляет собой спецификацию передачи данных по оптоволоконным и электрическим (коаксиальным кабелям) каналам связи, поверх которой работает *SCSI*-3. *FC-AL* представляет собой устаревший сетевой интерфейс, описанный пятиуровневой моделью.

На нижнем уровне *FC*-0 описан интерфейс с топологией типа «кольцо», «точка-точка» или «звезда» (с трансляцией кадров от устройства к устройству. Дуплексное соединение реализуется при помощи двух противоположно-направленных симплексных каналов.

На уровне *FC*-1 определен протокол передачи, включая алгоритмы преобразования параллельного кода в последовательный и обратно и контроль ошибок.

Уровень *FC*-2 является транспортным, в функции которого входят формирование кадров из передаваемой последовательности байт, различные механизмы управления и средства диспетчеризации последовательности передаваемых данных.

На уровне *FC*-3 определен набор функций для повышения эффективности передачи, таких как распределение данных по нескольким портам, работающим параллельно для увеличения полосы пропускания канала, или доставка широковещательных сообщений множеству портов-получателей.

Уровень *FC*-4 является высшим в стеке протоколов и определяет прикладной интерфейс (он может быть реализован в том числе и *SCSI*).

FC-AL обеспечивает пропускную способность до 4 Гбит/с (400 Мбайт/с с учетом кодирования 8/10). Главное преимущество интерфейса – длина кабеля, которая для коаксиального кабеля равна 30 м, а для оптоволокна может достигать 10 км. *FC-AL* поддерживает до 126 самоконфигурирующихся устройств с возможностью «горячего» подсоединения без перезагрузки.

FC-AL в основном ориентирован на серверные приложения уровня *high-end*, где необходимо иметь быстрый доступ к ресурсам и где высокая помехоустойчивость является одним из важнейших преимуществ. Сверхдлинные кабели также важны для этой области приложений, так как зеркальные диски обычно располагаются в разных зданиях для надежной защиты от непредвиденных ситуаций. Спецификация *FC-AL* является естественным решением для создания на ее базе гигабитных опорных магистралей и локальных сетей, которые требуют широкой полосы пропускания для обработки изображений, видеоинформации и передачи сверхбольших объемов данных.

Интерфейс Serial Attached SCSI (*SAS*) построен по той же концепции и на базе тех же аппаратных решений, что и *Serial SATA*, и у них очень много общего. В частности, была обеспечена полная совместимость контроллеров *SAS* с устройствами *SATA* за счет унификации электрического и физического интерфейсов, включения канального и транспортного слоев *SATA* в состав архитектуры *SAS*. Основная цель такого решения – достижение максимальной гибкости при формировании массива дисков.

В то же время *SAS* имеет собственную архитектурную модель, в которой протокол *SSP* (*Serial SCSI Protocol*) является частью общей схемы и может быть заменен другими протоколами транспортного уровня. Для физической передачи данных используется интерфейс, аналогичный *SATA*. Однако общая концепция *SAS* намного сложнее, и протокол *SATA* фактически является его подсистемой. *SAS* обеспечивает более сложную схему управления, обмена командами и данными, подключения и обслуживания, нежели *SATA*. Фактически это интерфейс нового поколения, использующий общие принципы *SCSI* (команды, адреса, ресурсы и т. п.).

Предусмотрен один физический интерфейс, но три различных протокола транспортного уровня:

- *Serial SCSI Protocol* (*SSP*) – для серверных жестких дисков и других устройств хранения данных;

- *Serial ATA Tunneling Protocol* (*STP*) – для жестких дисков *SATA*;

- *Serial Management Protocol* (*SMP*) – для задач оптимизации и управления массивами жестких дисков.

Поддержка *SATA* позволяет использовать один и тот же конструктив и набор компонентов для создания как экономичных, так и скоростных дисковых подсистем.

Протокол SAS содержит четыре уровня:

- физический (*phy layer*);
- коммуникационный (*link layer*);
- уровень портов (*port layer*);
- транспортный уровень (*transport layer*).

Для гарантированной доставки данных, передаваемых по интерфейсу *SAS*, для каждого кадра данных, полученных целевым устройством, генерируется квитанция о принятии либо непринятии кадра. Передача данных является неблокирующей, т. е. передатчик не ожидает приема квитанции об успешной доставки предыдущего кадра, а сразу же передает следующий. Управление потоком передачи основано на кредитах. Для того чтобы передать кадр данных, передатчик должен иметь ненулевой кредит.

SAS поддерживает большое количество устройств (более 16 384), в то время как интерфейс *SCSI* поддерживает 8, 16, или 32 устройства на шине, и обеспечивает более высокую пропускную способность (1,5; 3,0; 6,0 или 12,0 Гбит/с). Такая пропускная способность может быть обеспечена на каждом соединении Инициатор – ЦУ, в то время как на шине *SCSI* пропускная способность шины разделена между всеми подключенными к ней устройствами. Преимущество *SATA* – в низком энергопотреблении и невысокой стоимости оборудования, а интерфейса *SAS* – большей надежности.

Специализированные интерфейсы для обработки звука служат для обеспечения работы звуковой подсистемы ПЭВМ (звуковых карт, аудиокодеков) и подключения акустических систем.

Широко используются три цифровых аудиоинтерфейса:

- *S/PDIF* (*Sony/Philips Digital Interconnect Format*) – предназначен для передачи *PCM* данных и сжатых аудио потоков *MPEG*4, *AC*3, *DTS* и т. п. (Межгосударственный стандарт *IEC* 60958). Средой передачи является электрический коаксиальный, либо оптический пластиковый кабель. Данные организованы в 32-битные субкадры, в которых передаются выборки 20 или 24 бита и использованием частотной модуляции (*FM*). Данные защищены битами четности, субкадры отделяются последовательностью синхронизации;

- *AC-Link* – служит для подключения кодеков *AC*'97 к цифровому контроллеру *DC*'97 (*Digital Controller*). Архитектура *AC*'97 была разработана *Intel* в конце 90-х годов для стандартизации подсистемы ввода-вывода аудио в ПЭВМ типа *x*86. Она определяет параметры и протокол взаимодействия двух компонентов ПЭВМ – кодека и контроллера. Разделение кодека и контроллера было необходимо для конструктивного отделения аналоговой части от цифровой (цифровая переносится в чипсет). При этом кодек может обрабатывать либо

только аудиоданные (*AC*'97) и модемные данные (*MC*'97), либо оба типа данных (*AMC*'97). Интерфейс двунаправленный, поддерживает протокол мультиплексирования по времени (*Time Division Multiplexing, TDM*). Передача ведется по 12 слотов в кадре; разрядность слотов, кроме нулевого, – 20 бит;

- *HDA-Link* – применяется для кодеков *HDA* (*High-Definition Audio*). Архитектура *HDA* была разработана *Intel* в 2004 году и впервые внедрена в чипсеты серии 915/910. Ее задача – сменить морально устаревшую и не имеющую перспективы роста архитектуру *AC*'97. Помимо описания способа подключения и управления кодеками, *HDA* формулирует полный интерфейс программирования звуковой подсистемы (*API*), а также протоколы обмена нескольких уровней. Главное отличие от *AC*'97 состоит в большей гибкости, управляемости, расширяемости архитектуры, введении понятий потоков, каналов, прерываний, сообщений, команд и т. д. Ввиду формализации полной программно-аппаратной архитектуры для корректной поддержки *HDA* не требуются специфические драйверы для конкретного чипсета. В отличие от *AC*'97, в *HDA* данные обрабатываются с помощью контроллеров *DMA*, реализованных в составе хост-контроллера. Каждый из потоков, входных или выходных, обрабатывается с помощью назначенного контроллера *DMA*.

Специализированные интерфейсы для обеспечения работы графических подсистем включают интерфейсы подключения видеокарт, мониторов, проекторов и устройств печати и сканирования.

Интерфейсы подключения видеокарт (по времени внедрения) – это *ISA* и *EISA*, интерфейсы для свободного кроссплатформенного медиапроигрывателя *VLC* (*video LanClient*), *PCI*, *AGP*, *PCI Express*. Возможно также использование внешних интерфейсов, однако они обычно не предоставляют достаточной полосы пропускания для осуществления 3*D*-рендеринга. Первые шины не имели специальных возможностей и разделяли полосу пропускания между всеми периферийными устройствами. Начиная с шины *AGP*, видеокарты получили выделенный быстродействующий канал. Необходимость в таком канале связана с большими объемами текстур при 3*D*-рендеринге.

AGP (*Accelerated Graphic Port, ускоренный порт для графической карты*) – это специализированный 32-битный параллельный синхронный интерфейс для подключения видеокарты, рассчитан на топологию «точка-точка». Асинхронными, т. е. не привязанными к фронту синхросигнала *CLK*, являются линии *RST#*, *INTA# – INTD#*, *PME#* и *CLKRUN#*.

Шина *AGP* полностью поддерживает операции шины *PCI*, поэтому *AGP*-трафик представляет себой смесь чередующихся *AGP* и *PCI* операций чтения/записи. Операции шины *AGP* являются расщепленными (*split*). Это означает, что запрос на проведение операции отделен от собственно пересылки данных. Такой подход позволяет *AGP*-устройству генерировать очередь запросов, не дожидаясь завершения текущей операции, что также повышает быстродействие шины.

Графическая карта требует высокоскоростного доступа к системной памяти. Однако ядро системы зачастую не может предоставить устройствам не-

прерывный блок памяти большой длины из-за сильной фрагментации, связанной с активным использованием механизма виртуальных страниц. В качестве решения был предложен механизм ремаппинга памяти, выделенной графической карте. Графической карте выделяется непрерывный блок адресов памяти, который называется *AGP*-апертурой (*AGP Aperture*). Блок делится на страницы. Каждая страница отображается на непрерывный блок физической памяти с помощью таблицы *GART* (*Graphics Address Remapping Table* – таблица переопределения графических адресов). Обращение *AGP*-устройства к апертуре вызывает автоматическую замену одного физического адреса на другой, подкрепленный реальной памятью.

Для подключения мониторов служат аналоговый интерфейс *VGA*, цифровой интерфейс *DVI* (*Digital Visual Interface*), мультимедийные интерфейсы *HDMI* и *DisplayPort*.

Аналоговый интерфейс VGA (*Video Graphics Array* – матрица видеографики) разработан в 1987 году и предназначен для дисплеев на ЭЛТ. *VGA* передает сигналы красного, зеленого и синего цветов (*RGB*) построчно, при этом изменение напряжения означает изменение яркости (напряжение сигнала составляет 0,7 – 1 В), для ЭЛТ оно означает изменение интенсивности луча электронных пушек кинескопа (и, соответственно, яркость светового пятна на экране). Интерфейс *VGA* передает также информацию о горизонтальной развертке (*H-Sync*) и вертикальной синхронизации (*V-Sync*). Все современные видеокарты имеют такой интерфейс или же обеспечивают его при помощи переходника из универсального комбинированного интерфейса *DVI-I* (*DVI-integrated*). Также данным интерфейсом оснащаются некоторые проигрыватели *DVD* и многие плазменные и ЖК-телевизоры. *VGA* обеспечивает разрешение 640×480 точек либо 256 цветов с меньшим разрешением, *SVGA* (*Super VGA*, усовершенствованная матрица видеографики) – разрешение 1280×1024 точки с возможностью отображения 16,7 млн цветов.

Для обеспечения работы дисплеев проекторов с цифровым управлением было разработано несколько альтернативных цифровых интерфейсов подключения. Для передачи данных в них используется протокол *Transmission Minimizing Differential Signaling* (*TMDS*), разработанный *Silicon Image*. Этот протокол использует дифференциальную асинхронную передачу с минимизацией потребления тока и кодированием десятьюбитными символами. Каждый цветовой канал кодируется четырьмя или восемью битами, добавляются биты синхронизации и контрольные биты. Канал *TMDS* является однонаправленным, используется только для передачи данных от видеокарты к монитору.

Из всех стандартов распространение получил только *DVI* (*Digital Visual Interface*) и *HDMI*. Обеспечивается передача цифрового сигнала по одному или двум трехразрядным каналам, а также аналогового сигнала *VGA* и ряда сигналов о состоянии и управлении мониторами. Прогрессивными интерфейсами, конкурирующими с названными выше цифровыми интерфейсами, являются *UDI* и *DisplayPort*.

Цифровой интерфейс UDI (*Unified Display Interface*) является практически аналогом интерфейса *HDMI*, используемого для подключения компьютеров

к современным *HD*-телевизорам, а также будет поддерживать все известные в настоящее время системы защиты контента, что позволит беспрепятственно воспроизводить новые носители, оборудованные защитой от копирования, такие как *HD-DVD* и *Blu-ray*. Основным отличием *UDI* от *HDMI* является отсутствие звукового канала, т. е. *UDI* будет передавать только видеоизображение и целиком рассчитан на работу с компьютерными мониторами, а не с *HD*-телевизорами.

Технология, реализованная в *DisplayPort*, позволяет передавать одновременно как графические, так и аудиосигналы. Основное отличие от *HDMI* – более широкий канал для передачи данных (10,8 Гбайт/с). Другая особенность *DisplayPort* заключается в поддержке функций защиты контента (аналогично *HDMI* и *UDI*). Встроенные средства безопасности позволяют отображать содержимое документа или видеофайла только на ограниченном количестве «санкционированных» устройств, что теоретически уменьшает вероятность незаконного копирования материалов, защищенных авторскими правами. И наконец, разъемы, выполненные в соответствии с новым стандартом, тоньше современных разъемов *DVI* и *D-Sub*. *DisplayPort* является универсальным цифровым интерфейсом, позволяющим подключать дисплеи различных типов (плазменные, жидкокристаллические, ЭЛТ-мониторы и др.) к бытовым устройствам и к компьютерному оборудованию.

Комбинированные интерфейсы. Протоколы такого типа основаны на объединении нескольких протоколов в один последовательный сигнал, использовании специальных портов для подключения различных ПУ.

Аппаратный интерфейс Thunderbolt, разработан *Intel* в сотрудничестве с *Apple*, комбинирует в одном кабеле интерфейсы *PCIe* и *DisplayPort* и обеспечивает максимальные скорости передачи данных между ПЭВМ и ПУ порядка 10 Гбит/с по медному проводу и 20 Гбит/с при использовании оптического кабеля. Контроллеры *Thunderbolt* мультиплексируют один или более каналов данных от подключенных к ним ПУ для передачи через один дуплексный канал *Thunderbolt*, затем демультиплексируют их для использования ПУ. Один порт *Thunderbolt* поддерживает до шести устройств *Thunderbolt*, подключаемых через концентраторы (хабы) или цепочкой.

Новый тип физического разъема USB – Type C может работать в так называемых альтернативных режимах, которые позволяют передавать через разъем и кабели не только *USB*-данные, но и поддерживают видеоинтерфейсы *DisplayPort, HDMI* и *Thunderbolt*3. Одного разъема *USB-C* достаточно для подключения дисплеев, внешних жестких дисков и других ПУ. При этом задействуются другие физические протоколы и обеспечивается передача до 40 Гбит/с.

Интерфейсы подключения принтеров. Специализированный интерфейс для принтеров – *Centronics* (*IEEE* 1284) предназначен для подключения принтеров к параллельным портам *LPT* (*Line Print Terminal*). Изначально *LPT*-порты на материнской плате отсутствовали физически и реализовались дополнительной картой расширения, вставляемой в один из *ISA*-слотов расширения на ма-

теринской плате. Данный интерфейс ограничен в использовании ввиду низкого быстродействия и отсутствия гибкости.

