Undergraduate Topics in Computer Science

Gerard O'Regan

Mathematics in Computing

An Accessible Guide to Historical, Foundational and Application Contexts

Second Edition

Springer

Gerard O'Regan
SQC Consulting
Mallow, Cork, Ireland

ISSN 1863-7310 ISSN 2197-1781 (electronic)
Undergraduate Topics in Computer Science
ISBN 978-3-030-34208-1 ISBN 978-3-030-34209-8 (eBook)
https://doi.org/10.1007/978-3-030-34209-8

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Contents

List of Figures

What Is a Computer?

1

Key Topics

Analog computers
Digital computers
Vacuum tubes
Transistors
Integrated circuits
von Neumann architecture
Generation of computers
Hardware
Software

1.1 Introduction

It is difficult to think of western society today without modern technology. We have witnessed in recent decades a proliferation of high-tech computers, mobile phones, text messaging, the Internet, the World Wide Web and social media. Software is pervasive, and it is an integral part of automobiles, airplanes, televisions and mobile communication. The pace of change is relentless, and communication today is instantaneous with technologies such as Skype, Twitter and WhatsApp.

Today, people may book flights over the World Wide Web as well as keeping in contact with friends and family members around the world. In previous generations, communication involved writing letters that often took months to reach the

G. O'Regan, *Mathematics in Computing*, Undergraduate Topics in Computer Science, https://doi.org/10.1007/978-3-030-34209-8_1

recipient. However, today's technology has transformed the modern world into a global village, and the modern citizen may make video calls over the Internet or post pictures and videos on social media sites such as Facebook and Twitter. The World Wide Web allows business to compete in a global market.

A computer is a programmable electronic device that can process, store and retrieve data. It processes data according to a set of instructions or program. All computers consist of two basic parts, namely, *hardware* and *software*. The hardware is the physical part of the machine, and the components of a digital computer include memory for short-term storage of data or instructions; an arithmetic/logic unit for carrying out arithmetic and logical operations; a control unit responsible for the execution of computer instructions in memory; and peripherals that handle the input and output operations. Software is a set of instructions that tells the computer what to do.

The original meaning of the word '*computer*' referred to someone who carried out calculations rather than an actual machine. The early digital computers built in the 1940s and 1950s were enormous machines consisting of thousands of vacuum tubes. They typically filled a large room but their computational power was a fraction of the personal computers and mobile devices used today.

There are two distinct families of computing devices, namely, *digital computers* and the historical *analog computer*. The earliest computers were analog not digital, and these two types of computer operate on quite different principles.

The computation in a digital computer is based on binary digits, i.e. '0' and '1'. Electronic circuits are used to represent binary numbers, with the state of an electrical switch (i.e. 'on' or 'off') representing a binary digit internally within a computer.

A digital computer is a sequential device that generally operates on data one step at a time, and the earliest digital computers were developed in the 1940s. The data are represented in binary format, and a single transistor (initially bulky vacuum tubes) is used to represent a binary digit. Several transistors are required to store larger numbers.

An *analog computer* operates in a completely different way to a digital computer. The representation of data in an analog computer reflects the properties of the data that are being modelled. For example, data and numbers may be represented by physical quantities such as electric voltage, whereas a stream of binary digits is used to represent them in a digital computer.

1.2 Analog Computers

James Thompson (who was the brother of the physicist Lord Kelvin) did early foundational work on analog computation in the nineteenth century. He invented a wheel-and-disc integrator, which was used in mechanical analog devices, and he worked with Kelvin to construct a device to perform the integration of a product of two functions. Kelvin later described a general-purpose analog machine (he did not

build it) for integrating linear differential equations. He built a tide predicting analog computer that remained in use at the Port of Liverpool up to the 1960s.

The operations in an analog computer are performed in parallel, and they are useful in simulating dynamic systems. They have been applied to flight simulation, nuclear power plants and industrial chemical processes.

Vannevar Bush developed the first large-scale general-purpose mechanical analog computer at the Massachusetts Institute of Technology. Bush's differential analyser (Fig. 1.1) was a mechanical analog computer designed to solve sixth-order differential equations by integration, using wheel-and-disc mechanisms to perform the integration. The mechanization allowed integration and differential equations problems to be solved more rapidly. The machine took up the space of a large table in a room and weighed about 100 tonnes.

It contained wheels, discs, shafts and gears to perform the calculations. It required a considerable setup time by technicians to solve an equation. It contained 150 motors and miles of wires connecting relays and vacuum tubes.

Data representation in an analog computer is compact, but it may be subject to corruption with noise. A single capacitor can represent one continuous variable in an analog computer. Analog computers were replaced by digital computers shortly after the Second World War.

Fig. 1.1 Vannevar Bush with the differential analyser

1.3 Digital Computers

Early digital computers used vacuum tubes to store binary information, and a vacuum tube may represent the binary value '0' or '1'. These tubes were large and bulky and generated a significant amount of heat. Air conditioning was required to cool the machine, and there were problems with the reliability of the tubes.

Shockley and others invented the transistor in the late 1940s, and it replaced vacuum tubes from the late 1950s onwards. Transistors are small and consume very little power, and the resulting machines were smaller, faster and more reliable.

Integrated circuits were introduced in the early 1960s, and a massive amount of computational power could now be placed on a very small chip. Integrated circuits are small and consume very little power, and may be mass-produced to a very high-quality standard. However, integrated circuits are difficult to modify or repair, and are nearly always replaced on failure.

The fundamental architecture of a computer has remained basically the same since von Neumann and others proposed it in the mid-1940s. It includes a central processing unit which includes the control unit and the arithmetic unit, an input and output unit, and memory.

1.3.1 Vacuum Tubes

A vacuum tube is a device that relies on the flow of an electric current through a vacuum. Vacuum tubes (*thermionic valves*) were widely used in electronic devices such as televisions, radios and computers until the invention of the transistor.

The basic idea of a vacuum tube is that the current passes through the filament, which then heats it up so that it gives off electrons. The electrons are negatively charged and are attracted to the small positive plate (or anode) within the tube. A unidirectional flow is thus established between the filament and the plate. Thomas Edison had observed this while investigating the reason for breakage of lamp filaments. He noted an uneven blackening (darkest near one terminal of the filament) of the bulbs in his incandescent lamps and noted that current flows from the lamp's filament and a plate within the vacuum.

The first generation of computers used several thousand bulky vacuum tubes, with several racks of vacuum tubes taking up the space of a large room. The vacuum tube used in the early computers was a three-terminal device, and it consisted of a cathode, a grid and a plate. It was used to represent one of two binary states, i.e. the binary value '0' or '1'.

The filament of a vacuum tube becomes unstable over time. In addition, if air leaks into the tube then oxygen will react with the hot filament and damage it. The size and unreliability of vacuum tubes motivated research into more compact and reliable technologies. This led to the invention of the transistor in the late 1940s.

The first generation of digital computers all used vacuum tubes, e.g. the Atanasoff–Berry computer (ABC) developed at the University of Iowa in 1942;

Colossus developed at Bletchley Park, England in 1944; and ENIAC developed in the United States in the mid-1940s.

1.3.2 Transistors

The transistor is a fundamental building block in modern electronic systems, and its invention revolutionized the field of electronics. It was smaller, cheaper and more reliable than the existing vacuum tubes.

The transistor is a three-terminal, solid-state electronic device. It can control electric current or voltage between two of the terminals by applying an electric current or voltage to the third terminal. The three-terminal transistor enables an electric switch to be made which can be controlled by another electrical switch. Complicated logic circuits may be built up by cascading these switches (switches that control switches that control switches, and so on.).

These logic circuits may be built very compactly on a silicon chip with a density of over a million transistors per square centimetre. The switches may be turned on and off very rapidly (e.g. every 0.000000001 s). These electronic chips are at the heart of modern electronic devices.

The transistor (Fig. 1.2) was developed at Bell Labs after the Second World War. The goal of the research was to find a solid-state alternative to vacuum tubes, as this technology was too bulky and unreliable. Three Bell Labs inventors (Shockley, Bardeen and Brattain) were awarded the Nobel Prize in physics in 1956 in recognition of their invention of the transistor.

William Shockley was involved in radar research and anti-submarine operations research during the Second World War, and after the war he led the Bell Labs research group (that included Bardeen and Brattain) that aimed to find a solid-state alternative to the glass-based vacuum tubes.