Интерфейс *SCSI* применялся для высокоскоростных сканеров, но сегодня он не актуален. Для подключения принтера чаще всего применяется универсальный интерфейс *USB*. Сетевые принтеры содержат встроенные сетевые узлы и таким образом подключаются с помощью *Ethernet*, *Wi-Fi* и других сетевых интерфейсов. Фотопринтеры позволяют использовать твердотельные носители информации (карточки памяти), оптические диски и *USB*-накопители в качестве источника данных, не требуя подключения к ПЭВМ. В качестве хоста может также выступать портативное фотоустройство. Разработаны протоколы (*PictBridge*, *DirectPrint*, *BubbleDirect* и др.) для обмена изображениями между принтером и фотокамерой.

Интерфейсы подключения и программирования сканеров. Типичными интерфейсами являются *Centronics*, *SCSI*, *USB* и *FireWire* (*IEEE* 1394). Специализированные интерфейсы обычно не используются. Что касается программых интерфейсов, то их большое количество: *TWAIN* (*Technology Without An Interesting Name*), *WIA* (*Windows Image Acquisition*), *ISIS* (*Image and Scanner Interface Specification*), *SANE* (*Scanner Access Now Easy*) и др.

Интерфейс TWAIN разработан в 1992 году, обеспечивает единый способ доступа к устройствам-источникам графических данных. Он предоставляет несколько слоев (*Application Layer*, *Protocol Layer*, *Acquisition Layer*, *Device Layer*), которые имеют стандартный интерфейс и позволяют ПО получить управление устройством сканирования независимо от особенностей его реализации. В рамках *TWAIN* заданы такие объекты, как: приложение, запрашивающее данные; физическое устройство, например, сканер; источник, реализуемый драйвером устройства; менеджер источников, управляющий доступом к источникам.

*Интерфейс WIA р*азработан как аналог *TWAIN*, но в рамках модели *COM* и с учетом особенностей модели драйверов *Windows*. Имеет свою многоуровневую структуру, опирается на драйверы и промежуточные уровни доступа к устройствам, совместим с *TWAIN*.

Интерфейсы подключения клавиатуры. В настоящее время используются так называемые расширенные клавиатуры *AT* или *PS*/2, имеющие более 100 клавиш. Процессор общается с клавиатурой через контроллер интерфейса клавиатуры – микроконтроллер 8042 или программно-совместимый с ним, установленный на системной плате.

Для подключения клавиатуры к компьютеру используются два проводных интерфейса – *PS*/2 и *USB*. Большинство дешевых и сравнительно старых моделей ориентировано на интерфейс *PS*/2. Соединение через порт *USB* (соответствующим разъемом оснащены современные клавиатуры) имеет ряд преимуществ. Например, *USB*-клавиатуру можно подключать и отключать в «горячем» режиме (выполнение данной операции с клавиатурой стандарта *PS*/2 может привести к системному сбою или повреждению контроллера на материнской плате). Интерфейс *AT* (*DIN*) и *COM*-интерфейсы в современных компью-

терах практически не используются. Беспроводные клавиатуры можно подключать с помощью ИК-порта с использованием модулирования светового потока инфракрасного диапазона, низкочастотного радиоканала (частота – десятки мегагерц, чаще в районе 27 МГц) и по высокочастотному радиоканалу (частота около 2,4 ГГц) как напрямую по *Bluetooth*, так и через адаптер.

Для подключения координатных устройств (манипуляторов типа «мышь» и «трекбол», сенсорных панелей, графических планшетов и перьев, игровых устройств) к универсальному последовательному порту (*COM*) используется интерфейс *RS*-232*C*. Также широко используются интерфейсы *PS*/2 (*PS*/2-*Mouse*), *USB* и беспроводные интерфейсы, подобные интерфейсам подключения клавиатуры. С самых первых моделей *IBM PC* был введен и фактически стандартизован интерфейс игрового адаптера – *Game port*. Современные игровые устройства имеют свой микроконтроллер и подключаются к компьютеру цифровым интерфейсом – по шине *USB* или через *COM*-порт. Их функциональные возможности богаче, они позволяют устанавливать и двустороннюю связь с игроком (вводить механические воздействия).

Джойстик обычно подключается к адаптеру игрового порта, расположенному на многофункциональной плате ввода-вывода (*Multi I/O Card*) или звуковой карте (в последнем случае разъем игрового порта совмещается с интерфейсом *MIDI*).

Световое перо было введено во время распространения графических карт стандарта *EGA* (*Enhanced Graphic Adapter* – расширенный графический адаптер), которые обычно имели разъем для подключения светового пера. По своей сути световое перо является расширением видеосистемы. Разъем для подключения светового пера был обязательным для видеоадаптеров *CGA* (*Color Graphic Adapter* – цветной графический адаптер), встречался время от времени у видеоадаптеров *EGA* (*Enhanced Graphic Adapter* – расширенный графический адаптер), но практически исчез с распространением *VGA*.

В игровых манипуляторах, как правило, используются интерфейс *MIDI*, который может присутствовать на материнской плате или звуковой карте. В последнее время для подключения используется шина *USB*, что позволило увеличить количество функций устройств. Кроме того, может быть использовано подключение к последовательному порту, а также имеются комбинированные интерфейсы подключения, например используется интерфейс игрового адаптера и *USB*.

Сенсорная панель TouchPad (с англ. яз. *touch* – касаться, *pad* – подушечка) – устройство ввода данных, обычно используемое в ноутбуках, смартфонах и прочих мобильных устройствах, в которых управление курсором представляет собой жестовый интерфейс. Подобный интерфейс может быть реализован как при помощи устройств координатного ввода с возможностью считывания координаты одной точки касания, так и таких, в которых имеется возможность считывания координат более чем одной точки (так называемое. мультикасание, *multitouch*).

Современные версии ОС, как правило, распознают сенсорное устройство и устанавливают драйвер (фирмы *Synaptics* или собственный), который и обеспечивает правильную работу сенсорного указателя.

Сенсорные устройства могут быть внешними ПУ, тогда они подключаются к ПЭВМ через интерфейс *PS*/2, *USB* или др.

Некоторые из называемых выше интерфейсов подключения ПУ были стандартизованы, и их стали относить к универсальным интерфейсам. Среди параллельных к таковым относится симплексный интерфейс *Centronics* с топологией «точка-точка», логика *TTL*, предназначенный для подсоединения механических печатающих устройств, дальнейшее развитие которого привело к созданию *IEEE* 1284 (1994 год).

Интерфейс IEEE 1284–1994 помимо однонаправленного вывода по протоколу *Centronics* (режим *SPP – Standard Parallel Port*) определяет несколько новых режимов работы порта *LPT*, в том числе двунаправленные и аппаратно управляемые:

- полубайтный (*Nibble mode*) – ввод байта в два цикла по четыре бита;

- побайтный обмен (*Byte mode*) – работает только на портах, допускающих чтение выходных данных;

- режим двунаправленного обмена данными (*EPP*, *Enhanced Parallel Port*) с аппаратной генерацией управляющих сигналов интерфейса во время цикла обращения к порту (чтения или записи в порт). Эффективен при работе с устройствами внешней памяти, адаптерами локальных сетей;

- *ECP* (*Extended Capability Port*) – режим двунаправленного обмена данными с возможностью аппаратного сжатия данных по методу *RLE* (*Run Length Encoding*) и использования *FIFO*-буферов и *DMA*. Управляющие сигналы интерфейса генерируются аппаратно. Эффективен для принтеров, сканеров, программаторов.

Из последовательных интерфейсов к универсальным относят интерфейсы *RS*-232, *USB*, *IEEE* 1394 и беспроводные *IrDA*, *Bluetooth* и *Wi-Fi*.

Универсальный последовательный интерфейс USB строится на основе пересылок пакетов. Пакет пересылается целиком, а синхронизация возможна только по принимаемому потоку бит. Способен передавать изохронный трафик аудио- и видеоданных (отправка кадров данных происходит в заданные, известные приемнику и отправителю моменты времени, при этом данные, передаваемые одним узлом с постоянной скоростью, будут поступать к приемнику с той же скоростью). Протокол *USB* использует циклический избыточный код (*CRC*) для защиты полей пакета.

В основе архитектуры *USB* лежит принцип централизованного управления всей сетью подключенных устройств со стороны одного контроллера, за которым стоит хост-система. Логическая топология *USB* – звезда, физическая топология – многоярусная звезда с хост-контроллером в центре.

В настоящее время интерфейс развивается в трех направлениях:

- *Wireless USB* (*WUSB*) – т. е. способность передавать *USB*-протокол через беспроводное подключение;

- развитие скорости проводного подключения путем внедрения *Hi-Speed USB – USB*2 и *USB*3;

- *On-The-Go* – двухточечное соединение пары ПУ посредством *USB* без участия компьютера.

Возможны также любые совмещения указанных технологий.

Интерфейс IEEE 1394 (*FireWire, i.Link*) – универсальная последовательная шина для объединения в сеть разнообразных цифровых устройств, включая записывающую *AV*-аппаратуру, ресиверы, плееры, рекордеры, музыкальные инструменты, устройства хранения данных, персональные компьютеры и т. д. Основные достоинства этого интерфейса по сравнению с *USB* определяются тем, что *Fire Wire* ориентирован на интенсивный обмен между любыми подключенными к ней устройствами, а *USB* – на взаимосвязь ПУ и ПЭВМ. Допускается использование различных физических топологий (звезда, цепочка, дерево), с ограничениями по длине сегмента (4,5 м), количеству сегментов (16) и количеству устройств (63). Исходная логическая топология – шина – может наращиваться до сети за счет применения мостов.

Шина *FireWire* возникла как развитие *SCSI* в сторону последовательного интерфейса, позаимствованы идеи равнорангового взаимодействия, многоуровневый протокол, арбитраж, модель применения и др.

Последовательный инфракрасный интерфейс IrDA (*Infrared Data Assotiation*) определяет требования к мощности передатчика и чувствительности приемника, методы модуляции и схемы кодирования/декодирования. Стандарт *IrDA* включает в себя стек протоколов трех согласованных обязательных уровней:

- физического (*IrPL, Physical Layer*) – устанавливает стандарты для *Ir*-трансиверов;

- доступа (*IrLAP, Link Access Protocol*) - занимается разбиением данных на блоки, контролем ошибок и другими функциями низкого уровня;

- управления соединением (*IrLMP, Link Management Protocol*) – позволяет по одной линии обмениваться данными между несколькими приложениями.

Транспортный уровень обеспечивается протоколом *TinyTP* (*IrDA Transport Protocol*) – здесь обслуживаются виртуальные каналы между устройствами, обрабатываются ошибки (потерянные пакеты, ошибки данных и т. п.), производится упаковка данных в пакеты и сборка исходных данных из пакетов (протокол напоминает *TCP*). На транспортном уровне может работать и протокол *IrTP*.

Связь в *IrDA* полудуплексная, передача данных идет асинхронно в обоих направлениях. Для обнаружения ошибок используется циклический код *CRC*-8 в коротких пакетах и *CRC*-16 – в длинных.

Наряду с *IrDA* существуют и собственные системы фирм: *Hewlett Packard, Sharp* и др.

Беспроводной интерфейс Bluetooth – технология беспроводной передачи данных по радиоканалу между различными типами электронных устройств с целью обеспечения их взаимодействия, определяется стандартом *IEEE* 802.15. Создание этого интерфейса было начато в 1998 году компаниями *Ericsson, IBM, Intel, Nokia, Toshiba*, а позднее продолжено группой *SIG* (*Special Interest Group*), в

которую входят многие фирмы, в том числе корпорации *Lucent, Microsoft* и др. При разработке интерфейса выдвигались следующие требования: аппаратура должна быть компактной, недорогой и экономичной, т. е. иметь возможность работы при малых значениях потребляемого тока.

Bluetooth использует специальные небольшие приемопередатчики, работает в диапазоне 2,4–2,48 ГГц (диапазон *ISM – Industry, Science, Medicine* – промышленный, научный, медицинский) и предназначен для передачи данных на дальность от 10 до 100 м как цифровых данных, так и звуковых сигналов. Скорость передачи в устройствах, поддерживающих базовые спецификации *Bluetooth* (1.1, 1.2 и 2.0) составляет 1 Мбит/с. В устройствах, поддерживающих ускорение передачи *EDR* (*Enhanced Data Rate*) (с 2004 года, начиная со спецификации *Bluetooth* 2.0), скорость передачи может составлять в зависимости от используемого способа модуляции до 3 Мбит/с.

Применяется скачкообразная перестройка частоты 1600 раз в секунду по псевдослучайному закону или шаблону, составленному из 79 подчастот (принцип *FHSS − Frequency-Hopping Spread Spectrum*). Настройка на один шаблон позволяет устройствам, использующим *Bluetooth*, осуществлять обмен данными, в то время как другие устройства будут воспринимать передаваемую информацию как шум. Для обеспечения информационной безопасности в *Bluetooth* используются односторонняя или двусторонняя аутентификация и шифрование передаваемых данных. Длина ключа шифрования может варьироваться от 8 до 128 бит, что дает возможность регулировать криптостойкость используемого алгоритма шифрования.

В основе технологии *Bluetooth* лежит объединение устройств в пикосети, которые представляют собой два или более устройства (основное и подчиненные), которые используют один и тот же какал. Протокол *Bluetooth* поддерживает соединение типа «точка-точка», а также и соединение типа «точка-многоточка». Обмен данными между устройствами осуществляется на основе разделения времени (*TDD – Time Division Duplexing*). Основное устройство передает пакеты в нечетные временные сегменты, а подчиненное устройство – в четные. При этом частота канала не меняется до окончания передачи пакета.

Для кодирования пакетной информации используется частотная модуляция.

В зависимости от мощности передатчика устройства *Bluetooth* делятся на три класса, различающиеся максимальной выходной мощностью и дальностью связи:

1) мощность 100 мВт, дальность связи до 100 м;
2) мощность до 2,5 мВт, дальность связи до 10 м;
3) мощность до 1 мВт, дальность связи до 1 м.

Протокол *Bluetooth* может поддерживать:

- асинхронный канал данных, который предназначен для пакетной передачи данных и обеспечивает в асимметричном режиме передачу со скоростью до 723,2 Кбит/с (88,3 Кбайт/с) в прямом и 57,6 Кбит/с в обратном направлениях или до 433,9 Кбит/с (53 Кбайт/с) в каждом направлении в симметричном режиме;

- до трех синхронных (с постоянной скоростью) аудиоканалов, каждый из них поддерживает по 64 Кбит/с в обоих направлениях;

- канал с одновременной асинхронной передачей данных и синхронной передачей звука.

При работе устройств *Bluetooth* используются как специфические протоколы *Bluetooth*, так и общие, которые используются в различных телекоммуникационных системах. Все они образуют стек протоколов, которые можно разделить на четыре слоя:

- корневые протоколы;
- протокол замены кабеля;
- протокол управления телефонией;
- заимствованные протоколы.

Корневые протоколы отвечают за обнаружение обслуживания, управление соединением и логическими связями, а также определяют передачу на физическом и канальном уровнях. Они требуются для большинства устройств. Остальные слои совместно определяют совокупность протоколов, ориентированных на приложения, которые позволяют прикладным задачам выполняться над корневыми протоколами.

Следует отметить, что стандарт *Bluetooth* лишь определяет базовые принципы взаимодействия устройств, и на их основе формируютсяч различные профили, например, для передачи файлов, для беспроводной телефонии, для доступа к локальной сети и т. п. Когда два устройства устанавливают соединение, они обмениваются информацией о том, какие профили они поддерживают. После чего они смогут работать друг с другом по тем профилям, которые поддерживаются обоими устройствами.

Беспроводной интерфейс Wi-Fi (*Wireless Fidelity*) обозначает стандарт беспроводной радиосвязи в диапазоне частот от 2,4 до 2,4835 ГГц (обеспечивается максимальная скорость 11 Мбайт/с), который объединяет несколько протоколов и имеет официальное наименование *IEEE* 802.11. Он был зарегистрирован в 1999 году. Технология *Wi-Fi* – беспроводной аналог стандарта *Ethernet*, на основе которого сегодня построена большая часть офисных компьютерных сетей, обеспечивает доступ к серверам, хранящим базы данных или программные приложения, позволяет выйти в Интернет, распечатывать файлы и т. д.

Ядром беспроводной сети *Wi-Fi* является так называемая точка доступа (*Wireless Access Point – WAP*), которая подключается к какой-либо наземной сетевой инфраструктуре (например, офисной *Ethernet*-сети) и обеспечивает передачу радиосигнала. Обычно точка доступа состоит из приемника, передатчика, интерфейса для подключения к проводной сети и программного обеспечения для обработки данных. После подключения вокруг точки доступа образуется территория радиусом до 100 м (ее называют хот-спотом или зоной *Wi-Fi*), на которой можно пользоваться беспроводной сетью.

Перспективным направлением в развитии беспроводных технологий является *WiMAX* (*Worldwide Interoperability for Microwave Access*), стандарт *IEEE* 802.16, созданный, созданной в 1999 году (в форум вошли такие фирмы, как *Nokia, Harris*

Corporation, *Ensemble*, *Crosspan* и *Aperto*). Цель технологии *WiMAX* заключается в том, чтобы предоставить универсальный беспроводной доступ для широкого спектра устройств (рабочих станций, бытовой техники «умного дома», портативных устройств и мобильных телефонов) и их логического объединения – локальных сетей.

Контрольные вопросы

1 Назовите основные типы архитектур вычислительных систем и их особенности.

2 Укажите специфические свойства языков логического управления.

3 В чем заключаются преимущества графовых моделей?

4 Как используется избыточная информация при контроле работы ЦА?

5 Почему контроль передачи и хранения информации отделяется от контроля выполнения арифметических операций?

6 Как обнаруживают и исправляют ошибки в работе ЦА?

7 В чем различие между одноключевыми и многоключевыми системами шифрования?

8 Назовите и опишите принципы сшивки изображений.

9 Назовите и опишите принципы сканирования и отображения графических образов.

10 Какова роль средств интерфейса в решнии задач?

Рекомендуемая литература [2, 4–6, 8, 18–22, 24].

3 ИНТЕЛЛЕКТУАЛЬНЫЕ СРЕДСТВА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

3.1 Подходы к созданию систем искусственного интеллекта

Понятие интеллектуальной технологии. Развитие самой вычислительной техники привело в последние годы к существенному прогрессу в создании систем с элементами «искусственного» интеллекта, позволяющих решать задачи, считающиеся интеллектуальными (творческими) для человека. В первую очередь это были системы, обладающие способностями к анализу конкретных ситуаций и самостоятельному поиску рациональных решений на основе баз знаний, содержащих не только важнейшие для данного процесса сведения, но и правила вывода для выработки необходимой информации. Близкие по характеру результаты были достигнуты в ряде отраслей при использовании экспертных систем, аккумулирующих знания ведущих специалистов в конкретных отраслях с правилами получения, а иногда и объяснения рекомендуемых решений.

Технологии с такими элементами можно назвать «интеллектуальными» информационными технологиями. Роль «интеллектуальных» технологий в перспективе будет возрастать. Такой вывод напрашивается в связи с построением информационного общества во всех передовых странах. В частности, об этом говорят успехи в автоматизации проектно-конструкторских задач, автоматизации управления предприятиями, начатые разработки по созданию «электронных» правительств, управлению космическими полетами, автоматизации военных операций, прогнозированию сложных процессов, управлению транспортными средствами на Земле из космоса и др.

В первых исследованиях в основном решался принципиальный вопрос о возможности создания технических устройств, обладающих творческими способностями. Вскоре был получен и ряд обнадеживающих результатов прикладного плана при попытках моделирования творческой деятельности человека по управлению различными объектами, проектированию, распознаванию образов, доказательству теорем, игре в шахматы и шашки, сочинению стихов, музыки и т. п. Развитие робототехники в последние годы породило еще ряд сложных проблем, связанных с организацией поведения роботов. В области искусственного интеллекта началась работа по формальному представлению систем процедур, обеспечивающих заданное нормативное поведение при заданных знаниях о внешней среде и целях функционирования системы. Это привело к развитию идей, связанных с нахождением эффективных систем представления знаний и планирования целесообразной деятельности. В остальных аспектах продвижение работ в области робототехники существенно опирается на исследования в области машинного интеллекта.

Принципиальная возможность реализации различных алгоритмизируемых процессов на вычислительной машине была доказана еще английским математиком А. Тьюрингом, который развивал функциональный подход к оценке деятельности автоматов. Такой подход нашел сторонников при решении вопросов о возможности машинного мышления.

Если определить мышление как нечто свойственное только человеку, то ответ на вопрос о возможности машинного мышления будет отрицательным. Если же стать на позицию оценки «деятельности» машины по конечным результатам, то ответ будет положительным, что хорошо иллюстрирует следующий опыт (рисунок 3.1).

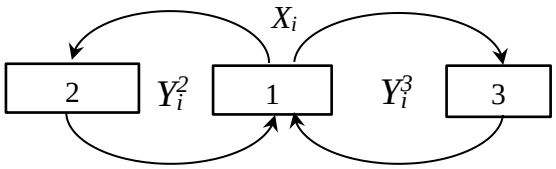


Рисунок 3.1 – Схема эксперимента Тьюринга

Имеются три комнаты. В первой комнате находится человек – экспериментатор, во второй – машина, в третьей – человек, выполняющий определенную работу. Экспериментатор точно не знает, в какой комнате находится человек, а в какой – машина. Ему предлагается задать конечную серию задач (вопросов) и определить по ответам, в какой комнате принимаются решения человеком, а в какой – машиной. Если экспериментатор однозначно не может указать, где находится машина, естественно признать, что машина может «мыслить». Простейшими примерами для эксперимента могут послужить решение шахматных этюдов или задач, проектирование режущего инструмента и т. д.

Работы *по машинному интеллекту ведутся в двух направлениях*:

1) моделирование деятельности мозга на ЭВМ;

2) поиск алгоритмов для решения различных творческих задач.

Исследователи, придерживающиеся первого направления, идут двумя путями. С одной стороны, они стремятся детально изучить функции и свойства элементарной ячейки нервной системы – нейрона, а с другой – научиться «собирать» из этих ячеек сложные конструкции по типу человеческого мозга. На этом пути пока не удалось достигнуть существенных в прикладном отношении результатов, так как имеются трудности, как с описанием самой элементарной ячейки, так и с поиском закономерностей объединения их в сети. Имитировать же эффективно мозг с помощью случайно скоммутированной сети маловероятно. Другой путь связан с моделированием работы мозга на уровне информационных процессов, добиваясь того, чтобы функции модели были по возможности неотличимы от функции моделируемого объекта.

Второе направление представлено серией работ по искусственному интеллекту, представляющих как теоретический, так и практический интерес. Среди них можно выделить ряд результатов, полученных с помощью методов ситуационного управления и эвристического программирования.

Методы ситуационного управления, предложенные Д. А. Поспеловым и Ю. А. Клыковым, ориентированы на те случаи, когда формальное описание модели объекта методами классической математики нецелесообразно или же невозможно. Взамен традиционных подходов в ситуационном управлении строится семиотическая модель объекта и протекающих в нем процессов. Описание

модели базируется на естественном языке. Для процессов проектирования и управления такой подход весьма перспективен.

Мы детальнее остановимся на той ветви второго направления работ по искусственному интеллекту, которое широко известно как *эвристическое программирование*, и остановимся главным образом на его прикладных аспектах, так как в недрах эвристического программирования зародились идеи создания универсальной программы для решения любых задач.

Современное эвристическое программирование занимается исследованием типовых приемов, полезных в процессах решения проблем. Эвристика строится на основе наблюдений за тем, как люди решают задачи на основании общего в решении самых различных проблем.

До начала решения задачи надо быть уверенным, что она поставлена. В современной эвристике рассматриваются лишь некоторые методы решения задач, но не исследуются вопросы, связанные с постановкой проблемы. Методы эвристического программирования в основном направлены на ограничение области поиска решения за счет использования опыта решения данного класса задач или выдвижения некоторых гипотез.

Прежде чем приступить к решению, надо обладать *методом распознавания* удовлетворительного ответа. Иными словами, за конечное число шагов требуется установить, является ли полученный ответ удовлетворительным. С этой точки зрения крайне желательно иметь эффективный алгоритм, гарантирующий получение решения, если оно существует в пределах разумных затрат времени.

Анализ эффективности эвристических программ показывает, что для решения сложных задач необходимо иметь в составе программ стандартные процедуры для осуществления поиска, распознавания, обучения, планирования и индуктивного вывода.

Поиск заключается в проверке пригодности всех предлагаемых решений. Однако подобный путь не представляет практический интерес из-за больших затрат времени. Обычно почти в каждой интересующей нас задаче можно оценивать некоторые предварительные решения и делать следующие попытки в более перспективном направлении.

Распознавание образов позволяет не исследовать все возможности для организации поиска, чтобы сразу применить к конкретной проблеме наиболее эффективный метод. В связи с этим возникает задача классификации методов и распознавания соответствующих им ситуаций. Наиболее простой путь в этом отношении – сравнение неизвестного объекта с рядом эталонов. На практике оказывается, что число эталонов довольно велико. Поэтому пытаются истолковать образ как систему в некотором смысле подобных объектов. Возникает задача выявления хорошей системы признаков, описывающих образы, инвариантных относительно различных преобразований.

Обучение позволяет при решении новых задач использовать методы, оказавшиеся эффективными при решении аналогичных проблем. Такую эвристику обычно реализуют на основе моделей, повышающих ее приоритет при успешном применении. Любая обучающая система должна использовать результаты

прошлого опыта как основу для более общих предположений. Простейшим способом обобщения внутри множества признаков является построение «типичного» члена данного множества (т. е. усреднение). Все же возможности системы с простым повышением приоритета ограничены ее зависимостью от «учителя» (тренировочной последовательностью задач). Один из путей преодоления этой трудности состоит в разработке процедуры обобщения действий «учителя». Тогда при решении задач машина будет сама повышать приоритет эвристик в процессе работы.

Планирование служит для выбора перспективной цепочки подзадач. В ходе решения проблемы приходится сталкиваться с множеством взаимосвязанных подзадач. Выбор подзадачи должен основываться на относительных оценках трудностей, которые встречаются при ее исследовании, и на оценках, характеризующих важность подзадачи для решения всей проблемы. При решении сложных задач применение таких «пошаговых» эвристик не дает результатов. Машина должна обладать способностью к анализу структуры проблемы в целом, т. е. способностью планирования. На практике всякая возможность планирования оказывается полезной, если задача достаточно трудна. Лишь от схем, которые активно продолжают анализ для выработки набора цепочек, можно ожидать, что они смогут осваивать решение задач все возрастающей сложности.

Индуктивный вывод обеспечивает построение общих утверждений о событиях, не встречавшихся ранее. Однако может оказаться, что не существует системы индуктивного вывода, которая работала бы во всевозможных средах. Все же считается, если задана среда и критерий успеха, то эта проблема для машины является чисто технической. Отметим, что в настоящее время не имеется пока программ с развитым индуктивным мышлением.

Все пять видов процедур в той или иной степени были использованы впервые в США в 60-х годах XX века А. Ньюэллом, Дж. Шоу и Г. Саймоном в программе «Общий решатель задач», предназначенной для решения широкого класса задач (доказательство теорем, интегрирование с использованием табличных интегралов, игровые задачи и др.). Они осознали необходимость развития новой теории, опирающейся на имитацию процедур универсального характера, с помощью которых порождались бы конкретные процедуры, направленные на решение определенных задач интеллектуального типа.

В качестве основной ими была взята процедура, имитирующая поиск по лабиринту возможных альтернатив. Авторы программы выдвинули тезис, что решение любой задачи человеком состоит в поиске пути, приводящего от начальной площадки лабиринта, соответствующей исходному описанию ситуаций, к некоторой конечной площадке, соответствующей решению этой задачи. На каждом шаге поиска пути используются локальные критерии успеха, дающие возможность выбора очередного коридора лабиринта. Авторы вначале были уверены, что с помощью найденной ими процедуры удастся решить все интеллектуальные задачи. Первыми успешными экспериментами в этом направлении были доказательство теорем исчисления высказываний и игровые программы. Но проверка универсальности процедуры поиска по лабиринту на

шахматной программе показала ее авторам, что, основываясь лишь на ней, практически невозможно решить сколько-нибудь серьезную задачу.

В прикладном плане, по-видимому, в настоящее время будут иметь успех методы, сочетающие точный и приближенный путь решения задач, т. е. такие методы, где сокращение объема вычислений по точному методу планируется вести за счет использования эффективных эвристик, учитывающих особенности данного класса задач.

Транзакции в сетях. Информационные компьютерные технологии в сетях существенно отличаются масштабами, формой организации и средствами управления базами данных (БД). Наряду с традиционными БД, используемыми в оперативном режиме для решения задачи, создаются дополнительные хранилища и витрины данных. Второй особенностью является использование в локальных сетях информационных технологий и сервисов Интернета. Появляется возможность реализации дополнительных функций типа электронной коммерции, совместной обработки документов, электронного безбумажного документооборота, телеконференций и т. д. В-третьих, появляются специфические алгоритмы поиска и обработки данных, например, обращение к платным хранилищам и услугам.

Специфичен транзакционный подход: на период решения задачи запрещается применение некоторых функций для других проблем и в случае неудачной попытки решить задачу отката системы в исходное состояние.

Основу любых дистанционных операций составляет транзакция. Этот термин используют как в деловом мире электронной торговли, так и в банковских платежах, обращениях с запросом к банкам данных. Однако пользователи, исходя из характера выполняемой операции, могут вкладывать в него иной смысл. Так, транзакция – это:

- сделка, единица коммуникативного процесса;
- единичное действие, имеющее два состояния – выполнено (1) и не выполнено (0);
- минимальная логически осмысленная операция;
- группа взаимосвязанных операций, которые обладают свойствами атомарности, непротиворечивости, изолированности.

Предлагаем использовать в дистанционной работе и дистанционных коммерческих операциях термин «дистанционная транзакция», включающий не только совершение операций (сделки), но и соблюдение ряда требований к ее наиболее ответственным процедурам, касающимся обеспечения конфиденциальности сделки, обеспечения юридической ответственности за передаваемую информацию, невозможность до завершения процедуры всей транзакции начать другую транзакцию, а при срыве выполнения хотя бы одного этапа всю систему затронутых элементов вернуть в исходное состояние. Похожая схема сейчас реализуется при операциях с базами данных в информационных системах. Примерно такая же схема рекомендуется для сделок между хозяйствующими субъектами с требованиями к неделимости, согласованности, надежности и изолированности.

Мультиагентные системы. На заре развития теории искусственного интеллекта решение задачи сводилось к созданию одной системы, способной ее решить. В

дальнейшем по мере возрастания потребностей из различных областей (наука, техника, экономика, управление государством и крупными фирмами в мировом масштабе) выделился специфический класс процессов управления, когда элементы системы, владея частичной информацией о глобальной проблеме, должны эффективно принимать свои решения с позиций интересов функционирования общего большого объекта. Выяснилась необходимость организации работы многих агентов (программ, людей), обладающих правом на самостоятельные действия при сохранении ответственности за них. Решение такого рода задач вызвало необходимость создания теории мультиагентных систем (раздела теории распределенного искусственного интеллекта, учитывающего возможности самообучения отдельных агентов при принятии решений в рамках своей компетенции). Сложность этих задач возросла и из-за необходимости использования знаний из теорий параллельных вычислений, сетевых компьютерных технологий, защиты информации, мобильной связи и ряда других.

Под *агентом* часто понимается программа (робот или человек), способная действовать в интересах достижения целей, поставленных пользователем.