Bardeen and Brattain succeeded in creating a point-contact transistor in 1947 independently of Shockley who was working on a junction-based transistor. Shockley believed that the points contact transistor would not be commercially viable, and his junction point transistor was announced in 1951.

Shockley formed Shockley Semiconductor Inc. (part of Beckman Instruments) in 1955. The second generation of computers used transistors instead of vacuum tubes. The University of Manchester's experimental Transistor Computer was one of the earliest transistor computers. The prototype machine appeared in 1953, and the full-size version was commissioned in 1955. The invention of the transistor is discussed in more detail in (O'Regan 2018).

1.3.3 Integrated Circuits

Jack Kilby of Texas Instruments invented the integrated circuit in 1958. His invention used a wafer of germanium, and Robert Noyce of Fairchild Semiconductors did subsequent work on silicon-based integrated circuits. The integrated

Fig. 1.2 Replica of transistor. Public domain

circuit was an effective solution to the problem of building a circuit with many components, and the Nobel Prize in Physics was awarded to Kirby in 2000 for his contributions to its invention.

An integrated circuit consists of a set of electronic circuits on a small chip of semiconductor material, and it is much smaller than a circuit made from independent components. Integrated circuits today are extremely compact and may contain billions of transistors and other electronic components in a tiny area. The width of each conducting line has got smaller and smaller over the years due to advances in technology, and it is now measured in tens of nanometers.

The number of transistors per unit area has been doubling (roughly) every 1–2 years over the last 30 years. This amazing progress in circuit fabrication is known as Moore's law after Gordon Moore (one of the founders of Intel) who formulated the law in the mid-1960s (O'Regan 2018).

Kilby was designing micromodules for the military, and this involved connecting many germanium[1] wafers of discrete components together by stacking each wafer on top of one another. The connections were made by running wires up the sides of the wafers.

Kilby saw this process as unnecessarily complicated and realized that if a piece of germanium was engineered properly that it could act as many components simultaneously. That is, instead of making transistors one-by-one several transistors

[1]Germanium is an important semiconductor material used in transistors and other electronic devices.

could be made at the same time on the same piece of semiconductor. In other words, transistors and other electric components such as resistors, capacitors and diodes can be made by the same process with the same materials.

This idea led to the birth of the first integrated circuit and its development involved miniaturizing transistors and placing them on silicon chips called semiconductors. The use of semiconductors led to third-generation computers, with a major increase in speed and efficiency.

Users interacted with third-generation computers through keyboards and monitors and interfaced with an operating system, which allowed the device to run several applications at one time with a central program that monitored the memory. Computers became accessible to a wider audience, as they were smaller and cheaper than their predecessors.

1.3.4 Microprocessors

The Intel P4004 microprocessor was the world's first microprocessor, and it was released in 1971. It was the first semiconductor device that provided, at the chip level, the functions of a computer.

The invention of the microprocessor happened by accident rather than design. Busicom, a Japanese company, requested Intel to design a set of integrated circuits for its new family of high-performance programmable calculators. Ted Hoff, an Intel engineer, studied Busicom's design and rejected it as unwieldy. He proposed a more elegant solution requiring just four integrated circuits (Busicom required twelve integrated circuits), and his design included a chip that was a general-purpose logic device that derived its application instructions from the semiconductor memory. This was the Intel 4004 microprocessor.

It provided the basic building blocks that are used in today's microcomputers, including the arithmetic and logic unit and the control unit. The 4-bit Intel 4004 ran at a clock speed of 108 kHz and contained 2,300 transistors. It processed data in 4 bits, but its instructions were 8-bit long. It could address up to 1 Kb of program memory and up to 4 Kb of data memory.

Gary Kildall of Digital Research was one of the early people to recognize the potential of a microprocessor as a computer. He worked as a consultant with Intel, and he began writing experimental programs for the Intel 4004 microprocessor. He later developed the CP/M operating system for the Intel 8080 chip, and he set up Digital Research to commercialize the operating system.

The development of the microprocessor led to the fourth generation of computers with thousands of integrated circuits placed onto a single silicon chip. A single chip could now contain all the components of a computer from the CPU and memory to input and output controls. It could fit in the palm of the hand, whereas first generation of computers filled an entire room.

Table 1.1 von Neumann architecture

Component	Description
Arithmetic unit	The arithmetic unit can perform basic arithmetic operations
Control unit	The program counter contains the address of the next instruction to be executed. This instruction is fetched from memory and executed. This is the basic fetch-and-execute cycle (Fig. 1.4) The control unit contains a built-in set of machine instructions
Input–output unit	The input and output unit allows the computer to interact with the outside world
Memory	The one-dimensional memory stores all program instructions and data. These are usually kept in different areas of memory The memory may be written to or read from, i.e. it is random access memory (RAM) The program instructions are binary values, and the control unit decodes the binary value to determine which instruction to execute

1.4 von Neumann Architecture

The earliest computers were fixed programs machines that were designed to do a specific task. This proved to be a major limitation as it meant that a complex manual rewiring process was required to enable the machine to solve a different problem.

The computers used today are general-purpose machines designed to allow a variety of programs to be run on the machine. von Neumann and others (von Neumann 1945) described the fundamental architecture underlying the computers used today in the late 1940s. It is known as von Neumann architecture (Fig. 1.3).

The von Neumann architecture arose on work done by von Neumann, Eckert, Mauchly and others on the design of the EDVAC computer (which was the successor to ENIAC computer). von Neumann's draft report on EDVAC (von Neumann 1945) described the new architecture[2] (Table 1.1).

The architecture led to the birth of stored-program computers, where a single store is used for both machine instructions and data. Its key components are as follows:

The key approach to building a general-purpose device according to von Neumann was in its ability to store not only its data and the intermediate results of computation, but also to store the instructions or commands for the computation. The computer instructions can be part of the hardware for specialized machines, but for general-purpose machines the computer instructions must be as changeable as

[2]Eckert and Mauchly were working with him on this concept during their work on ENIAC and EDVAC, but their names were removed from the final report due to their resignation from the University of Pennsylvania to form their own computer company. von Neumann architecture includes a central processing unit which includes the control unit and the arithmetic unit, an input and output unit, and memory.

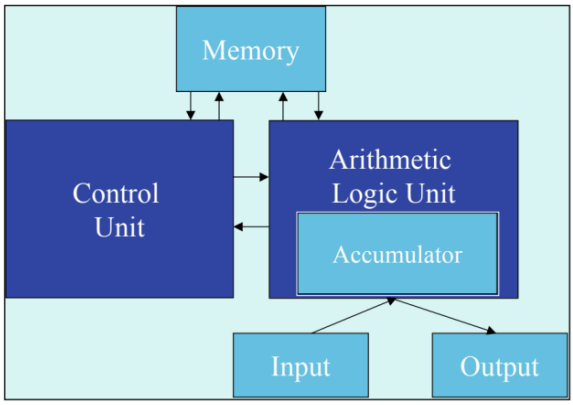


Fig. 1.3 von Neumann architecture

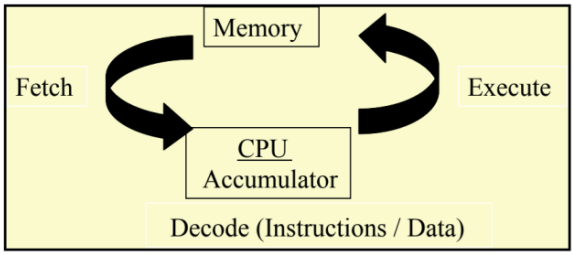


Fig. 1.4 Fetch/execute cycle

the data that is acted upon by the instructions. *His insight was to recognize that both the machine instructions and data could be stored in the same memory.*

The key advantage of the von Neumann architecture over the existing approach was that it was much simpler to reconfigure a computer to perform a different task. All that was required was to enter new machine instructions in computer memory rather than physically rewiring a machine as was required with ENIAC. The limitations of von Neumann architecture include that it is limited to sequential processing and not very suitable for parallel processing.

1.5 Hardware and Software

Hardware is the physical part of the machine. It is tangible and may be seen or touched, and includes punched cards, vacuum tubes, transistors and circuit boards, integrated circuits and microprocessors. The hardware of a personal computer includes a keyboard, network cards, a mouse, a DVD drive, hard disc drive, printers and scanners and so on.