Индустрия многоагентных систем пока находится в стадии становления. Она постепенно проникает в *следующие области*:

- подготовку вариантов информации для отдельных ЛПР или коллективов, принимающих решения;
- поддержку конкурентоспособности производств и завоевание рынков сбыта;
- поиск сервисов и дефицитных продуктов для сложных производств;
- управление роботами и людьми в различных сложных технических объектах.

При полной автоматизации управления сложными объектами интеллектуальными агентами обычно являются программы и роботы. *Для мультиагентных систем сформировались некоторые общие требования к агентам*:

- автономность (функционирование без прямого вмешательства других агентов);
- взаимодействие с другими агентами на общем коммуникационном языке;
- восприятие внешней среды через различные типы интерфейсов;
- реакция на происходящие изменения в среде;
- проявление инициативы;
- наличие собственной картины окружающего мира;
- способность к интеллектуальному поведению на основе обучения и логических выводов;
- каждый агент способен обмениваться своими знаниями с другими агентами.

Кроме того, при отказе одного или даже нескольких агентов система должна продолжить функционирование. Независимые задачи могут выполняться различными агентами, цели выдвигаются пользователем системы. Замечено, что при требовании согласования решений между несколькими агентами временные затраты на функционирование системы растут примерно по экспоненциальному закону. Время работы системы и трудности ее реализации резко возрастают при требовании менять критерии принятия решений в особых ситуаях.

3.2 Методы поиска информации

Информационно-логические задачи. Одним из самых распространенных и важных применений формальных языков являются задачи автоматизированного информационного поиска. Развитие научно-технического прогресса сопровождается резким увеличением потока информации.

В США на поиск научно-технической информации тратится более миллиарда долларов в год. На дублирование разработок из-за неполной информированности тратится около 10 % всех средств, расходуемых на новые разработки. Поэтому поиск информации превратился в крупную научную проблему.

Задачи обработки информации, библиографического поиска и анализа фактографических данных часто объединяют под общим названием информационно-логических задач. Для них характерны логическая обработка больших объемов информации и ее хранение. Информация часто представляется не только в количественной, но и в качественной форме. В первую очередь такого рода задачи используются при применении различных автоматических систем обработки информации, при поиске конструкторско-технологической информации в системах автоматизированного проектирования, в оперативном управлении предприятиями.

Применение автоматических библиографических систем имеет большое значение для развития всех областей науки и техники. Число публикаций и разработок по всем отраслям знаний настолько велико, что специалисты не в состоянии следить за ними даже в своих узких областях. Из-за трудностей поиска иногда легче произвести исследования или разработки заново, чем найти их описание.

Типы систем поиска информации. Фактографические системы поиска информации представляют собой новый уровень автоматизации умственных процессов. Они должны позволить не только находить соответствующую литературу, но и выполнять логический анализ и сопоставление содержания различных статей и книг, производить обобщение сведений и т. д.

Библиографические дескрипторные системы накопления, хранения, обработки и поиска информации условно делятся на два типа:

1) дескрипторные системы без грамматики;

2) дескрипторные системы с грамматикой.

Рассмотрим сущность дескрипторного метода поиска. Содержание документа в общих чертах может быть представлено набором характерных для данного текста слов. Их называют ключевыми словами для данного текста. Для построения поисковой системы из всего многообразия ключевых слов, выбранных из ряда характерных для данной тематики текстов, составляется стандартный набор терминов без синонимов со строго фиксированными значениями. Эти термины называются *дескрипторами*, а полный набор таких терминов называется *словарем дескрипторов*. Для каждого документа составляется свой набор дескрипторов, называемый поисковым образом документа.

Массив дескрипторных наборов документов может быть построен двумя различными способами: прямым и инверсным.

При прямом способе в памяти машины последовательно записываются номера документов и за каждым из них указываются дескрипторы или их коды. Процесс поиска заключается в последовательном сравнении для каждого документа всех дескрипторов запроса с дескрипторами документа и выделении тех документов, для которых выполняется критерий соответствия.

При инверсном способе за основные позиции принимают не документы, а дескрипторы. Для каждого дескриптора записываются все номера документов, которые имеют в своих поисковых образах этот дескриптор. Поиск нужных документов осуществляется в словаре дескрипторов путем обращения к тем дескрипторам, которые имеются в запросе, и путем выбора всех номеров документов этих дескрипторов. Отобранными считаются те документы, номера которых оказались общими для всех дескрипторов, указанных в запросе.

Пример – Пусть требуется найти литературу по алгоритмическим языкам для программирования экономических задач в заданном массиве документов (таблица 3.1).

Таблица 3.1 – Инверсный массив

Наименование дескриптора	Код дескриптора	Номер документа
Вычислительная машина	101	0031, 0034, 0038, 0045
Логическая обработка	102	0012, 0026, 0031, 0046
Алгоритмический язык	105	0003, 0022, 0027, 0031
Экономика	205	0031, 0168, 1342
Оптимизация	337	0512, 1314

Наш запрос можно записать следующим образом: 101, 105, 205.

Шаги поиска:

1) 0031, 0034, 0038, 0045 (дескриптор 101);

2) 0003, 0022, 0027, 0031 (дескриптор 105);

3) 0031, 0168, 1342 (дескриптор 205);

4) 0031 – ответ.

Ответ получается как нахождение общей части для всех шагов поиска.

Иногда дескрипторы запросов делятся на категории, указывающие их важность.

В простейшем способе дескрипторного поиска дескрипторы, относящиеся к документам, и дескрипторы, входящие в состав запроса, представляют собой простые наборы, не связанные какой-либо грамматикой, в частности, порядок их расположения безразличен. Однако такой способ приводит либо к выдаче излишних документов, не относящихся к интересующему нас вопросу, либо к пропуску нужных документов.

Например, слова «банк, данные, электронный» могут встретиться в тексте о банке, ведущем финансовые операции и запирающемся на электронный замок, а также в тексте об электронном банке данных научной информации.

Более эффективными являются дескрипторные поисковые системы с грамматикой, в которых дескрипторы, относящиеся к документам, и дескрипторы, входящие в состав запросов, снабжаются дополнительными символами, указывающими на семантическую роль этих дескрипторов (объект процесса, процесс, причина).

Иногда роль дескрипторов определяется их положением в наборе или связями между отдельными дескрипторами. В этом случае дескрипторные системы приближаются по своему принципу действия к фактографическим информационным системам с грамматикой. Например, для приведенного выше запроса можно потребовать, чтобы выражение «электронный банк данных» шло в заданном порядке и неразрывно в одном предложении. Для фактографических информационных систем с грамматикой строится специальный информационный язык, позволяющий записывать фактические сведения, относящиеся к тем или иным областям знаний.

Основные части информационного языка:

1) набор базисных терминов, обозначающих основные предметы данной области знаний (например, для вычислительной техники это двоичный разряд, машинное слово и т. п.);

2) набор смысловых отношений между предметами (примеры отношений: один предмет является элементом класса, представляющего другой предмет; предмет является субъектом процесса; предмет является объектом процесса и т. п.);

3) набор формальных правил (синтаксис), позволяющих из основных терминов и отношений языка строить более сложные термины и отношения, а также осуществлять переход от информационного языка к естественному;

4) система кодирования понятий, отношений и синтаксических правил языка, позволяющей осуществлять преобразование и хранение информации, представленной на этом языке, с помощью ЭВМ.

Сложные фактографические системы обычно предназначены для выдачи фактического материала на различные вопросы определенного круга и характера.

Простейшей системой может быть такая, которая осуществляет проверку вводимых в машину утверждений и дает три ответа: «да», «нет», «неизвестно».

Специальные информационные языки для поиска информации. В качестве иллюстрации приведем пример одного из возможных специальных информационных языков для поиска конструкторско-технологической информации на основе порождающей грамматики:

$$\Gamma = <V, W, I, R>,$$

где V – множество понятий (терминологические словосочетания, характеристики объектов), составляющих основной словарь;

W – вспомогательный словарь;

I ($I \in W$) – начальный символ;

R – множество правил подстановки.

Правила подстановки представим в форме Бэкуса с использованием знаков /\ (И), \/ (ИЛИ) и ^ (НЕ):

<описание объекта на информационном языке> ::= <логическое произведение сообщений>

<логическое произведение сообщений> ::= <сообщение> | <сообщение> /\ <логическое произведение сообщений>

<сообщение> ::= <логическая сумма фраз>

<логическая сумма фраз> ::= <фраза> | <фраза> \/ <логическая сумма фраз>

<фраза> ::= <понятие> ^ <понятие> | <характеристика> | ^ <характеристика>

<характеристика> ::= <понятие> <понятие> | <понятие> <знак отношения> <значение числовой характеристики>

<знак отношения> ::= > | < | ≥ | ≤ | =

<значение числовой характеристики> ::= <положительное число> | <отрицательное число>

Язык, порождаемый грамматикой Г, будем называть информационным языком, а элементы множества *W* – сообщениями на информационном языке. Для осуществления поиска нужных сведений в этом случае сообщения на языке взаимодействия переводятся на информационный язык (кодирование) и после получения ответа на информационном языке проводится его декодирование. Сложность поиска для пользователя в этом случае сводится к умению описать требуемый объект, используя его характерные признаки в их логической взаимосвязи.

Особенность автоматизированной системы состоит в том, что она не только дает положительный или отрицательный ответ, но, учитывая предъявляемые к материалу требования, рекомендует, какой полимер выбрать, а в случае отсутствия возможности дать точный ответ, предлагает рекомендацию о материалах с близкими свойствами по отношению к заданным.

Развитие поисковых систем такого типа в будущем окажет значительное влияние на развитие научно-технического прогресса и поднятие культурного уровня специалистов в различных областях деятельности. Уже современные автоматизированные хранилища позволяют в малых объемах сосредоточить колоссальные объемы информации. Например, с помощью лазерной технологии можно записать на специальной пластинке, которая по размерам примерно равна этикетке спичечного коробка, порядка тысячи страниц текста и чертежей. Телевидение, спутниковые и другие средства связи позволяют обеспечить передачу информации на большие расстояния.

В настоящее время страны выработали международные коммуникативные форматы, позволяющие передавать информацию по каналам связи. Например, информацией на машинных носителях в области изобретательской деятельности сейчас обменивается ряд стран. Аналогичные обмены ведутся и в области реферативной информации по публикуемым источникам.

Таким образом, использование стандартных форм описания документов позволяет упростить решение такой важнейшей задачи, как международный обмен информацией во всех областях деятельности человека. Большие перспекти-

вы открываются и в области автоматического перевода с разных языков на язык страны – потребителя информации.

В технических областях знаний запись информации на практически однозначно *воспринимаемых* машиной подмножествах естественного языка позволяет уже сегодня получать машинные переводы, пригодные для понимания затрагиваемого вопроса специалистом. Поэтому информационная система, снабженная соответствующими программами для перевода с наиболее употребительных языков планеты, позволяет специалистам, минуя языковой барьер, знакомиться с последней информацией в мире в конкретных областях деятельности.

Назначение баз данных и знаний. Большинство сложных производственных автоматизированных систем различного назначения требует не только использования специфических процессов для подготовки исходных данных для решения отдельных задач, но и обмена данными с выполняемыми одновременно другими задачами, а при создании интеллектуальных систем и подбора методов для разрешения создавшихся ситуаций. Это привело к развитию систем, близких по духу к операционным, но имеющим целевое назначение – автоматизацию подготовки, обработки, хранения, защиты и актуализации данных, включая такие интеллектуальные процессы, как выбор методов и оценка путей решения текущей задачи.

Эта ситуация в своем разрешении начала свой путь с формирования понятий базы данных и знаний. Идея базы данных имеет в своей основе подход в решении ряда задач, когда исходные данные и другие промежуточные результаты хранятся отдельно от исполняемых программ и с помощью заранее описанных механизмов управления извлекаются из базы данных по стандартным схемам, используемым каждой задачей независимо от вычислительного процесса на стадии подготовки исходного материала для текущего этапа его реализации. Базу данных часто определяют как совокупность взаимосвязанной информации, организованной по определенным правилам. База данных обычно определена по схеме, не зависящей от программ, которые к ней обращаются. Чтобы легче искать данные и их обрабатывать, обычно используют системы управления базами данных (СУБД), которые ориентированы на поддержку выполнения различных запросов и ведения БД. На практике чаще других используются реляционные СУБД – когда данные отображаются в табличной форме. В каждой СУБД имеется специальный язык запросов, обеспечивающий эффективный доступ к различным элементам (или их группам) в базе данных. Для баз данных реляционного типа массовое применение получил язык запросов *SQL*.

В отличие от базы данных, в базе знаний располагаются проверенные и накопленные человечеством истины, факты, принципы и другие аналогичные объекты. Обычно их источником являются документы, книги, статьи и т. п. В базе знаний (БЗ) располагаются в соответствии с принятой системой классификации объекты познания, представляющие собой совокупность знаний. Каждый из объектов представляет набор элементов знаний. Благодаря использованию концептуальных

связей объекты объединяются, образуя базу знаний. Каждая база знаний включает в себя набор сведений, правила и механизмы логического вывода.

Такие базы знаний являются необходимым компонентом при решении задач искусственного интеллекта, в частности, при создании экспертных систем, так как они позволяют получать знания, которые непосредственно не вносятся в БД, а определяются с помощью правил вывода.

В экспертных системах знания специалистов являются источником формирования баз знаний, а правила вывода могут использоваться из универсальных систем или их стандартных оболочек.

Банк данных. На основе баз данных и знаний в различных автоматизированных системах используются банки данных. Вместе с принятой к использованию СУБД они являются его ядром. Автоматизированные системы управления, спроектированные на фундаменте концепций банков данных, позволяют обеспечить многоаспектный доступ к совокупности взаимосвязанных данных, интеграцию и централизацию управления данными, устранение их избыточности и защиту, возможность телепроцессорной обработки. По своей сути банк данных является информационной системой. Формирование и ведение банков данных связаны с большими затратами, которые окупаются при большом трафике (потоке запросов). В ряде разработок применяется реляционная (табличная) модель данных, которая поддерживается соответствующими СУБД. Опора на эту модель позволяет легче организовать процесс обмена и между различными банками данных. Популярность такого подхода основывается на использовании таблиц как структуры для описания объектов реляционной модели. Структура таблицы определяется совокупностью столбцов. В каждой строке таблицы содержится по одному значению в соответствующем столбце. Столбец соответствует некоторому элементу данных – атрибуту. Каждый столбец имеет имя соответствующего элемента данных (атрибута). Один или несколько атрибутов, значения которых однозначно идентифицируют строку таблицы, называют ключом таблицы. Он и является исходным элементом различных операций с объектами и для организации получения информации из других таблиц. Табличная форма работы с информацией широко используется в различных сферах деятельности, и поэтому такие банки данных облегчают их применение непрофессионалами.

К схемам такого рода относятся и банки данных, позволяющие осуществлять в текстовых документах поиск необходимой информации на основе логических конструкций, построенных пользователем с применением ключевых слов.

Особую роль интеллектуальной части СУБД играют функции интеграции текстовых, числовых и графических данных и в последние годы и речевых сообщений, особенно в диалоговых процедурах. К элементам интеллектуализации можно также отнести и протоколы обмена документами при создании различных систем, когда при установлении соединения абонентов проверяются их права на доступ к ресурсам и выполнение контроля защиты баз данных от разрушения, для регистрации пользователей на основе средств искусственного интеллекта.

При автоматизации проектирования и принятия решений в различных областях деятельности полезно в БД иметь особый подраздел в виде архива гото-

вых прототипов (проектов решений). Однако это связано с рядом особенностей введения этого подраздела, так как необходим экспертный отбор элементов архива, их заверения электронной цифровой подписью с выполнением аналогичных условий и при изъятии элементов архива.

Специфика использования знаний в интеллектуальных системах. При обработке на ЭВМ данные постоянно трансформируются, последовательно проходя от входной информации как результата измерений и наблюдений к данным на материальных носителях информации в виде таблиц, протоколов, справочников; структур данных в виде диаграмм, графиков, функций; в компьютере на языках описания данных и т. д. Более высокой формой организации данных является их представление в виде баз знаний.

Знания связаны с данными, основываются на них, но представляют собой результат мыслительной деятельности человека, обобщают его опыт, полученный в ходе практической деятельности. Знания – это выявленные закономерности предметной области. При обработке на ЭВМ знания также трансформируются: от существующих форм в памяти человека как результат обучения, воспитания, мышления к знаниям, помещенным на материальных носителях – учебниках, инструкциях, методических пособиях, книгах и далее к знаниям, описанным на языках их представления для занесения в компьютерные базы знаний.

Знания могут быть классифицированы на *первичные* (о видимых взаимосвязях между отдельными событиями и фактами в предметной области) и *глубинные* (абстракции, аналогии, схемы, отображающие структуру и процессы в предметной области).

Часто знания разделяют на *процедурные* и *декларативные.*

Исторически первыми были процедурные знания, т. е. знания, представленные в алгоритмах.

Рассмотрим, например, фрагмент программы на алгоритмическом языке Паскаль:

Pi := 3.14;
R := 20;
S := *Pi* * *R* * *R*;
WRITELN ('Площадь круга *S* = ', *S*).

Первые два оператора представляют собой данные, третий оператор – знание. Оно является результатом интеллектуальной деятельности древних геометров и представляет собой закон, выражающий площадь круга через его радиус.

Существуют десятки способов представления декларативных знаний для различных предметных областей. Большинство из них может получаться на основе следующих классов моделей сетей:

- продукционных;
- фреймовых;
- семантических.

Продукционная модель состоит из трех основных компонентов:

- первый из них – это база правил типа ЕСЛИ (условие), ТО (действие);

- вторым компонентом является рабочая память, в которой хранятся исходные данные к задаче и выводы, полученные в ходе работы системы;

- третий компонент – механизм логического вывода, использующий правила в соответствии с содержимым рабочей памяти.

Пример – В базе правил экспертной системы имеются два правила.

Правило 1: ЕСЛИ «намерение – отдых» и «дорога ухабистая», ТО «использовать джип».

Правило 2: ЕСЛИ «место отдыха – горы», ТО «дорога ухабистая».

Допустим, что в рабочую память поступили исходные данные: «намерение – отдых», «место отдыха – горы».

Механизм вывода начинает сопоставлять образцы из условных частей правил с образцами, хранимыми в рабочей памяти. Если образцы из условной части имеются в рабочей памяти, то условная часть считается истинной, в противном случае – ложной.

В данном примере при рассмотрении правила 1 оказывается, что образец «намерение – отдых» находится в рабочей памяти, а образец «дорога ухабистая» отсутствует, поэтому условная часть правила 1 считается ложной. При рассмотрении правила 2 выясняется, что его условная часть истинна. Механизм вывода выполняет заключительную часть этого правила, и образец «дорога ухабистая» заносится в рабочую память. Правило 2 при этом выбывает из числа кандидатов на рассмотрение.

Снова рассматривается правило 1, условная часть которого теперь становится истинной, и содержимое рабочей памяти пополняется образцом «использовать джип». В итоге правил, которые можно было бы применять, не остается, и система останавливается.

В рассмотренном примере приведен прямой вывод – процесс от данных к поиску цели. Однако применяют и обратный вывод – процесс от цели для ее подтверждения к данным. Продемонстрируем этот способ на нашем примере. Допустим, что наряду с исходными данными «намерения – отдых» и «место отдыха – горы» имеется цель «использовать джип».

Согласно правилу 1 для достижения этой цели требуется выполнение условия «дорога ухабистая», поэтому условие становится новой целью. При рассмотрении правила 2 оказывается, что условная часть этого правила в данный момент истинна, поэтому рабочая память пополняется образцом «дорога ухабистая». При повторном рассмотрении правила 1 подтверждается цель «использовать джип».

При обратном выводе система останавливается в двух случаях: либо достигается первоначальная цель, либо кончаются правила. При прямом выводе система останавливается только тогда, когда кончаются правила, либо при появлении в рабочей памяти специально предусмотренного образца, например, «использовать джип».

В приведенном примере на каждом этапе прямого вывода можно было использовать только одно правило. В общем же случае на каждом этапе вывода таких правил несколько, и тут возникает проблема выбора. Например, введем в рассмотрение еще одно правило.

Правило 3: ЕСЛИ «намерение – отдых», ТО «нужна скорость».

Кроме того, введем условие останова системы – появление в рабочей памяти образца «использовать джип».

Теперь на первом этапе прямого вывода появляется возможность применять либо правило 2, либо правило 3. Если сначала применить правило 2, то на следующем этапе можно будет применять правило 1 и правило 3. Если на этом этапе применить правило 1, то выполнится условие останова системы, но если прежде применить правило 3, то потребуется еще один этап вывода. Этот пример показывает, что выбор применяемого правила оказывает прямое влияние на эффективность вывода. В реальной системе, где имеется множество правил, появляется проблема их оптимального выбора.

Если на каждом этапе логического вывода существует множество применимых правил, то это множество носит название *конфликтного набора,* а выбор одного из них называется *разрешением конфликта.*

Аналогичная ситуация возникает и при обратном выводе. Например, дополним предыдущий пример еще одним правилом.

Правило 4: ЕСЛИ «место отдыха – пляж», ТО «дорога ухабистая».

Если на основании этого условия подтверждается цель «использовать джип», то для достижения первоначальной цели достаточно применить только одно правило 1. Однако, чтобы подтвердить новую цель «дорога ухабистая», открывается возможность применения правила 1, нужно использовать либо правило 2, либо правило 4. Если сначала применить правило 2, то это будет самый удачный выбор, поскольку сразу же можно применить и правило 1. С другой стороны, если попытаться применить правило 2, то, поскольку образца «место отдыха – пляж», который является условием правила 4, в рабочей памяти не существует и, кроме того, не существует правила, подтверждающего его, данный выбор является неудачным. И лишь со второго захода, применяя правило 2, можно подтвердить цель «дорога ухабистая».

Следует обратить внимание на то, что при обратном выводе правило 3, которое не оказывает прямого влияния на достижение цели, не принималось в расчет с самого начала. Таким образом, для обратных выводов характерна тенденция исключения из рассмотрения правил, не имеющих прямого отношения к заданной цели, что позволяет повысить эффективность вывода.

Модели знаний. Продукционная модель – это наиболее часто используемый способ представления знаний в современных экспертных системах. Основными преимуществами продукционной модели являются наглядность, высокая модульность, легкость внесения изменений и дополнений, простота механизма логического вывода.

Следующей эффективной моделью представления знаний является фреймовая модель. Фреймовая модель использовалась первоначально в психологии и философии для описания понятия абстрактного образа. Например, слово «автомобиль» вызывает у слушающих образ устройства, способного перемещаться, имеющего четыре колеса, салон для шофера и пассажиров, двигатель, руль.

Приведенное описание абстрактного образа «автомобиль» является минимальным и из него ничего нельзя убрать без потери его сущности.

Фрейм – это модель абстрактного образа, минимально возможное описание сущности какого-либо объекта, явления, события, ситуации, процесса. Фрейм состоит из имени и отдельных единиц, называемых *слотами*. Он имеет однородную структуру:

ИМЯ ФРЕЙМА
Имя 1-го слота : значение 1-го слота;
Имя 2-го слота : значение 2-го слота;
...
Имя *N*-го слота : значение *N*-го слота.

В качестве значения слота может выступать имя другого фрейма. Таким образом фреймы объединяются в сеть. Свойства фреймов наследуются сверху вниз, т. е. от вышестоящих к нижестоящим через так называемые *AKO*-связи (начальные буквы слов в английской фразе «*A Kind Of*», что можно перевести как «это» или «разновидность»). Слот с именем *AKO* указывает на имя фрейма более высокого уровня иерархии.

Например, на рисунке 3.2 фрейм «Студент» имеет ссылки на вышестоящие фреймы: «Человек» и «Млекопитающее». Поэтому на вопрос: «Может ли студент мыслить?» – ответ будет положительным, так как этим свойством обладает вышестоящий фрейм «Человек».

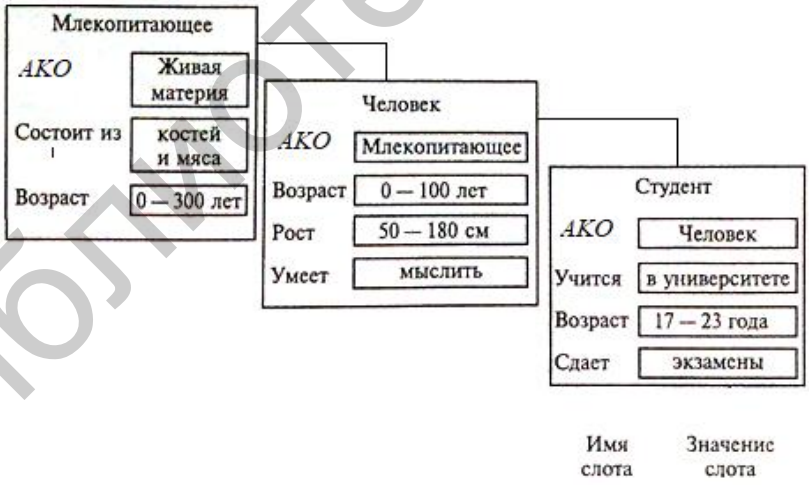


Рисунок 3.2 – Сеть фреймов

Если одно и то же свойство указывается в нескольких, связанных между собой, фреймах, то приоритет отдается нижестоящему фрейму. Так, возраст фрейма «Студент» не наследуется из вышестоящих фреймов.

Основным преимуществом фреймов как способа представления знаний являются наглядность и гибкость в употреблении. Кроме того, фреймовая

структура согласуется с современными представлениями о хранении информации в памяти человека.

В основе семантических сетей для представления знаний лежит идея о том, что любые знания можно представить в виде совокупности *понятий* (объектов) и *отношений* (связей). Семантическая сеть представляет собой ориентированный граф, вершинами которого являются понятия, а дугами – отношения между ними. Сам термин «семантическая» означает смысловая.

Пример семантической сети приведен на рисунке 3.3.

Рисунок 3.3 – Семантическая сеть

Эта модель также позволяет наглядно описать знания в соответствии с современным представлением об организации долговременной памяти человека. Недостаток – сложность поиска вывода, а также затруднения корректировки, т. е. удаления и дополнения сети новыми знаниями.

3.3 Диалоговые методы решения задач и экспертные системы

Решение проблемы создания искусственного интеллекта в различных областях деятельности человека, как отмечалось выше, еще долгие годы будет идти по пути вскрытия универсальных механизмов принятия решений, направленных на автоматическое сужение области поиска результатов в соответствии с конкретной исходной ситуацией. Тем не менее, человеку приходится на практике решать целый ряд задач, которые требуют использования вычислительных ресурсов. В этом случае становится логичным использование самого человека в качестве звена автоматизированной системы, что порождает проблему обеспечения эффективного общения человека с ЭВМ в диалоговом режиме.

Средства общения могут развиваться в следующих основных направлениях:

- создание общего ПО для манипулирования с различными объектами в памяти ЭВМ на подмножествах естественного языка, опираясь на внешние устройства ЭВМ (дисплей, клавиатуру, звуковой ввод информации и т. п.);

- специальное ПО, рассчитанное на использование средств естественного языка и графику для выполнения некоторой части процесса решения конкретной задачи человеком.

Оба направления существенно опираются на программное и техническое обеспечение диалога на уровне простейших операций, но построение программ в области автоматизации сложных процессов немыслимо на базе только элементарных действий, так как решение задачи сводится к большим затратам времени вплоть до утраты разумных границ временных ограничений на получение ответа.

Диалоговый режим открывает новые возможности и в осуществлении моделирования (вычислительного эксперимента). Некоторый объект можно представить в виде ряда элементов, формальное описание которых позволяет имитировать их поведение на ЭВМ. В результате имитации поведения среды можно проверить качество будущей конструкции без ее физической реализации и тем самым экономить средства, затрачиваемые на неудачные экспериментальные образцы. Наличие моделей для различных элементов позволяет инженеру в диалоговом режиме путем подбора построить модель объекта из нужного набора элементов, обладающую искомыми свойствами.

Одновременное использование при решении сложных задач ряда специалистов в различных областях знаний позволит применить системный подход к решению проблемных вопросов. Сети ЭВМ в таких случаях могут стать тем звеном, которое в диалоговом режиме поможет группе специалистов, находящихся в разных местах, прийти к эффективному решению.

Поэтому проблему общения с ЭВМ на подмножествах естественного языка пришлось выделить как самостоятельную.

Знания, которыми обладает специалист в какой-либо области, можно разделить на формализуемые и плохо формализуемые. *Формализуемые знания* излагаются в книгах и руководствах в виде законов, формул, моделей, алгоритмов. Формализуемые знания характерны для точных наук, таких как математика, физика, химия, астрономия. Науки, которые принято называть описательными, обычно оперируют с *плохо формализуемыми знаниями*. К таким наукам можно отнести, например, зоологию, ботанику, экологию, социологию, педагогику, медицину и др.

Существуют неформализуемые знания, которые вообще не попадают в книги и руководства в связи с их неконкретностью, субъективностью, приблизительностью. Знания этого рода являются результатом многолетних наблюдений, опыта работы, интуиции. Они обычно представляют собой множество эмпирических и эвристических приемов и правил. Такие знания передаются из поколения в поколение в виде определенных навыков, ноу-хау, секретов ремесла. Есть также знания, которые не могут быть выражены ни в математическом виде, ни в терминах обычного человеческого языка.

Класс задач, относящихся к неформализуемым и плохо формализуемым знаниям, значительно больше класса задач, для которых знания формализуемы. Этим объясняется особая популярность и широкое практическое применение экспертных систем, которые открыли возможность применения компьютерных технологий в предметных областях, в которых знания плохо формализуемы.

Экспертные системы – это сложные программные комплексы, аккумулирующие знания специалистов в конкретных предметных областях и тиражирующие эти знания для консультаций менее квалифицированных пользователей.

При решении различных задач иногда приходится обращаться к другим специалистам или системам, иначе говоря, привлекать экспертов. В последнее время интенсивно ведется разработка таких программ, которые обеспечивают экспертизу в различных областях знаний: управлении производством, медицине и т. д.

Экспертная система должна содержать существенную часть знаний эксперта-человека в конкретной области и использовать накопленную информацию подобно человеку. Например, экспертная система в области медицины должна уметь ставить диагноз на основе анализа симптомов пациента. Из такого класса программ наиболее интересна в прикладном плане система МИЦИН, созданная в Стэнфордском университете в США, и ее последующая версия ТЕЙРЕСИАС. Их успех у практикующих медиков в том, что они помогают врачу поставить правильный диагноз на основании рассуждений типа: «ЕСЛИ…, ТО можно предположить, что…». Программа дает врачу численную оценку вероятности каждого из заключений. За словами «можно предположить», например, стоит вероятностная оценка 0,1, а за словом «вероятно» – оценка 0,8 и т. д. Программа также указывает, какие дополнительные наблюдения и тесты могут снизить неопределенность ответа.

Главная особенность этих программ в том, что они способны на языке, близком к естественному, объяснить, как было выработано решение. Программа это делает на основе записи шагов, по которым можно объяснить, почему та или иная возможность была исключена из рассмотрения.

Способность программы давать объяснения сыграла существенную роль в признании системы практикующими медиками. Конечная ответственность за принятый диагноз, конечно, лежит на враче. Естественно, как упоминалось выше, врач, работающий в контакте с вычислительной машиной, будет иметь преимущество, так как в худшем случае он примет свое решение, а в лучшем воспользуется диагнозом машины или примет дополнительное решение изучить глубже на основе точных анализов сложившуюся ситуацию.