Software is intangible and consists of a set of instructions that tells the computer what to do. It is an intellectual creation of a programmer (or a team of programmers).

Operating system software manages the computer hardware and resources and acts as an intermediary between the application programs and the computer hardware.

Application software refers to software programs that provide functionality for users to exploit the power of the computer to perform useful tasks such as business applications including spreadsheets and accountancy packages, financial applications, editors, compilers for programming languages, computer games, social media and so on.

1.6 Review Questions

1. Explain the difference between analog and digital computers.
2. Explain the difference between hardware and software.
3. What is a microprocessor?
4. Explain the difference between vacuum tubes, transistors and integrated circuits.
5. Explain the von Neumann architecture.
6. What are the advantages and limitations of the von Neumann architecture?
7. Explain the difference between a fixed program machine and a stored-program machine.

1.7 Summary

A computer is a programmable electronic device that can process, store and retrieve data. It processes data according to a set of instructions or program. All computers consist of two basic parts, namely, the hardware and software. The hardware is the physical part of the machine, whereas software is intangible and is the set of instructions that tells the computer what to do.

There are two distinct families of computing devices, namely, digital computers and the historical analog computer. These two types of computer operate on quite different principles. The earliest digital computers were built in the 1940s, and these were large machines consisting of thousands of vacuum tubes. However, their computational power was a fraction of what is available today.

A digital computer is a sequential device that generally operates on data one step at a time. The data are represented in binary format, and a single transistor in a digital computer can store only two states, i.e. on and off. Several transistors are required to store larger numbers.

The representation of data in an analog computer reflects the properties of the data that is being modelled. Data and numbers may be represented by physical quantities such as electric voltage, whereas a stream of binary digits represents the data in a digital computer.

von Neumann architecture is the fundamental architecture used on digital computers, and a single store is used for both machine instructions and data. Its introduction made it much easier to reconfigure a computer to perform a different task. All that was required was to enter new machine instructions in computer memory rather than physically rewiring a machine as was required with ENIAC.

References

O'Regan G (2018) World of computing. Springer
O'Regan G (2018) Giants of computing. Springer
von Neumann J (1945) First draft of a report on the EDVAC. University of Pennsylvania

Foundations of Computing

2

Key Topics

Leibniz
Binary numbers
Step reckoner
Babbage
Difference engine
Analytic engine
Lovelace
Boole
Shannon
Switching circuits

2.1 Introduction

This chapter considers important foundational work done by Wilhelm Leibniz, Charles Babbage, George Boole, Lady Ada Lovelace and Claude Shannon. Leibniz was a seventeenth-century German mathematician, philosopher and inventor, and he is recognized (with Isaac Newton) as the inventor of Calculus. He developed the Step Reckoner calculating machine that could perform all four basic arithmetic operations (i.e. addition, subtraction, multiplication and division), and he also invented the binary number system (which is used extensively in the computer field).

G. O'Regan, *Mathematics in Computing*, Undergraduate Topics in Computer Science, https://doi.org/10.1007/978-3-030-34209-8_2

Boole and Babbage are considered grandfathers of the computing field, with Babbage's Analytic Engine providing a vision of a mechanical computer, and Boole's logic providing the foundation for modern digital computers.

Babbage was a nineteenth-century scientist and inventor who did pioneering work on calculating machines. He invented the Difference Engine (a sophisticated calculator that could be used to produce mathematical tables), and he also designed the Analytic Engine (the world's first mechanical computer). The design of the Analytic Engine included a processor, memory and a way to input information and output results.

Lady Ada Lovelace was introduced into Babbage's ideas on the analytic engine at a dinner party. She was fascinated and predicted that such a machine could be used to compose music, produce graphics, as well as solving mathematical and scientific problems. She explained how the Analytic Engine could be programmed, and she wrote what is considered the first computer program.

Boole was a nineteenth-century English mathematician who made important contributions to mathematics, probability theory and logic. Boole's logic provides the foundation for digital computers.

Shannon was the first person to apply Boole's logic to switching theory, and he showed that this could simplify the design of circuits and telephone routing switches. It provides the perfect mathematical model for switching theory and for the subsequent design of digital circuits and computers.

2.2 Step Reckoner Calculating Machine

Leibniz (Fig. 27.6) was a German philosopher, mathematician and inventor in the field of mechanical calculators. He developed the binary number system used in digital computers, and he invented the Calculus independently of Sir Isaac Newton. He became familiar with Pascal's calculating machine, the *Pascaline*, while in Paris in the early 1670s. He recognized its limitations as the machine could perform addition and subtraction operations only.

He designed and developed a calculating machine that could perform addition, subtraction, multiplication, division and the extraction of roots. He commenced work on the machine in 1672, and the machine was completed in 1694. It was the first calculator that could perform all four arithmetic operations, and it was called the *Step Reckoner* (Fig. 2.2). It allowed the common arithmetic operations to be carried out mechanically (Fig. 2.1).

The operating mechanism used in his calculating machine was based on a counting device called the stepped cylinder or '*Leibniz wheel*'. This mechanism allows a gear to represent a single decimal digit from zero to nine in just one revolution, and this remained the dominant approach to the design of calculating machines for the next 200 years. It was essentially a counting device consisting of a set of wheels that were used in calculation. The Step Reckoner consisted of an accumulator which could hold 16 decimal digits and an 8-digit input section. The

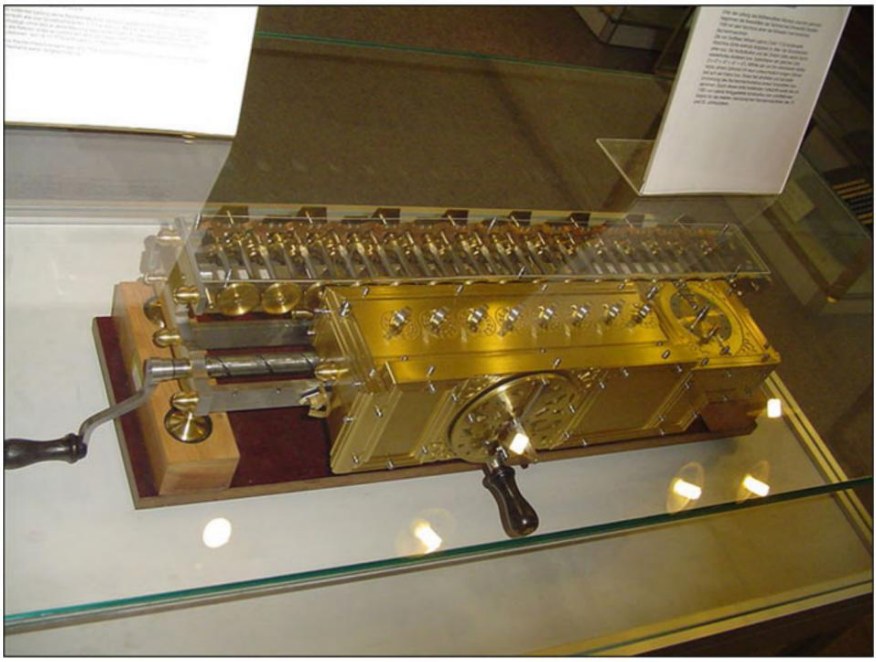


Fig. 2.1 Replica of step reckoner at Technische Sammlungen Museum, Dresden

eight dials at the front of the machine set the operand number, which was then employed in the calculation.

The machine performed multiplication by repeated addition and division by repeated subtraction. The basic operation is to add or subtract the operand from the accumulator as many times as desired. The machine could add or subtract an 8-digit number to the 16-digit accumulator to form a 16-digit result. It could multiply two 8-digit numbers to give a 16-digit result, and it could divide a 16-bit number by an 8-digit number. Addition and subtraction are performed in a single step, with the operating crank turned in the opposite direction for subtraction. The result is stored in the accumulator.

2.3 Binary Numbers

Arithmetic has traditionally been done using the decimal notation,[1] and Leibniz was one of the first to recognize the potential of the binary number system. This system uses just two digits, namely, '0' and '1', with the number two represented by 10, the

[1]The sexagesimal (or base-60) system was employed by the Babylonians c. 2000 BC. Indian and Arabic mathematicians developed the decimal system between 800 and 900 AD.