Рассмотрим одну из возможных структур экспертной системы.

Обобщенная блок-схема экспертной системы представлена на рисунке 3.4.

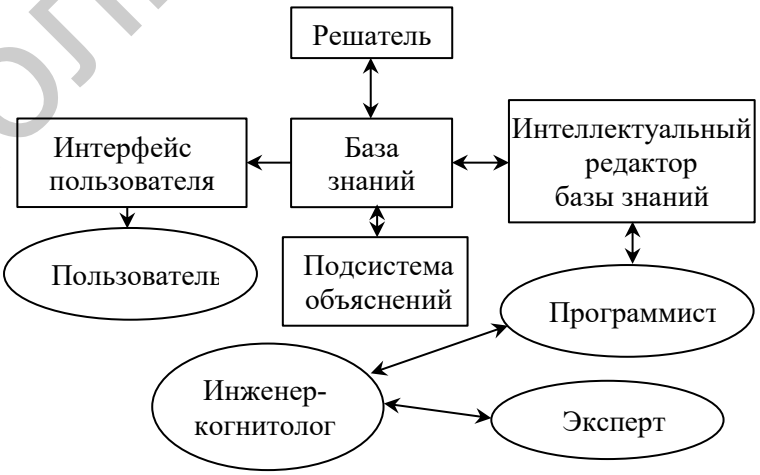


Рисунок 3.4 – Типичная блок-схема экспертной системы

Обычно в ее состав входят следующие взаимосвязанные между собой модули:

- база знаний – ядро экспертной системы, совокупность знаний предметной области, записанная на машинном носителе в форме, понятной эксперту и пользователю;

- интеллектуальный редактор базы знаний – программа, представляющая инженеру-когнитологу и программисту возможность создавать базу знаний в диалоговом режиме (она включает в себя системы вложенных меню, шаблонов языка представления знаний, подсказок (*help*-режим) и других сервисных средств, облегчающих работу с базой знаний);

- интерфейс пользователя – комплекс программ, реализующих диалог пользователя с экспертной системой на стадии как ввода информации, так и получения результатов;

- решатель (дедуктивная машина, блок логического вывода) – программа, моделирующая ход рассуждений эксперта на основании знаний, имеющихся в базе знаний;

- подсистема объяснений – программа, позволяющая пользователю получать ответы на вопросы: «Как была получена та или иная рекомендация?» и «Почему система приняла такое решение?». Ответ на вопрос «Как?» – это трассировка всего процесса получения решения с указанием исполняющих фрагментов базы знаний, т. е. всех шагов цепи умозаключений. Ответ на вопрос «Почему?» – ссылка на умозаключение, непосредственно предшествовавшее полученному решению, т. е. отход на один шаг назад.

В коллектив разработчиков экспертной системы входят как минимум четыре специалиста (или четыре группы специалистов):

- эксперт;
- инженер-когнитолог;
- программист;
- пользователь.

Возглавляет коллектив инженер-когнитолог – ключевая фигура при разработке систем, основанных на знаниях. Обычно это руководитель проекта, в задачу которого входит организация всего процесса создания экспертной системы. С одной стороны, он должен быть специалистом в области искусственного интеллекта, а с другой – разбираться в предметной области, общаться с экспертом, извлекая и формализуя его знания, передавать их программисту, кодирующему и помещающему их в базу знаний экспертной системы.

Экспертная система работает в двух режимах – приобретения знаний и решения задач или консультаций.

В режиме приобретения знаний происходит формирование базы знаний (выполняют прораммист совместно с экспертом и инженером-когнитологом).

В режиме решения задач или консультаций общение с экспертной системой осуществляет конечный пользователь.

Обычно знания, которыми располагает эксперт, различаются степенью надежности, важности, четкости. В этом случае они снабжаются некоторыми

весовыми коэффициентами, которые называют коэффициентами доверия. Такие знания обрабатываются с помощью алгоритмов нечеткой математики.

В процессе опытной эксплуатации коэффициенты доверия могут подвергаться корректировке. В этом случае говорят, что происходит обучение экспертной системы. Процесс обучения экспертной системы может производиться автоматически с помощью обучающего алгоритма либо путем вмешательства инженера-когнитолога, выполняющего роль учителя.

В процессе разработки экспертные системы проходят определенные стадии, в результате которых создаются различные версии, называемые *прототипами*.

Демонстрационный прототип – экспертная система, которая решает часть требуемых задач, демонстрируя жизнеспособность метода инженерии знаний. Работает, имея в базе знаний всего 50–100 правил.

Исследовательский прототип – экспертная система, которая решает все требуемые задачи, но неустойчива в работе и не полностью проверена. База знаний содержит 200–500 правил.

Действующий прототип надежно решает все задачи, но для решения сложных задач может потребоваться много времени и памяти. База знаний содержит 500–1000 правил.

Промышленная экспертная система обеспечивает высокое качество решения всех задач при минимуме времени и памяти, что достигается переписыванием программ с использованием более совершенных инструментальных средств и языков низкого уровня. База знаний содержит 1000–1500 правил.

Коммерческая экспертная система отличается от промышленной тем, что помимо собственного использования она может продаваться различным потребителям. База знаний содержит 1500–3000 правил.

В настоящее время уже сложилась определенная технология разработки экспертных систем (рисунок 3.5):

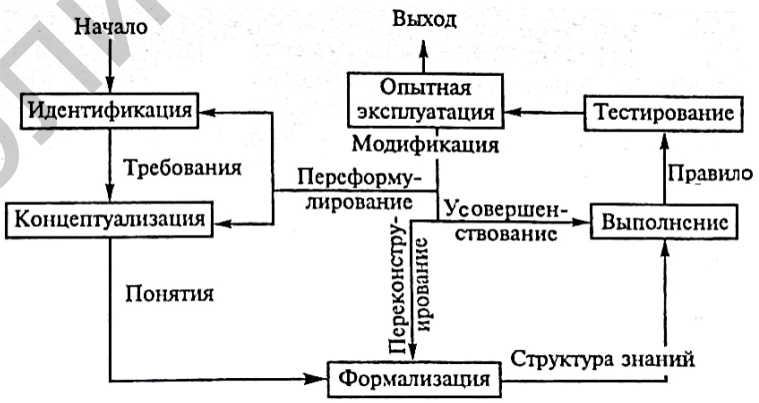


Рисунок 3.5 – Технология разработки экспертной системы

1 *Идентификация* (*постановка задачи*). На этапе устанавливаются задачи, которые подлежат решению, выявляются цели разработки, требования к экспертной системе, ресурсы, используемые понятия и их взаимосвязи, определяются методы решения задач. Цель этапа – сформулировать задачу, охаракте-

ризовать поддерживающую ее базу знаний и таким образом обеспечить начальный импульс для развития базы знаний.

2 *Концептуализация.* Проводится содержательный анализ проблемной области, выявляются используемые понятия и их взаимосвязи, определяются методы решения задач.

3 *Формализация.* Определяются способы представления всех видов знаний, формализуются основные понятия, определяются способы интерпретации знаний, оценивается адекватность целям системы зафиксированных понятий, методов решения, средств представления и манипулирования знаниями.

4 *Выполнение.* Осуществляется наполнение экспертом базы знаний. Процесс приобретения знаний разделяют на извлечение знаний из эксперта, организацию знаний, обеспечивающую эффективную работу системы, и представление знаний в виде, понятном экспертной системе. Из-за эвристического характера знаний их приобретение является весьма трудоемким.

5 *Тестирование.* Эксперт и инженер по знаниям в интерактивном режиме, используя диалоговые и объяснительные средства, проверяют компетентность экспертной системы. Процесс тестирования продолжается до тех пор, пока эксперт не решит, что система достигла требуемого уровня компетентности.

6 *Опытная эксплуатация.* Проверяется пригодность экспертной системы для конечных пользователей. По результатам этого этапа может потребоваться модификация экспертной системы.

7 *Модификация.* В ходе создания экспертной системы почти постоянно производится ее модификация: переформулирование понятий и требований, переконструирование представления знаний и усовершенствование прототипа.

Усовершенствование прототипа осуществляется в процессе циклического прохождения через этапы выполнения и тестирования для отладки правил и процедур вывода.

Переконструирование выбранного ранее способа представления знаний предполагает возврат с этапа тестирования на этап формализации.

Если возникшие проблемы еще более серьезны, то после неудачи на этапе тестирования может потребоваться возврат на этап концептуализации и идентификации. В этом случае речь идет о переформулировании понятий, используемых в системе, т. е. перепроектировании системы заново.

3.4 Распознавание образов и обработка изображений

Общая характеристика задач распознавания образов. *Под образом* понимается структурированное описание изучаемого объекта или явления, представленное вектором признаков, каждый элемент которого представляет числовое значение одного из признаков, характеризующих соответствующий объект.

Суть задачи распознавания образов – установить, обладают ли изучаемые объекты фиксированным конечным набором признаков, позволяющим отнести их к определенному классу.

Задачи распознавания образов имеют следующие характерные черты:

1) решение, как правило, состоит из двух этапов:

- приведение исходных данных к виду, удобному для распознавания;
- собственно распознавание (указание принадлежности объекта определенному классу);

2) можно вводить понятие аналогии или подобия объектов и формулировать понятие близости объектов в качестве основания для зачисления объектов в один и тот же класс или разные классы;

3) можно оперировать набором прецедентов-примеров, классификация которых известна и которые в виде формализованных описаний могут быть предъявлены алгоритму распознавания для настройки на задачу в процессе обучения;

4) трудно строить формальные теории и применять классические математические методы (часто недоступна информация для точной математической модели или выигрыш от использования модели и математических методов несоизмерим с затратами);

5) в этих задачах возможна «плохая» информация (информация с пропусками, разнородная, косвенная, нечеткая, неоднозначная, вероятностная).

Целесообразно выделить следующие *типы задач распознавания*:

1) задача идентификации – отнесение предъявленного объекта по его описанию к заданному классу объектов;

2) задача классификации – разбиение множества объектов (ситуаций) по их описаниям на систему непересекающихся классов;

3) задача выбора информативного набора признаков при распознавании;

4) задача приведения исходных данных к виду, удобному для распознавания;

5) динамическое распознавание и динамическая классификация – задачи типов 1 и 2 для динамических объектов;

6) задача прогнозирования – это задачи типа 5, в которых решение должно относиться к некоторому моменту в будущем.

3.5 Нейронные сети как инструмент решения сложных задач

Формальный нейрон. Основным элементом нейронной сети (НС) является формальный нейрон, который осуществляет операцию нелинейного преобразования суммы произведений входных сигналов на весовые коэффициенты:

$$y = F\left(\sum_{i=1}^{n} \omega_i x_i\right) = F(W, X),$$

где $X = (x_1, \ldots, x_n)^T$ – вектор входного сигнала;

$W = (\omega_1, \ldots, \omega_n)$ – весовой вектор;

F – оператор нелинейного преобразования.

Схема нейронного элемента изображена на рисунке 3.6 и состоит из сумматора и блока нелинейного преобразования F. Каждому i-му входу нейрона

соответствует весовой коэффициент ω_i (синапс), который характеризует силу синаптической связи по аналогии с биологическим нейроном.

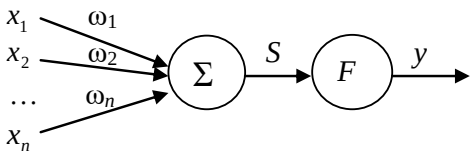


Рисунок 3.6 – Нейронный элемент

Сумма произведений входных сигналов на весовые коэффициенты называется взвешенной суммой. Она представляет собой скалярное произведение вектора весов на входной вектор:

$$S = \sum_{i=1}^{n} \omega_i x_i = (W, X) = |W| \cdot |X| \cdot \cos\alpha,$$

где $|W|$ – длина вектора W;

$|X|$ – длина вектора X;

α – угол между векторами W и X.

Длины весового и входного векторов определяются через их координаты:

$$|W| = \sqrt{\omega_1^2 + \omega_2^2 + \cdots + \omega_n^2},$$

$$|X| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}.$$

Так как для нейронного элемента длина весового вектора после обучения $|W| = \text{const}$, то величина взвешенной суммы определяется проекцией входного на весовой вектор:

$$S = |W| \cdot |X| \cdot \cos\alpha = |W| \cdot X_W,$$

где X_W – проекция вектора X на вектор W.

Если входные векторы являются нормированными, т. е. $|W| = \text{const}$, то величина взвешенной суммы будет зависеть только от угла между векторами X и W. Тогда при различных входных сигналах взвешенная сумма будет изменяться по косинусоидальному закону. Максимального значения она будет достигать при коллинеарности входного и весового векторов.

Если сила связи w_i является отрицательной, то такая связь называется тормозящей. В противном случае синаптическая связь является усиливающей.

Оператор нелинейного преобразования называется функцией активации нейронного элемента. Вектор входного сигнала называется паттерном входной активности нейронной сети, а вектор выходного сигнала – паттерном выходной активности.

Рассмотренная здесь модель формального нейрона лишь отдаленно напоминает биологический нейрон. Она используется для построения искусственных нейронных сетей, которые являются базовыми элементами нейрокомпьютеров. Для работы нейрокомпьютера выполняется обучение сети, суть которого заключается в подаче набора образов с известной теоретически реакцией сети. Цель ее настройки – решение избранного класса задач. Настройка сети обычно сводится к изменениям весовых коэффициентов w_{ij} при многократной подаче образов до тех пор, пока не будет гарантирован предельный уровень ошибки в работе сети.

Однослойный персептрон. Однослойный персептрон принято изображать в виде двухслойной нейронной сети, где первый слой нейронных элементов является распределительным, а второй – обрабатывающим. Распределительный слой передает входные сигналы на обрабатывающий слой нейронных элементов, который преобразует входную информацию в соответствии с синаптическими связями и функцией активации (рисунок 3.7). При этом каждый нейрон распределительного слоя имеет синаптические связи со всеми нейронами обрабатывающего слоя.

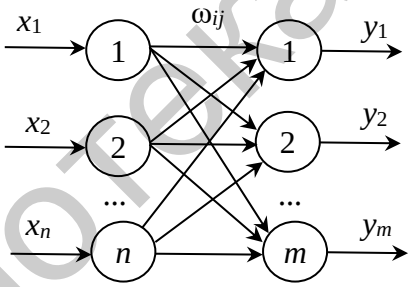


Рисунок 3.7 – Однослойный персептрон

Выходное значение j-го нейронного элемента второго слоя можно представить как

$$y_j = F(S_j) = F\left(\sum_{i=1}^{n} \omega_{ij} x_i - T_j\right),$$

где T_j – порог j-го нейронного элемента выходного слоя;

ω_{ij} – сила синаптической связи между i-м нейроном распределительного слоя и j-м нейроном обрабатывающего слоя;

F – оператор нелинейного преобразования или функция активации нейронных элементов.

Рассмотрим правило обучения однослойного персептрона, который имеет линейную функцию активации. Такое правило обучения называется дельта-

правилом и может использоваться как для обучения линейного персептрона, так и для персептрона с пороговой функцией активации.

Выходное значение линейного однослойного персептрона определяется, как

$$y_j = \sum_{i=1}^{n} \omega_{ij} x_i - T_j \ .$$

Дельта-правило предполагает минимизацию суммарной среднеквадратичной ошибки нейронной сети, которая для L входных образов определяется следующим образом:

$$E_s = \sum_{k=1}^{L} E(k) = \frac{1}{2} \sum_{k=1}^{L} \sum_{j=1}^{m} (y_j{}^k - e_j{}^k)^2,$$

где $E(k)$ – среднеквадратичная ошибка сети для k-го образа;
y_i^k – выходное значение нейронной сети для k-го образа;
e_i^k – эталонное значение нейронной сети для k-го образа.

Для минимизации суммарной квадратичной ошибки используется метод градиентного спуска.

Многослойный персептрон. В 1986 году рядом авторов (Д. Румельхард, Д. Хинтон, Р. Уильямс) независимо друг от друга был предложен *алгоритм обратного распространения ошибки* (*backpropagation algorithm*), который стал эффективным средством обучения многослойных нейронных сетей. Персептрон с одним скрытым слоем изображен на рисунке 3.8.

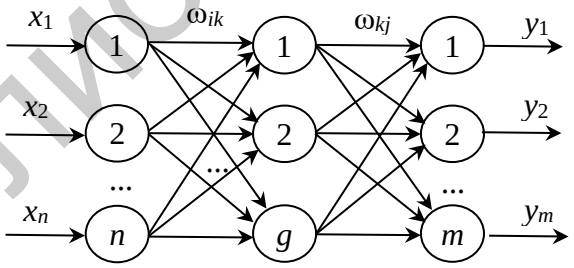


Рисунок 3.8 – Персептрон с одним скрытым слоем

Выходное значение j-го нейрона последнего слоя определяется как

$$y_j = F(S_j),$$

$$S_j = \sum_{k=1}^{g} \omega_{kj} y_k - T_j,$$

где F – оператор нелинейного преобразования или функция активации нейронных элементов;

S_j – взвешенная сумма j-го нейрона выходного слоя;

y_k – выходное значение k-го нейрона скрытого слоя;

T_j – порог j-го нейронного элемента выходного слоя;

ω_{kj} – сила синаптической связи между k-м нейроном скрытого слоя и j-м нейроном выходного слоя.

Аналогичным образом для скрытого слоя выходное значение k-го нейрона определяется как

$$y_k = F(S_k),$$

$$S_k = \sum_{i=1}^{n} \omega_{ik} x_i - T_k,$$

где S_k – взвешенная сумма k-го нейрона,

ω_{ik} – весовой коэффициент от i-го к k-му нейрону;

T_k – порог k-го нейрона скрытого слоя.

Нейронные сети глубокого доверия. В общем случае нейронная сеть глубокого доверия (*deep belief neural networks*) представляет собой многослойный персептрон. До 2006 года в научной среде была приоритетной парадигма, что многослойный персептрон с одним, максимум двумя скрытыми слоями является более эффективным для нелинейного преобразования входного пространства образов в выходное по сравнению с персептроном с большим количеством скрытых слоев. Считалось, что персептрон с более чем двумя скрытыми слоями не имеет смысла применять. Данная парадигма базировалась на теореме, что персептрон с одним скрытым слоем является универсальным аппроксиматором. Другой аспект этой проблемы заключается в том, что все попытки использовать алгоритм обратного распространения ошибки для обучения персептрона с тремя и более скрытыми слоями не приводили к улучшению решения различных задач. Это связано с тем, что алгоритм обратного распространения ошибки является неэффективным для обучения персептронов с тремя и более слоями. В 2006 году Д. Хинтон предложил «жадный» алгоритм послойного обучения (*greedy layer-wise algorithm*), который стал эффективным средством обучения нейронных сетей глубокого доверия, которые как уже отмечалось, представляют собой персептрон с большим количеством слоев. Было показано, что нейронная сеть глубокого доверия имеет большую эффективность нелинейного преобразования и представления данных по сравнению с традиционным персептроном. Такая сеть осуществляет глубокое иерархическое преобразование входного пространства образов. В результате первый скрытый слой выделяет низкоуровневое пространство признаков входных данных, второй слой детектирует пространство признаков более высокого уровня абстракции (иерархии) и т. д.

Для нейронной сети глубокого доверия, имеющей $K + 1$ слой, выходное значение j-го нейрона k-го слоя, $k = 1, 2, ..., K$, определяется следующим образом:

$$y_j^k = F(S_j^k),$$

$$S_j^k = \sum_{i=1}^{n} w_{ij}^k y_j^k + T_j^k,$$

где F – функция активации нейронного элемента;
S_j^k – взвешенная сумма j-го нейрона k-го слоя;
w_{ij}^k – весовой коэффициент между i-м нейроном $(k – 1)$-го слоя и j-м нейроном k-го слоя;
T_j^k – пороговое значение j-го нейрона k-го слоя.

В матричном виде выходной вектор k-го слоя

$$Y^k = F(S^k) = F(W^k Y^{k-1} + T^k),$$

где W – матрица весовых коэффициентов;
Y^{k-1} – выходной вектор $(k–1)$-го слоя;
T^k – вектор пороговых значений нейронов k-го слоя.

Для нулевого (распределительного) слоя

$$y_i^0 = x_i.$$

Если нейронная сеть глубокого доверия используется для классификации образов, то выходные значения сети часто определяются на основе функции активации *soft*max:

$$y_j = soft\ \max\ (S_j) = \frac{e^{S_j}}{\sum_{l=1}^{m} e^{S_l}},$$

где S_j – взвешенная сумма j-го нейрона выходного слоя;
S_l – взвешенная сумма l-го нейрона выходного слоя;
m – число нейронов выходного слоя.

Процесс обучения нейронных сетей глубокого доверия в общем случае состоит из двух этапов:

1) послойное предварительное обучение нейронной сети, начиная с первого слоя (*pre-training*, предобучение – осуществляется без учителя);

2) настройка синаптических связей всей сети (*fine-tuning*) при помощи алгоритма обратного распространения ошибки или алгоритма «бодрствования и сна» (*wake-sleep algorithm*).

Теоретически число слоев может быть произвольным, однако фактически оно ограничено ресурсами компьютера или специализированной микросхемы, на которых обычно реализуется нейронная сеть (НС). Чем сложнее НС, тем масштабнее задачи, подвластные ей.

Задачи. Первые успехи нейросетевого подхода связаны с решением плохо формализованных задач. Кроме упомянутых выше задач прогнозирования и распознавания, следует назвать следующие задачи:

- *кластеризация данных* – сводится к группировке входных данных по присущей им «близости». НС кластеризует данные на заранее неизвестное число кластеров, особенно при сжатии данных;

- *аппроксимация функций*, когда по набору экспериментальных данных $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, представляющему значения y_i неизвестной функции от аргумента x_i, требуется найти функцию, аппроксимирующую неизвестную и удовлетворяющую некоторым критериям.

- *оптимизация* – найти решение, удовлетворяющее ряду ограничений и оптимизирующее значение целевой функции (например, задача коммивояжера).

Нейронные компоненты в сетях. Причины развития работ по введению интеллектуальных компонентов в высокопроизводительные и обычные вычислительные сети – это в первую очередь обеспечение гибкости новых структур в динамике.

Ставка на параллелизм вычислений за счет использования жестко связанных компонентов сети (постоянное соединение) и ориентация на узкие классы задач исчерпали себя при переходе к решению более широких классов задач, для которых характерно:

- возникновение проблемы разработки методов алгоритмизации задач с учетом особенностей системы и организации вычислений с распараллеливанием;

- трудоемкое создание трансляторов и обучения кадров программистов;

- проблемы переносимости старого и вновь созданного программного обеспечения на другие машины;

- появились трудности в организации соединений между отдельными узлами сети с учетом конкретной задачи.

Поэтому взоры исследователей обратились к идее использования в системе элементов искусственного интеллекта. Привлекательной стороной явились два основных момента: возможность адаптации к решению нужного класса задач методами искусственного интеллекта без больших усилий человека и возможность без разработки алгоритма и программы после настройки системы получить алгоритм и программу в скрытой форме в виде соответствующей нейросети для реализации на нейрокомпьютере.

Нейрокомпьютером (НК) называют ЭВМ (аналоговую или цифровую), основной операционный блок (центральный процессор) которой построен на основе нейронной сети и реализует нейросетевые алгоритмы. НК качественно отличается от действующих классических систем параллельного типа тем, что

для решения задач используются не заранее разработанные алгоритмы, а специальная нейронная сеть, обучение которой проводится для решения избранного класса задач на специально подобранных примерах.

В качестве важных направлений развития НК выделяют следующие:

1) решение традиционных задач искусственного интеллекта: распознавание образов, классификация (извлечение знаний из данных и т. п.). На этой основе создаются модели искусственных органов человека: искусственный глаз, ухо, нос и др.;

2) решение сложных вычислительных задач (систем линейных уравнений, молекулярное конструирование лекарств, с получением легкого распараллеливания работы алгоритмов на основе НС);

3) моделирование работы структур человеческого мозга;

4) создание на основе концепции НС принципиально новых систем обработки информации со свойствами адаптации к быстро меняющейся обстановке, возможностями высокоскоростной обработки как аналоговой, так и дискретной информации, возможностью решения оптимизационных задач в реальном времени.

НК в основном используются для решения трудноформализуемых задач (качество алгоритма решения которых трудно оценить или вообще получить достижимое решение) или неформализуемых задач (с неявно заданными функциями и параметрами типа распознавания образов, предсказания, аппроксимации незаданных в явном виде функций и т. п.).

Опишем общую структурную схему абстрактного НК (рисунок 3.9). Операционным блоком НК (процессором) является искусственная НС, состоящая из формальных нейронов, соединенных каналами передачи информации. Этот блок в обычном понимании не производит вычислений. Он трансформирует входной сигнал (образ) в выходной в соответствии с топологией НС и значениями коэффициентов межнейронной связи. В запоминающем устройстве хранится не программа решения задачи, а программа изменений коэффициентов связи между нейронами. Устройства ввода и вывода информации выполняют обычные функции. Устройство управления служит для синхронизации работы всех структурных блоков НК при решении конкретной задачи.

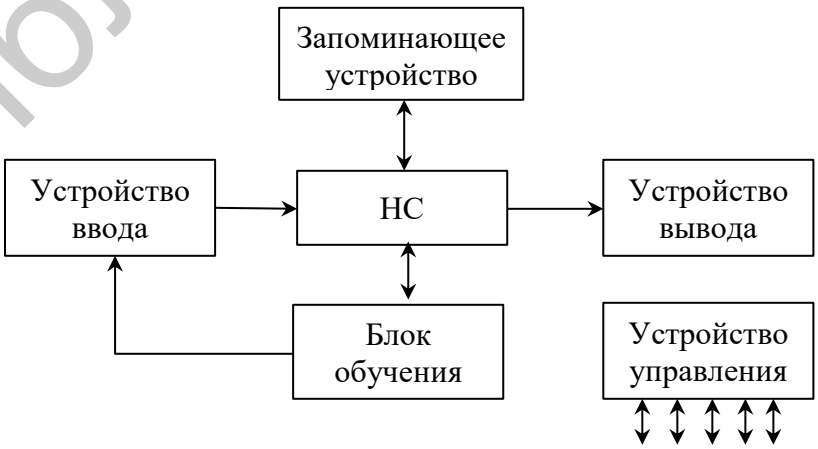


Рисунок 3.9 – Структурная схема абстрактного НК

В работе НК выделяют два режима:

- обучения;
- рабочий.

Чтобы НК решал требуемую задачу, его НС должна пройти обучение на совокупность входных образов этой задачи. Установка коэффициентов осуществляется на примерах, сгруппированных в обучающие множества (эталонные наборы).

Настройка и решаемая задача идут по итерационной схеме: при подаче очередного эталонного образа на НС выходной сигнал может отличаться от желаемого. Блок обучения оценивает величину ошибки и корректирует коэффициенты межнейронных связей с целью уменьшения ошибки на следующем шаге. По мере повторения процедуры ошибка уменьшается, и процесс обучения завершается при достижении ошибки менее заданной величины. Такой тип обучения относят к классу обучения с учителем.

В рабочем режиме блок обучения отключается.

Контрольные вопросы

1 В чем заключается особенность схемы эксперимента Тьюринга для оценки меры интеллектуальности системы?

2 В чем заключается особенность моделей ситуационного управления при описании объекта и создании схемы управления им?

3 Каковы основные особенности построения общего решателя задач?

4 Какие перспективы имеют методы диалогового решения задач и какова специфика их использования?

5 Какова специфика создания и использования экспертных систем?

6 Дайте общую характеристику задач по распознаванию образов и их типов.

7 Какие преимущества дает применение процессов обучения и самообучения НС при распознавании образов?

8 В чем заключается отличие базы данных от базы знаний?

9 Каковы основные функции систем управления базой данных (СУБД)?

10 В чем отличие банка данных от базы данных?

11 Чем отличаются процедурные и декларативные знания?

12 Назовите особенности продукционных моделей вывода.

13 Чем отличаются семантические представления знаний?

14 Назовите причины, побудившие развитие теории мультиагентных систем.

15 Назовите требования к агентам мультиагентных систем.

16 Что такое формальный нейрон?

17 Назовите структуры искусственных нейронных сетей.

18 Как обучаются нейронные сети?

18 Назовите типичные задачи, решаемые нейронными сетями.

Рекомендуемая литература [2, 3, 7, 8, 12–15].

ЗАКЛЮЧЕНИЕ

Перспективы развития интеллектуальных технологий. Перспективы использования различных систем «искусственного» интеллекта многообразны:

- стремительный рост количества «умной» бытовой техники, предполагающей дистанционное управление инфраструктурой жилых помещений и увеличения роботизированных систем жизнеобеспечения;
- развитие сервиса, связанного с повседневной жизнью человека (оплата различных услуг, управление всей системой жизнеобеспечения, основанной на современной технике, компоновка и запись кинофильмов, телерепортажей, идущих одновременно для последующего просмотра с автоматическим выделением их краткого дайджеста и т. д.);
- развитие систем совместного взаимодействия ЭВМ с мобильными средствами цифровой связи;
- развитие систем «электронных» правительств, обеспечивающих контакты с избирателями по получению информации о постановлениях и готовящихся проектах;
- управление из космоса движением на Земле транспортных средств и сельхозорудий;
- обеспечение систем безопасности в массовых мероприятиях, мгновенный анализ результатов видеонаблюдения в передвижении людей;
- создание автоматизированных производств с возможностями выполнять индивидуализацию изделий под требования заказчика;
- различные формы информационного обеспечения по индивидуальным заказам;
- создание виртуальных предприятий.

Можно специально выделить процессы обучения, так как их совершенствование коренным образом должно повлиять на индивидуальную подготовку отдельного специалиста:

- внедрение дистанционных систем обучения с индивидуальный скоростью подготовки для каждого учащегося;
- использование баз знаний во многих областях деятельности на уровне «экспертных» систем, когда их пользователь не будет иметь очень высокой квалификации в данной области, но обеспечит получение хороших общих решений в рамках поставленных ему задач;
- применение в преподавании вспомогательных систем, ускоряющих процесс обучения и позволяющих сосредоточиться учащимся на главных элементах (например, автоматическая нотная запись исполняемого музыкального произведения);
- анализ баз данных нейросетевыми методами с целью извлечения скрытых знаний в них;
- использование современных достижений науки в совершенствовании компьютеров и их внешних устройств (например, 3-*D* принтеров в моделировании и организации новых высокотехнологичных производств).

В последние годы большое внимание уделяется разработкам по организации прямого поиска и передаче актуальной информации в системах из разнородных технических и программных компонентов децентрализованной глобальной информационной сети (*Global Gnoseology Graph*, *GGG*-системы). Эти исследования открывают пути к глобальному получению информации в реальном масштабе времени работы пользователя из любого уголка Земли.

Проникновение нанотехнологий и мобильной связи в интеллектуальные информационные технологии. Нанотехнология – область науки и техники, занимающаяся разработкой теоретических и практических методов создания и применения продуктов с заданной атомной структурой путем контролируемого манипулирования отдельными атомами и молекулами. Размеры объектов в нанотехнологиях измеряются в нанометрах (1 нм – миллиардная часть метра), и их диапазон колеблется от 1 до 100 нм. Принципиальная особенность нанотехнологической революции – переход от движения в сторону миниатюризации в обратном направлении: складывание больших объектов из молекул и атомов. Развитие нанотехнологий идет на стыке ряда научных дисциплин и носит надотраслевой характер. Большие надежды возлагаются на разработки в робото- и компьютерной технике благодаря дальнейшей миниатюризации этих устройств и созданию нанотранзисторов.