Table 2.1 Binary number system

Binary	Dec.	Binary	Dec.	Binary	Dec.	Binary	Dec.
0000	0	0100	4	1000	8	1100	12
0001	1	0101	5	1001	9	1101	13
0010	2	0110	6	1010	10	1110	14
0011	3	0111	7	1011	11	1111	15

number four by 100 and so on. Leibniz described the binary system in *Explication de l'Arithmétique Binaire* (Leibniz 1703), which was published in 1703. A table of values for the first 15 binary numbers is given in Table 2.1.

Leibniz's (1703) paper describes how binary numbers may be added, subtracted, multiplied and divided, and he was an advocate of their use. The key advantage of the use of binary notation is in digital computers, where a binary digit may be implemented by an *on/off* switch, with the digit 1 representing that the switch is on, and the digit 0 representing that the switch is off.

The use of binary arithmetic allows more complex mathematical operations to be performed by relay circuits, and Boole's Logic (described in a later section) is the perfect model for simplifying such circuits and is the foundation underlying digital computing.

The binary number system (base 2) is a positional number system, which uses two binary digits 0 and 1, and an example binary number is 1001.01_2 which represents $1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = 1 \times 2^3 + 1 \times 2^0 + 1 \times 2^{-2} = 8 + 1 + 0.25 = 9.25$.

The decimal system (base 10) is more familiar for everyday use, and there are algorithms to convert numbers from decimal to binary and vice versa. For example, to convert the decimal number 25 to its binary representation 11001_2 we proceed as follows (Fig. 2.2):

The base 2 is written on the left, and the number to be converted to binary is placed in the first column. At each stage in the conversion, the number in the first column is divided by 2 to form the quotient and remainder, which are then placed on the next row. For the first step, the quotient when 25 is divided by 2 is 12 and the remainder is 1. The process continues until the quotient is 0, and the binary representation result is then obtained by reading the second column from the bottom up. Thus, we see that the binary representation of 25 is 11001_2.

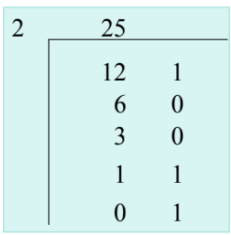


Fig. 2.2 Decimal to binary conversion

Similarly, there are algorithms to convert decimal fractions to binary representation (to a defined number of binary digits as the representation may not terminate), and the conversion of a number that contains an integer part and a fractional part involves converting each part separately and then combining them.

The octal (base 8) and hexadecimal (base 16) are often used in computing, as the bases 2, 8 and 16 are related bases and easy to convert between, as to convert between binary and octal involves grouping the bits into groups of three on either side of the point. Each set of 3 bits corresponds to one digit in the octal representation. Similarly, the conversion between binary and hexadecimal involves grouping into sets of 4 digits on either side of the point. The conversion from octal to binary or hexadecimal to binary is equally simple and involves replacing the octal (or hexadecimal) digit with the 3-bit (or 4-bit) binary representation.

Numbers are represented in a digital computer as sequences of bits of fixed length (e.g. 16 bits, 32 bits). There is a difference in the way in which integers and real numbers are represented, with the representation of real numbers being more complicated.

An integer number is represented by a sequence of (usually 2 or 4) bytes where each byte is 8 bits. For example, a 2-byte integer has 16 bits with the first bit used as the sign bit (the sign is 1 for negative numbers and 0 for positive integers), and the remaining 15 bits represent the number. This means that 2 bytes may be used to represent all integer numbers between −32768 and 32767. A positive number is represented by the normal binary representation discussed earlier, whereas a negative number is represented using 2's complement of the original number (i.e. 0 changes to 1 and 1 changes to 0 and the sign bit is 1). All the standard arithmetic operations may then be carried out (using modulo-2 arithmetic).

The representation of floating-point real numbers is more complicated, and a real number is represented to a fixed number of significant digits (the significand) and scaled using an exponent in some base (usually 2). That is, the number is represented (approximated) as

$$\text{significand} \times \text{base}^{\text{exponent}}$$

The significand (also called mantissa) and exponent have a sign bit. For example, in simple floating-point representation (4 bytes), the mantissa is generally 24 bits and the exponent 8 bits, whereas for double precision (8 bytes) the mantissa is generally 53 bits and the exponent 11 bits. There is an IEEE standard for floating-point numbers (IEEE 754).

2.4 The Difference Engine

Babbage (Fig. 2.3) is considered (along with Boole) to be one of the grandfathers of the computing field. He contributed to several areas including mathematics, statistics, astronomy, calculating machines, philosophy, railways and lighthouses. He founded the British Statistical Society and the Royal Astronomical Society.

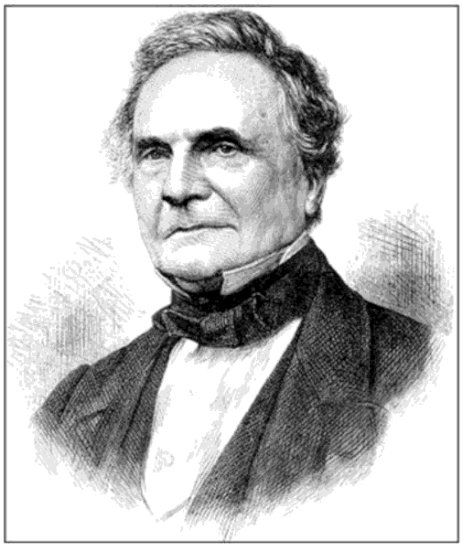


Fig. 2.3 Charles Babbage

Babbage was interested in accurate mathematical tables for scientific work. However, there was a high error rate in the existing tables due to human error introduced during calculation. He became interested in finding a mechanical method to perform calculation to eliminate the errors introduced by humans. He planned to develop a more advanced machine than the Pascaline or the Step Reckoner, and his goal was to develop a machine that could compute polynomial functions.

He designed the Difference Engine (No. 1) in 1821 for the production of mathematical tables. This was essentially a mechanical calculator (analogous to modern electronic calculators), and it was designed to compute polynomial functions of degree 4. It could also compute logarithmic and trigonometric functions such as sine or cosine (as these may be approximated by polynomials).[2]

The accurate approximation of trigonometric, exponential and logarithmic functions by polynomials depends on the degree of the polynomials, the number of decimal digits that it is being approximated to, and on the error function. A higher degree polynomial is generally able to approximate the function more accurately.

Babbage produced prototypes for parts of the Difference Engine, but he never actually completed the machine. The Swedish engineers, Georg and Edvard Scheutz, built the first working Difference Engine (based on Babbage's design) in 1853 with funding from the Swedish government. Their machine could compute

[2]The power series expansion of the Sine function is given by $\mathrm{Sin}(x) = x - x^3/3! + x^5/5! - x^7/7! + \cdots$. The power series expansion for the Cosine function is given by $\mathrm{Cos}(x) = 1 - x^2/2! + x^4/4! - x^6/6! + \cdots$. Functions may be approximated by interpolation and the approximation of a function by a polynomial of degree n requires $n + 1$ points on the curve for the interpolation. That is, the curve formed by the polynomial of degree n that passes through the $n + 1$ points of the function to be approximated is an approximation to the function. The error function also needs to be considered.

Table 2.2 Analytic engine

Part	Function
Store	This contains the variables to be operated upon as well as all those quantities, which have arisen from the result of intermediate operations
Mill	The mill is essentially the processor of the machine into which the quantities about to be operated upon are brought

polynomials of degree 4- on 15-digit numbers, and the 3rd Scheutz Difference Engine is on display at the Science Museum in London.

It was the first machine to compute and print mathematical tables mechanically. The machine was accurate, and it showed the potential of mechanical machines as a tool for scientists and engineers.

The machine is unable to perform multiplication or division directly. Once the initial value of the polynomial and its derivative are calculated for some value of x, the difference engine may calculate any number of nearby values using the numerical method of finite differences. This method replaces computational intensive tasks involving multiplication or division, by an equivalent computation that just involves addition or subtraction.

The British government cancelled Babbage's project in 1842. He designed an improved difference engine No.2 (Fig. 2.4) in 1849. It could operate on seventh-order differences (i.e. polynomials of order 7) and 31-digit numbers. The machine consisted of 8 columns with each column consisting of 31 wheels. However, it was over 150 years later before it was built (in 1991) to mark the two hundredth anniversary of his birth. The Science Museum in London also built the printer that Babbage designed, and both the machine and the printer worked correctly according to Babbage's design (after a little debugging).

2.5 The Analytic Engine—Vision of a Computer

The Difference Engine was designed to produce mathematical tables, but it required human intervention to perform the calculations. Babbage recognized its limitations, and he proposed a revolutionary solution by outlining his vision of a mechanical computer. His plan was to construct a new machine that would be capable of executing all tasks that may be expressed in algebraic notation. His vision of such a computer (Analytic Engine) consisted of two parts (Table 2.2).

Babbage intended that the operation of the Analytic Engine would be analogous to the operation of the *Jacquard loom*.[3] The latter is capable of weaving (i.e. executing on the loom) a design pattern that has been prepared by a team of skilled

[3]The Jacquard loom was invented by Joseph Jacquard in 1801. It is a mechanical loom which used the holes in punch cards to control the weaving of patterns in a fabric. The use of punched cards allowed complex designs to be woven from the pattern defined on the punched cards. Each punched card corresponds to one row of the design, and the cards were appropriately ordered. It

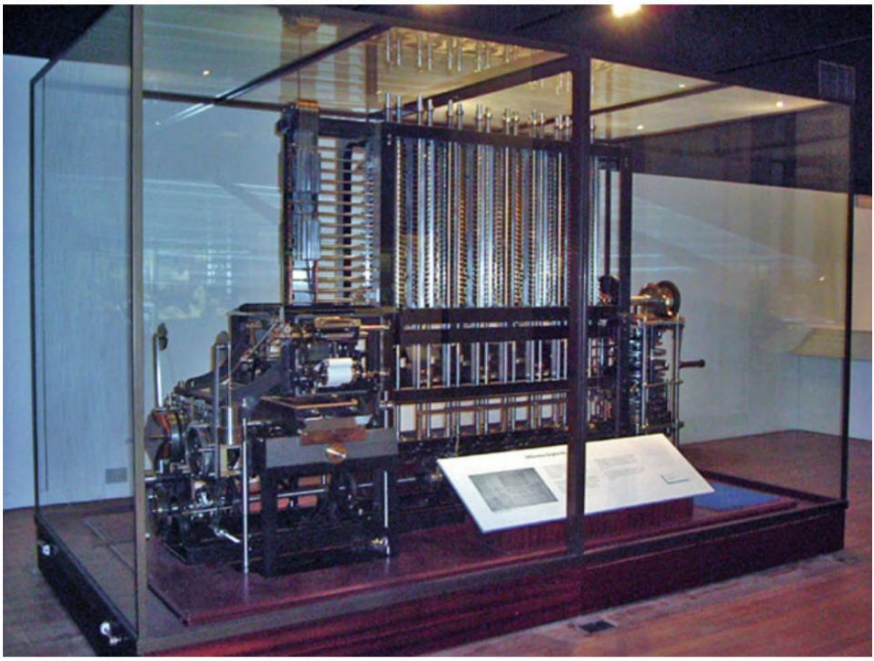


Fig. 2.4 Difference engine no. 2. Photo public domain

artists. The design pattern is represented by a set of cards with punched holes, where each card represents a row in the design. The cards are then ordered, placed in the loom and the loom produces the exact pattern.

The use of the punched cards in the Analytic Engine allowed the formulae to be manipulated in a manner dictated by the programmer. The cards commanded the analytic engine to perform various operations and to return a result. Babbage distinguished between two types of punched cards:

- *Operation Cards*
- *Variable Cards.*

Operation cards are used to define the operations to be performed, whereas the variable cards define the variables or data that the operations are performed upon. His planned use of punched cards to store programs in the Analytic Engine is similar to the idea of a stored computer program in Von Neumann architecture. However, Babbage's idea of using punched cards to represent machine instructions

was very easy to change the pattern of the fabric being weaved on the loom, as this simply involved changing cards.

Fig. 2.5 Lady Ada Lovelace

and data was over 100 years before digital computers. *Babbage's Analytic Engine is therefore an important milestone in the history of computing.*

Babbage intended that the program be stored on read-only memory using punch cards and that the input and output would be carried out using punch cards. He intended that the machine would be able to store numbers and intermediate results in memory that could then be processed. There would be several punch card readers in the machine for programs and data. He envisioned that the machine would be able to perform conditional jumps as well as parallel processing where several calculations could be performed at once.

The Analytic Engine was designed in 1834 as the world's first mechanical computer (Babbage and Menabrea 1842). It included a processor, memory and a way to input information and output results. However, the machine was never built, as Babbage was unable to secure funding from the British Government.

2.5.1 Applications of Analytic Engine

Lady Augusta Ada Lovelace (nee Byron)[4] (Fig. 2.5) was a mathematician who collaborated with Babbage on applications for the analytic engine. She is considered the world's first programmer, and the Ada programming language is named in her honour.

She was introduced to Babbage at a dinner party in 1833, and she visited Babbage's studio in London, where the prototype Difference Engine was on

[4]Lady Ada Lovelace was the daughter of the poet, Lord Byron.

display. She recognized the beauty of its invention, and she was fascinated by the idea of the analytic engine. She communicated regularly with Babbage with ideas on its applications.

Lovelace produced an annotated translation of Menabrea's '*Notions sur la machine analytique de Charles Babbage*' (Babbage and Menabrea 1842). She added copious notes to the translation,[5] which were about three times the length of the original memoir, and considered many of the difficult and abstract questions connected with the subject. These notes are regarded as a description of a computer and software.

She explained in the notes how the Analytic Engine could be programmed and wrote what is considered to be the first computer program. This program detailed a plan be written for how the engine would calculate *Bernoulli numbers*. Lady Ada Lovelace is therefore considered to be the first computer programmer, and Babbage called her the '*enchantress of numbers*'.

She saw the potential of the analytic engine to fields other than mathematics. She predicted that the machine could be used to compose music, produce graphics, as well as solving mathematical and scientific problems. She speculated that the machine might act on other things apart from numbers, and be able to manipulate symbols according to rules. In this way, a number could represent an entity other than a quantity.

2.6 Boole's Symbolic Logic

George Boole (Fig. 2.6) was born in Lincoln, England in 1815. His father (a cobbler who was interested in mathematics and optical instruments) taught him mathematics and showed him how to make optical instruments. Boole inherited his father's interest in knowledge, and he was self-taught in mathematics and Greek. He taught at various schools near Lincoln, and he developed his mathematical knowledge by working his way through Newton's Principia, as well as applying himself to the work of mathematicians such as Laplace and Lagrange.

He published regular papers from his early 20s, and these included contributions to probability theory, differential equations and finite differences. He developed Boolean algebra, which is the foundation for modern computing, and he is considered (along with Babbage) to be one of the grandfathers of computing. His work was theoretical, and he never actually built a computer or calculating machine. *However, Boole's symbolic logic was the perfect mathematical model for switching theory and for the design of digital circuits.*

Boole became interested in formulating a calculus of reasoning, and he published 'Mathematical Analysis of Logic' in 1847 (Boole 1848). This work developed novel ideas on a logical method, and he argued that logic should be considered

[5]There is some controversy as to whether this was entirely her own work or a joint effort by Lovelace and Babbage.

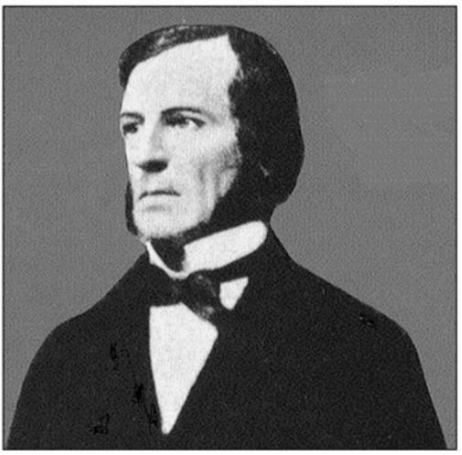


Fig. 2.6 George Boole

as a separate branch of mathematics, rather than as a part of philosophy. He argued that there are mathematical laws to express the operation of reasoning in the human mind, and he showed how Aristotle's syllogistic logic could be reduced to a set of algebraic equations. He corresponded regularly on logic with Augustus De Morgan.[6]

His paper on logic introduced two quantities '0' and '1'. He used the quantity 1 to represent the universe of thinkable objects (i.e. the universal set), and the quantity 0 represents the absence of any objects (i.e. the empty set). He then employed symbols, such as x, y, z, etc., to represent collections or classes of objects given by the meaning attached to adjectives and nouns. Next, he introduced three operators ($+$, $-$ and $\times$) that combined classes of objects.