Нанокомпьютер – вычислительное устройство на основе электронных (механических, биохимических, биологических, квантовых и др.) нанотехнологий. Их основой являются *нанотранзисторы* с реализацией логических переключательных функций на базовых элементах разной природы (биологических, электронных и др.).

Наиболее прогрессивной прикладной базой для построения нанотранзисторов являются *углеродные нанотрубки* (УНТ). Они характеризуются проводимостью около 1000 раз выше проводимости меди и имеют диаметр от 1,2 до 1,5 нм с размерами в длину до 360 нм. В нанодиапазоне электроны демонстрируют одновременно свойства волн и частиц. Заряд можно измерять только порциями (квантами). Половину электрона прибавить нельзя. При приложении небольшого электрического поля вдоль оси нанотрубки с ее концов происходит интенсивная эмиссия электронов (полевая эмиссия). Это позволяет создавать полевые транзисторы в качестве переключательных элементов. Одновременно появляется возможность на несколько порядков увеличить емкость и быстродействие памяти.

Огромное влияние эти достижения оказали на развитие *цифровой мобильной связи*. Мобильные средства связи сейчас позволяют обеспечить надежную беспроводную передачу информации в больших объемах благодаря встроенных в них миниатюрных модемов и компьютеров с возможностями для стационарных ЭВМ. Цифровая связь в состоянии одновременно обеспечить цифровую подпись информации и ее секретность.

Сейчас сети мобильной связи переживают очень бурное развитие благодаря расширению международной торговли и ряду специфических коммерческих услуг по передаче информации. Сетевые услуги мобильной связи постепенно преобразуются в «трубопроводы знаний» вследствие их сопряжения с бытовыми приборами, регистрационными и измерительными устройствами со встроенными микропроцессорами и приборами радиосвязи.

Благодаря выходу на спутниковую связь появилась возможность оказывать услуги мультимедийной связи и позиционирования подвижных объектов.

С точки зрения перспектив применения мобильной связи широко ее использование в сочетании с сетью Интернет, и следует обратить внимание на увеличение количества операций, связанных с передачей данных, развитием специфической рекламы и организацией производств и продаж.

Интеллектуализация принятия решений в сетевых технологиях. *Решение – это выбор альтернативы* в различных сферах деятельности.

Выделяют особо *организационные решения*, смысл которых – обеспечение движения к цели, т. е. создание условий для выполнения поставленной задачи. Число возможных альтернатив для рассмотрения, исходя из ресурсных возможностей (финансовых, технических, кадровых и т. п.), обычно ограничено, и часто удается назвать шаги, которые надо выполнить, и их порядок для реализации поставленной задачи. Такого типа решения можно запрограммировать и поручать их выработку компьютеру. Принятию таких решений способствуют часто повторяющиеся ситуации.

Обычно выделяют типичные группы решений по видам выполняемых функций планирования хода различных процессов, контроль.

Однако есть и ряд таких решений, отдельные этапы которых не поддаются программированию или допускают частичное программирование из-за наличия неизвестных и случайных факторов, которые трудно предвидеть. К таким объектам обычно относятся стратегические решения. Наиболее трудным являются те из них, в которых нужно преодолеть различного рода компромиссы (проблема занятости и переучивания работников из-за необходимости сохранить прибыльность предприятия, достижение оптимального решения за заданное ограниченное время и т. п.).

Принятие рациональных решений отличается тем, что они должны учитывать предшествующую историю развития технологии и/или предприятия и прогнозы развития событий. Поэтому информация для принятия решений базируется на результатах диагностики, которая позволяет сформулировать поле поиска альтернативных решений. Одновременно вырабатываются ограничения и критерии для принятия решений. К поиску рациональных решений побуждает и ограничение времени на их принятие, когда процесс выбора может прекращаться по критерию удовлетворения ограничений.

Риск в принятии решения должен опираться на оценки принимаемых решений по степени определенности их исхода, степень неопределенности может выражаться через вероятностную оценку. Ситуация значительной неопределенности наступает, когда невозможно оценить вероятность результатов. В частности, при изменении среды функционирования объекта в будущем, большими издержками времени и средств на получение информации, когда выгоды от ее получения сравнимы с затратами на ее добычу и обработку, трудность учета влияния принимаемых решений на технологию, условия труда и кадры.

Современные стандарты в области информационной безопасности, использующие концепцию управления рисками, очень важны в построении прикладных систем поддержки принятия решений.

Общие критерии предусматривают наличие двух типов требований безопасности: функциональных и доверия. Функциональные относятся к сервисам безопасности (идентификация, аутентификация, управление доступом). Требования доверия относятся к технологии разработки, тестированию, сопровождению, эксплуатационной документации и т. д. Работы должны производиться в выделенной вычислительной среде.

Интеллектуализация современных информационных компьютерных технологий становится главным элементом в повышении творческой роли человека в цепочке принятия решений, чтобы оттеснить его действия на уровни, которые трудно поддаются автоматизации (неопределенность ситуации, недостаток информации и т. п.). Пока задача о полной интеллектуализации принятия решений на машинном уровне не ставится. Речь идет о создании гибкой системы поддержки принятия решений, которая на формализуемых этапах оставляет их выбор за компьютером, а в сложных ситуациях для человека имеющаяся информация приводится в удобный вид для принятия окончательного решения или выбора пути продолжения процесса. Смысл использования системы поддержки принятия решений заключается в облегчении обработки больших объемов разнородной информации, включающей и такую, где необходимы советы специалистов различных профессий.

Особенно сложными стали управленческие решения при использовании управляющих и технических систем в едином комплексе, содержащих людей и программно-управляемое оборудование (станки с числовым программным управлением, роботы и т. п.). Сложность повышается еще и оттого, что принятое решение надо преобразовать в набор последовательностей сигналов для оборудования. Если же имеется еще и ряд критериев, среди которых есть противоречащие друг другу, то проблема дополнительно осложняется в выборе компромиссного варианта, который частично учитывает некоторые критерии. Например, выбор расписания преподавателя с минимумом ожидания, когда освободится специализированный кабинет, когда их недостаточно. При желании сохранить высокий уровень автоматизации принятия эффективных решений прибегают к использованию заранее выработанной системы приоритетов в конкретных ситуациях.

В современных технологиях процессов принятия решений считается важным использование сетецентрических (*GGG*) технологий в системах мониторинга и анализа с целью эффективного сбора нужной информации из разнородных средств (операционных систем, компьютеров, датчиков и т. п.) в реальном масштабе времени. В частности, такие системы оправдали себя при управлении военными операциями. В электронных правительствах с их помощью предполагается обеспечивать актуальной информацией в реальном времени управленцев.

Близкие к ним задачи решаются при использовании громадных объемов данных на основе витрин, в которых на мировом уровне содержится краткое описание поисковых объектов, чтобы по адресу выйти на их нужную часть.

Хранилища и витрины данных в системах поддержки принятия решений также важны при необходимости быстрого подбора целевой информации из хранилищ, рассредоточенных по всему миру для конкретного предприятия.

Концепция витрины данных – перечень множества тематических баз данных, содержащих информацию, относящуюся к различным аспектам деятельности организаций.

Такой подход полезен для выбора данных, которые действительно необходимы по причине их целенаправленного подбора из больших баз данных (по нескольким странам и т. п.).

Оперативные системы (ОС) с элементами искусственного интеллекта. ОС для конкретной ЭВМ с учетом состава ее оборудования и предполагаемого характера решаемых задач может с помощью специальных программ генерироваться в наиболее рациональном варианте с точки зрения затрат различных ресурсов на обслуживание пользователей. Таким образом, очевидно, что ОС является фактически довольно сложной автоматизированной системой специального назначения с рядом программ, имитирующих деятельность человека по эксплуатации и разработке программ. Создание таких ОС с элементами искусственного интеллекта за относительно короткий срок с момента появления машин, которые обслуживают одновременно нескольких пользователей, объясняется тем, что их создатели-программисты должны были алгоритмизировать свой собственный труд и им не требовалось привлекать знания из других областей деятельности человека.

Вместе с тем сложность современных ОС возросла до такой степени, что воспользоваться всеми возможностями, которые они могут предоставить, не всегда в состоянии даже опытный программист. Поэтому ОС для пользователей-непрофессионалов в программировании стали снабжать элементами «дружественного» интерфейса, направленного на облегчение диалога человека и машины. ЭВМ взяла на себя функции по обучению и консультированию пользователя при решении им конкретных задач: на базе тестовых примеров советовать пользователю, какой алгоритмический язык ему выбрать; на какие конструкции операторов и специальных средств следует ориентироваться; какими подсказками ЭВМ воспользоваться на базе ограниченного выбора действий («меню») и т. д.

Операционные системы с каждым днем продолжают совершенствоваться, и степень их доступности для пользователей будет упрощаться. Обобщение материалов по существующим ОС и перспективным разработкам ОС идет пока медленно. Это связано с тем, что большинство учебников по ОС тесно привязано к конкретным маркам машин, но постепенно общие закономерности начинают выкристаллизовываться и находить отражение в литературе по системному программированию. Следует отметить, что долгие годы работа по обеспечению «дружественного» интерфейса системными программистами-профессионалами считалась

черновой и неинтересной, так как она была направлена лишь на облегчение освоения ОС пользователем и не касалась основных функций ОС.

Время внесло свои коррективы в этот подход. Постепенно количественный рост числа системных программистов и вообще чистых программистов замедлился. С каждым днем возрастает число пользователей-непрограммистов, желающих применять вычислительную технику для решения задач в своей области. Количественное преобладание таких пользователей ЭВМ проблему создания «дружественного» интерфейса для них сделало главной: стали появляться специальные технические и программные средства для облегчения контакта непрофессионала с ЭВМ. Работа со многими программными средствами стала сводиться к управлению движением маркера на экране дисплея в нужных областях, к указке «световым» пером нужного объекта или его элемента, звуковому вводу информации на основе ограниченного словаря естественного языка и т. д.

Сейчас все яснее намечается тенденция к сведению интерфейса человека с ЭВМ к привычной для него схеме диалога с другими людьми. Естественно, как отмечал академик А. П. Ершов, придется пока считаться с трудностями проблемы общения и на первых порах пользоваться в «разговоре» с ЭВМ языком «деловой прозы», т. е. в соответствии с ориентацией на решение определенных задач будет и соответствующее языковое обеспечение данной группы специалистов. Таким образом, ОС, по-видимому, не сможет впитать в себя все виды программ, обеспечивающих диалог в специальных областях, но должна постепенно впитывать в себя элементы, пригодные для всех областей деятельности.

Идеи, развитые при создании и совершенствовании операционных систем, оказались весьма плодотворными в автоматизации процессов разработки алгоритмов и решения различных классов задач. Современным ОС присуща многоплатформенность, т. е. способность работать на совершенно различных типах компьютеров.

ЛИТЕРАТУРА

1 Логика. Автоматы. Алгоритмы / М. А. Айзерман [и др.]. – М. : Физматгиз, 1963. – 556 с.

2 Представление и обработка знаний в графодинамических ассоциативных машинах: Монография / В. В. Голенков [и др.]. – Минск : БГУИР, 2001. – 412 с.

3 Экспертные системы для персональных компьютеров / В. В. Краснопрошин [и др.]. – Минск : Выш. шк., 1990. – 198 с.

4 Таненбаум, А. Архитектура компьютера / А. Таненбаум; пер. с англ. – СПб. : Питер, 2002. – 704 с.

5 Бибило, П. Н. Основы языка VHDL / П. Н. Бибило. – М. : Солон, 2002. – 224 с.

6 Баричев, С. Г. Основы современной криптографии / С. Г. Баричев, В. В. Гончаров, Р. Е. Серов. – М. : Горячая линия, Телеком, 2001. – 120 с.

7 Татур, М. М. Классификаторы в системах распознавания: прикладные авпекты синтеза и анализа / М. М. Татур, Д. Н. Одинец. – Минск : Бестпринт, 2008. – 165 с.

8 Головко, В. А. Основы компьютерных технологий: учеб.-метод. пособие / В. А. Головко, А. А. Дудкин, Л. П. Матюшков. – 2-е изд., испр. и доп. – Брест : БрГТУ, 2015. – 196 с.

9 Дэо, Н. Комбинаторные алгоритмы / Н. Дэо, Ю. Нивергельт, Э. Рейнгольд; пер. с англ. – М. : Мир, 1980. – 476 с.

10 Кемени, Дж. Кибернетическое моделирование / Дж. Кемени, Дж. Снелл. – М. : Сов. радио, 1972. – 192 с.

11 Программирование и алгоритмические языки / Н. А. Криницкий [и др.]. – М. : Наука, 1979. – 509 с.

12 Базы данных. Интеллектуальная обработка информации / В. Г. Корнеев [и др.] – М. : Нолидж, 2001. – 496 с.

13 Лисьев, Г. А. Технология поддержки принятия решений / Г. А. Лисьев, И. В. Попова. – М. : Флинта, 2011. – 133 с.

14 Попов, Э. В. Общение с ЭВМ на естественном языке / Э. В. Попов. – М. : Наука, 1982.– 360 с.

15 Поспелов, Г. С. Искусственный интеллект – прикладные системы / Г. С. Поспелов, Д. А. Поспелов. – М. : Знание, 1985. – 46 с.

16 Успенский, В. А. Машина Поста / В. А. Успенский. – М. : Наука, 1979. – 95 с.

17 Гэри, М. Вычислительные машины и труднорешаемые задачи / М. Гэри, Д. Джонсон; пер. с англ. – М.: Мир, 1982.− 416 с.

18 Старовойтов, В. В. Цифровые изображения: от получения до обработки / В. В. Старовойтов, Ю. И. Голуб. – Минск : ОИПИ НАН Беларуси, 2014. – 202 с.

19 Авдеев, В. А. Периферийные устройства: интерфейсы, схемотехника, программирование / В. А. Авдеев. – М. : ДМК Пресс, 2012. – 848 с.

20 Гук, М. Ю. Аппаратные интерфейсы ПК. Энциклопедия / М. Ю. Гук. – СПб. : Питер, 2006. – 1072 с.

21 Гук, М. Ю. Шины *PCI, USB* и *FireWire*. Энциклопедия / М. Ю. Гук. – СПб. : Питер, 2005. – 540 с.

22 Гук, М. Интерфейсы устройств хранения: *ATA, SCSI* и др. Энциклопедия / М. Гук. – СПб. : Питер, 2006. – 448 с.

23 Таненбаум, Э. С. Архитектура компьютера / Э. С. Таненбаум. – 5-е изд. (+*CD*). – СПб. : Питер, 2007. – 848 с.

24 Абламейко С. В. Цифровая картография: история и этапы разработок отечественных технологий в институте / С. В. Абламейко, А. Н. Крючков // Информатика. – 2004. – №4.

25 Алексеев, Г. И. Планшетные устройства ввода изображений / Г. И. Алексеев // Информатика. – 2004. – №4.

Св. план 2017, поз. 26

Учебное издание

Дудкин Александр Арсентьевич
Самаль Дмитрий Иванович
Головко Владимир Адамович
Матюшков Леонид Петрович

ИНТЕЛЛЕКТУАЛЬНЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В КОМПЬЮТЕРНЫХ СИСТЕМАХ И СЕТЯХ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Редактор *Е. В. Иванюшина*
Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *О. И. Толкач*

Подписано в печать 13.02.2019. Формат 60×84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 9,18. Уч.-изд. л. 10,3. Тираж 100 экз. Заказ 275.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
ЛП №02330/264 от 14.04.2014.
220013, Минск, П. Бровки, 6