The expression xy (i.e. x multiplied by y or $x \times y$) combines the two classes x, y to form the new class xy (i.e. the class whose objects satisfy the two meanings represented by class x *and* class y). Similarly, the expression $x + y$ combines the two classes x, y to form the new class $x + y$ (that satisfies either the meaning represented by class x *or* class y). The expression $x - y$ combines the two classes x, y to form the new class $x - y$. This represents the class (that satisfies the meaning represented by class x but not class y). The expression $(1 - x)$ represents objects that do not have the attribute that represents class x.

Thus, if x = black and y = sheep, then xy represents the class of black sheep. Similarly, $(1 - x)$ would represent the class obtained by the operation of selecting all things in the world except black things; $x\,(1 - y)$ represents the class of all things that are black but not sheep; and $(1 - x)\,(1 - y)$ would give us all things that are neither sheep nor black.

[6]De Morgan was a nineteenth-century British mathematician based at University College London. De Morgan's laws in Set Theory and Logic state that $(A \cup B)^c = A^c \cap B^c$ and $\neg (A \vee B) \equiv \neg A \wedge \neg B$.

He showed that these symbols obeyed a rich collection of algebraic laws and could be added, multiplied, etc., in a manner that is like real numbers. These symbols may be used to reduce propositions to equations, and algebraic rules may be employed to solve the equations. The rules include the following:

1.	$x + 0 = x$	(Additive identity)
2.	$x + (y + z) = (x + y) + z$	(Associative)
3.	$x + y = y + x$	(Commutative)
4.	$x + (1 - x) = 1$	
5.	$\times \cdot 1 = x$	(Multiplicative identity)
6.	$x \cdot 0 = 0$	
7.	$x + 1 = 1$	
8.	$xy = yx$	(Commutative)
9.	$x(yz) = (xy)z$	(Associative)
10.	$x(y + z) = xy + xz$	(Distributive)
11.	$x(y - z) = xy - xz$	(Distributive)
12.	$x^2 = x$	(Idempotent)

These operations are similar to the modern laws of set theory with the set union operation represented by '+', and the set intersection operation is represented by multiplication. The universal set is represented by '1' and the empty by '0'. The associative and distributive laws hold. Finally, the set complement operation is given by $(1 - x)$.

He applied the symbols to encode Aristotle's Syllogistic Logic, and he showed how the syllogisms could be reduced to equations. This allowed conclusions to be derived from premises by eliminating the middle term in the syllogism. He refined his ideas on logic further in '*An Investigation of the Laws of Thought*' which was published in 1854 (Boole 1958). This book aimed to identify the fundamental laws underlying reasoning in the human mind and to give expression to these laws in the symbolic language of a calculus.

He considered the equation $x^2 = x$ to be a fundamental law of thought. It allows the principle of contradiction to be expressed as (i.e. for an entity to possess an attribute and at the same time not to possess it)

$$\begin{aligned} & x^2 = x \\ \Rightarrow\ & x - x^2 = 0 \\ \Rightarrow\ & x(1 - x) = 0 \end{aligned}$$

For example, if x represents the class of horses then $(1 - x)$ represents the class of 'not-horses'. The product of two classes represents a class whose members are common to both classes. Hence, $x\,(1 - x)$ represents the class whose members are at once both horses and 'not-horses', and the equation $x\,(1 - x) = 0$ expresses that fact that there is no such class. That is, it is the empty set.

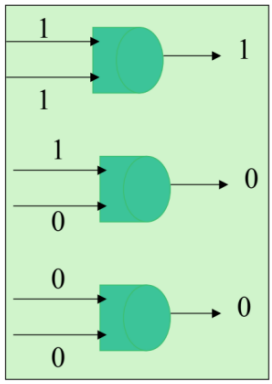


Fig. 2.7 Binary AND operation

Boole contributed to other areas in mathematics including differential equations, finite differences[7] and to the development of probability theory. Des McHale has written an interesting biography of Boole and Des McHale (1985). Boole's logic appeared to have no practical use, but this changed with Claude Shannon's (1937) Master's Thesis, which showed its applicability to switching theory and to the design of digital circuits.

2.6.1 Switching Circuits and Boolean Algebra

Claude Shannon's Master's Thesis showed that Boole's algebra provided the perfect mathematical model for switching theory and for the design of digital circuits. It may be employed to optimize the design of systems of electromechanical relays, and circuits with relays solve Boolean algebra problems. The use of the properties of electrical switches to process logic is the basic concept that underlies all modern electronic digital computers. Digital computers use the binary digits 0 and 1, and Boolean logical operations may be implemented by electronic AND, OR and NOT gates. More complex circuits (e.g. arithmetic) may be designed from these fundamental building blocks.

Modern electronic computers use billions of transistors that act as switches and can change state rapidly. A high voltage represents the binary value 1 with low voltage representing the binary value 0. A silicon chip may contain billions of tiny electronic switches arranged into logical gates. The basic logic gates are AND, OR and NOT. These gates may be combined in various ways to allow the computer to perform more complex tasks such as binary arithmetic. Each gate has binary value inputs and outputs.

The example in Fig. 2.7 is that of an 'AND' gate which produces the binary value 1 as output only if both inputs are 1. Otherwise, the result will be the binary

[7]Finite Differences are a numerical method used in solving differential equations.

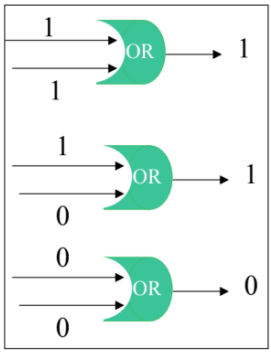


Fig. 2.8 Binary OR operation

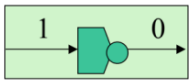


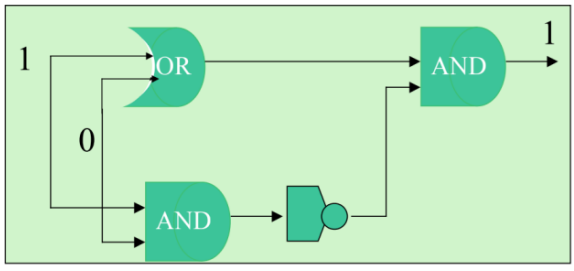
Fig. 2.9 NOT operation

value 0. Figure 2.8 shows an 'OR' gate which produces the binary value 1 as output if any of its inputs is 1. Otherwise, it will produce the binary value 0.

Finally, a NOT gate (Fig. 2.9) accepts only a single input which it inverts. That is, if the input is '1' the value '0' is produced and vice versa.

The logic gates may be combined to form more complex circuits. The example in Fig. 2.10 is that of a half adder of 1 + 0. The inputs to the top OR gate are 1 and 0 which yields the result of 1. The inputs to the bottom AND gate are 1 and 0 which yields the result 0, which is then inverted through the NOT gate to yield binary 1. Finally, the last AND gate receives two 1's as input and the binary value 1 is the result of the addition. The half adder computes the addition of two arbitrary binary digits, but it does not calculate the carry. It may be extended to a full adder that provides a carry for addition.

Fig. 2.10 Half adder



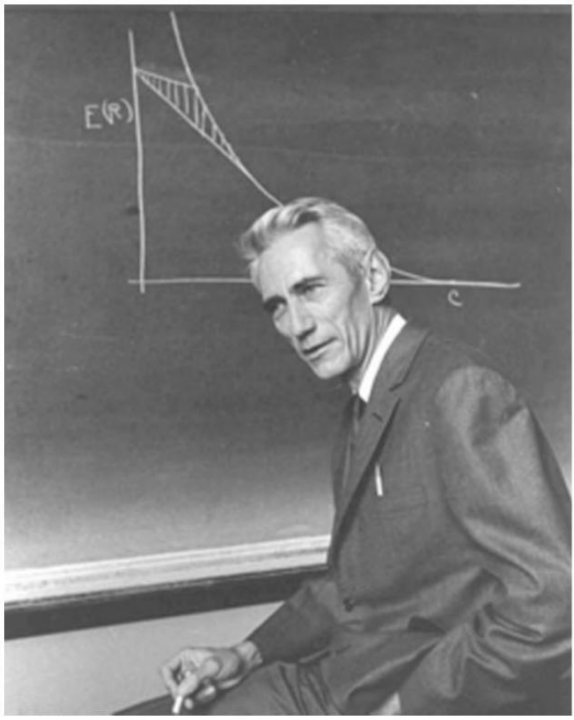


Fig. 2.11 Claude Shannon

2.7 Application of Symbolic Logic to Digital Computing

Claude Shannon (Fig. 2.11) was the first person[8] to see the applicability of Boole's algebra to simplify the design of circuits and telephone routing switches. He showed that Boole's symbolic logic was the perfect mathematical model for switching theory and for the subsequent design of digital circuits and computers.

His influential *Master's Thesis is a key milestone in computing*, and it shows how to lay out circuits according to Boolean principles. It provides the theoretical foundation of switching circuits, and *his insight of using the properties of electrical switches to do Boolean logic is the basic concept that underlies all electronic digital computers*.

Shannon realized that you could combine switches in circuits in such a manner as to carry out symbolic logic operations. This allowed binary arithmetic and more complex mathematical operations to be performed by relay circuits. He designed a circuit, which could add binary numbers, and he later designed circuits that could make comparisons and thus capable of performing a conditional statement. *This was the birth of digital logic and the digital computing age*.

[8]Victor Shestakov at Moscow State University also proposed a theory of electric switches based on Boolean algebra (published in Russian in 1941 whereas Shannon's were published in 1937).

Vannevar Bush (Regan 2013) was Shannon's supervisor at MIT, and Shannon's initial work was to improve Bush's mechanical computing device known as the Differential Analyser. This machine had a complicated control circuit that was composed of 100 switches that could be automatically opened and closed by an electromagnet. Shannon's insight was his realization that an electronic circuit is similar to Boolean algebra, and he showed how Boolean algebra could be employed to optimize the design of systems of electromechanical relays used in the analog computer. He also realized that circuits with relays could solve Boolean algebra problems.

His Master's thesis '*A Symbolic Analysis of Relay and Switching Circuits*' (Shannon 1937) showed that the binary digits (i.e. 0 and 1) can be represented by electrical switches. This allowed binary arithmetic and more complex mathematical operations to be performed by relay circuits, and provided electronics engineers with the mathematical tool that they needed to design digital electronic circuits and provided the foundation for the field.

The design of circuits and telephone routing switches could be simplified with Boole's symbolic algebra. Shannon showed how to lay out circuits according to Boolean principles, and his Master's thesis became the foundation for the practical design of digital circuits. These circuits are fundamental to the operation of modern computers and telecommunication systems, and his insight of using the properties of electrical switches to do Boolean logic is the basic concept that underlies all electronic digital computers.

2.8 Review Questions

1. Explain the significance of binary numbers in the computing field.
2. Explain the importance of Shannon's Master Thesis.
3. Explain the significance of the Analytic Engine.
4. Explain why Ada Lovelace is considered the world's first programmer.
5. Explain the significance of Boole to the computing field.
6. Explain the significance of Babbage to the computing field.
7. Explain the significance of Leibniz to the computing field.

2.9 Summary

This chapter considered foundational work done by Leibniz, Babbage, Boole, Ada Lovelace and Shannon. Leibniz developed a calculating machine (the Step Reckoner) that could perform the four basic arithmetic operations. He also invented the binary number system, which is used extensively in the computer field.

Babbage did pioneering work on calculating machines. He designed the Difference Engine (a sophisticated calculator that could be used to produce mathematical tables), and he also designed the Analytic Engine (the world's first mechanical computer).

Lady Ada Lovelace was introduced to Babbage's ideas on the analytic engine, and she predicted that such a machine could be used to compose music, produce graphics, as well as solving mathematical and scientific problems.

Boole was a nineteenth-century English mathematician who made important contributions to mathematics, and his symbolic logic provides the foundation for digital computers.

Shannon was a twentieth-century American mathematician and engineer, and he showed that Boole's symbolic logic provided the perfect mathematical model for switching theory and for the subsequent design of digital circuits and computer.

References

Babbage C, Menabrea LF (1842) Sketch of the analytic engine invented by charles babbage. Bibliothèque Universelle de Genève, October, No. 82 Translated by Ada, Augusta, Countess of Lovelace

Boole G (1848) The calculus of logic. Camb Dublin Math J III:183–98

Boole G (1958) An investigation into the laws of thought. Dover Publications (First published in 1854)

Leibniz WG (1703) Explication de l'Arithmétique Binaire. Memoires de l'Academie Royale des Sciences

McHale D (1985) George Boole, His Life and Work. Cork University Press

O'Regan G (2013) Giants of computing. Springer

Shannon C (1937) A symbolic analysis of relay and switching circuits. Masters thesis, Massachusetts Institute of Technology

Overview of Mathematics in Computing

3

Key Topics

Sets
Set operations
Russell's paradox
Computer representation of sets
Relations
Composition of relations
Functions
Partial and total functions
Functional programming
Number theory
Automata theory
Graph theory

3.1 Introduction

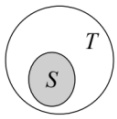
This chapter introduces essential mathematics for computing and discusses fundamental concept such as sets, relations and functions. Sets are collections of well-defined objects; relations indicate relationships between members of two sets

G. O'Regan, *Mathematics in Computing*, Undergraduate Topics in Computer Science, https://doi.org/10.1007/978-3-030-34209-8_3

In this example, $even(x)$ is a predicate that is true if x is even and false otherwise. In general, $A = \{x \in E \mid P(x)\}$ denotes a set A formed from a set E using the predicate P to restrict membership of A to those elements of E for which the predicate is true.

The elements of a finite set S are denoted by $\{x_1, x_2, \ldots x_n\}$. The expression $x \in S$ denotes that the element x is a member of the set S, whereas the expression $x \notin S$ indicates that x is not a member of the set S.

A set S is a subset of a set T (denoted $S \subseteq T$) if whenever $s \in S$ then $s \in T$, and in this case the set T is said to be a superset of S (denoted $T \supseteq S$). Two sets S and T are said to be equal if they contain identical elements, i.e. $S = T$ if and only if $S \subseteq T$ and $T \subseteq S$. A set S is a proper subset of a set T (denoted $S \subset T$) if $S \subseteq T$ and $S \neq T$. That is, every element of S is an element of T and there is at least one element in T that is not an element of S. In this case, T is a proper superset of S (denoted $T \supset S$).



The empty set (denoted by $\varnothing$ or $\{\}$) represents the set that has no elements. Clearly, $\varnothing$ is a subset of every set. The singleton set containing just one element x is denoted by $\{x\}$, and clearly $x \in \{x\}$ and $x \neq \{x\}$. Clearly, $y \in \{x\}$ if and only if $x = y$.

Example 3.2

(i) $\{1, 2\} \subseteq \{1, 2, 3\}$.
(ii) $\varnothing \subset \mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subset \mathbb{C}$.

The cardinality (or size) of a finite set S defines the number of elements present in the set. It is denoted by $|S|$. The cardinality of an infinite[4] set S is written as $|S| = \infty$.

Example 3.3

(i) Given $A = \{2, 4, 5, 8, 10\}$ then $|A| = 5$.
(ii) Given $A = \{x \in \mathbb{Z} : x^2 = 9\}$ then $|A| = 2$.
(iii) Given $A = \{x \in \mathbb{Z} : x^2 = -9\}$ then $|A| = 0$.

[4]The natural numbers, integers and rational numbers are countable sets (i.e. they may be put into a one-to-one correspondence with the Natural numbers), whereas the real and complex numbers are uncountable sets.

3.2.1 Set-Theoretical Operations

Several set-theoretical operations are considered in this section. These include the Cartesian product operation, the power set of a set, the set union operation, the set intersection operation, the set difference operation and the symmetric difference operation.

Cartesian Product

The Cartesian product allows a new set to be created from existing sets. The Cartesian[5] product of two sets S and T (denoted $S \times T$) is the set of ordered pairs $\{(s, t) \mid s \in S, t \in T\}$. Clearly, $S \times T \neq T \times S$ and so the Cartesian product of two sets is not commutative. Two ordered pairs (s_1, t_1) and (s_2, t_2) are considered equal if and only if $s_1 = s_2$ and $t_1 = t_2$.

The Cartesian product may be extended to that of n sets $S_1, S_2, \ldots, S_n$. The Cartesian product $S_1 \times S_2 \times \cdots \times S_n$ is the set of ordered n-tuples $\{(s_1, s_2, \ldots, s_n) \mid s_1 \in S_1, s_2 \in S_2, \ldots, s_n \in S_n\}$. Two ordered n-tuples $(s_1, s_2, \ldots, s_n)$ and $(s_1', s_2', \ldots, s_n')$ are considered equal if and only if $s_1 = s_1', s_2, = s_2', \ldots, s_n = s_n'$.

The Cartesian product may also be applied to a single set S to create ordered n-tuples of S, i.e. $S^n = S \times S \times \cdots \times S$ (n times).

Power Set

The power set of a set A (denoted $\mathbb{P}A$) denotes the set of subsets of A. For example, the power set of the set A = {1, 2, 3} has eight elements and is given by

$$\mathbb{P}A = \{\varnothing, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}.$$

There are $2^3 = 8$ elements in the power set of $A = \{1, 2, 3\}$ where the cardinality of A is 3. In general, there are $2^{|A|}$ elements in the power set of A.

Theorem 3.1 (Cardinality of Power Set of A) *There are $2^{|A|}$ elements in the power set of A.*

Proof Let $|A| = n$, then the cardinalities of the subsets of A are subsets of size 0, 1, …, n. There are $\binom{n}{k}$ subsets of A of size k.[6] Therefore, the total number of subsets of A is the total number of subsets of size 0, 1, 2, … up to n. That is,

$$|\mathbb{P}A| = \sum_{k=0}^{n} \binom{n}{k}$$

The Binomial Theorem states that

[5]Cartesian product is named after René Descartes who was a famous seventeenth-century French mathematician and philosopher. He invented the Cartesian coordinates system that links geometry and algebra and allows geometric shapes to be defined by algebraic equations.
[6]Permutations and combinations are discussed in Chap. 8.

$$(1+x)^n = \sum_{k=0}^{n} \binom{n}{k} x^k$$

Therefore, putting $x = 1$ we get that

$$2^n = (1+1)^n = \sum_{k=0}^{n} \binom{n}{k} 1^k = |\mathbb{P}\mathrm{A}|$$

Union and Intersection Operations

The union of two sets A and B is denoted by $A \cup B$. It results in a set that contains all of the members of A and of B and is defined by

$$\mathrm{A} \cup \mathrm{B} = \{r | r \in A \, or \, r \in B\}.$$

For example, suppose $A = \{1, 2, 3\}$ and $B = \{2, 3, 4\}$ then $A \cup B = \{1, 2, 3, 4\}$. Set union is a commutative operation, i.e. $A \cup B = B \cup A$. Venn Diagrams are used to illustrate these operations pictorially.

$A \cup B$ $A \cap B$

The intersection of two sets A and B is denoted by $A \cap B$. It results in a set containing the elements that A and B have in common and is defined by

$$A \cap B = \{r | r \in A \text{ and } r \in B\}.$$

Suppose $A = \{1, 2, 3\}$ and $B = \{2, 3, 4\}$ then $A \cap B = \{2, 3\}$. Set intersection is a commutative operation, i.e. $A \cap B = B \cap A$.

Union and intersection may be extended to more generalized union and intersection operations. For example,

$\cup_{i=1}^{n} A_i$ denotes the union of n sets.
$\cap_{i=1}^{n} A_i$ denotes the intersection of n sets.

Set Difference Operations

The set difference operation $A \backslash B$ yields the elements in A that are not in B. It is defined by

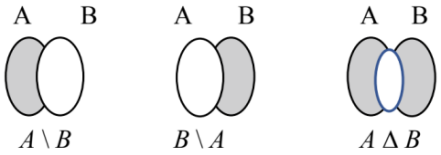
$$A \backslash B = \{a | a \in A \text{ and } a \notin B\}$$

For A and B defined as A = {1, 2} and B = {2, 3}, we have $A\backslash B$ = {1} and $B\backslash A$ = {3}. Clearly, set difference is not commutative, i.e. $A\backslash B \neq B\backslash A$. Clearly, $A\backslash A = \varnothing$ and $A\backslash\varnothing = A$.

The symmetric difference of two sets A and B is denoted by $A \Delta B$ and is given by

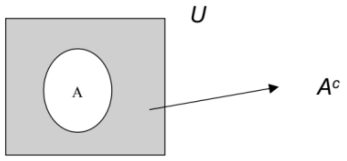
$$A \Delta B = A\backslash B \cup B\backslash A$$

The symmetric difference operation is commutative, i.e. $A \Delta B = B \Delta A$. Venn diagrams are used to illustrate these operations pictorially.



The complement of a set A (with respect to the universal set U) is the elements in the universal set that are not in A. It is denoted by A^c (or A') and is defined as

$$A^c = \{u | u \in U \text{ and } u \notin A\} = U\backslash A$$



The complement of the set A is illustrated by the shaded area.

3.2.2 Properties of Set-Theoretical Operations

The set union and set intersection properties are commutative and associative. Their properties are listed in Table 3.1.

These properties may be seen to be true with Venn diagrams, and we give a proof of the distributive property (this proof uses logic which is discussed in Chaps. 15–17).

Proof of Properties (*Distributive Property*) To show $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

Table 3.1 Properties of set operations

Property	Description
Commutative	Union and intersection operations are commutative, i.e. $S \cup T = T \cup S$ $S \cap T = T \cap S$
Associative	Union and intersection operations are associative, i.e. $R \cup (S \cup T) = (R \cup S) \cup T$ $R \cap (S \cap T) = (R \cap S) \cap T$
Identity	The identity under set union is the empty set $\varnothing$, and the identity under intersection is the universal set U $S \cup \varnothing = \varnothing \cup S = S$ $S \cap U = U \cap S = S$
Distributive	The union operator distributes over the intersection operator and vice versa $R \cap (S \cup T) = (R \cap S) \cup (R \cap T)$ $R \cup (S \cap T) = (R \cup S) \cap (R \cup T)$
De Morgan's (De Morgan's law is named after Augustus De Morgan, a nineteenth-century English mathematician who was a contemporary of George Boole) law	The complement of $S \cup T$ is given by $(S \cup T)^c = S \cap T$ The complement of $S \cap T$ is given by $(S \cap T)^c = S^c \cup T$

Suppose

$$
\begin{aligned}
& x \in A \cap (B \cup C) \text{ then} \\
& \quad x \in A \wedge x \in (B \cup C) \\
& \Rightarrow x \in A \wedge (x \in B \vee x \in C) \\
& \Rightarrow (x \in A \wedge x \in B) \vee (x \in A \wedge x \in C) \\
& \Rightarrow x \in (A \cap B) \vee x \in (A \cap C) \\
& \Rightarrow x \in (A \cap B) \cup (A \cap C)
\end{aligned}
$$

Therefore, $A \cap (B \cup C) \subseteq (A \cap B) \cup (A \cap C)$
Similarly $(A \cap B) \cup (A \cap C) \subseteq A \cap (B \cup C)$
Therefore, $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

3.2.3 Russell's Paradox

Bertrand Russell (Fig. 3.1) was a famous British logician, mathematician and philosopher. He was the co-author with Alfred Whitehead of *Principia Mathematica*, which aimed to derive all the truths of mathematics from logic. Russell's Paradox was discovered by Bertrand Russell in 1901 and showed that the system of

3.3 Relations

A binary relation $R(A, B)$ where A and B are sets is a subset of $A \times B$, *i.e.* $R \subseteq A \times B$. The domain of the relation is A, and the co-domain of the relation is B. The notation aRb signifies that $(a, b) \in R$.

A binary relation $R(A, A)$ is a relation between A and A (or a relation on A). This type of relation may always be composed with itself, and its inverse is also a binary relation on A. The identity relation on A is defined by $a\ i_A a$ for all $a \in A$.

Example 3.4 There are many examples of relations:

(i) The relation on a set of students in a class where $(a, b) \in R$ if the height of a is greater than the height of b.

(ii) The relation between A and B where A = {0, 1, 2} and B = {3, 4, 5} with R given by

$$R = \{(0,3), (0,4), (1,4)\}$$

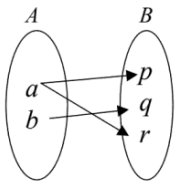
(iii) The relation less than (<) between and $\mathbb{R}$ and $\mathbb{R}$ is given by

$$\{(x, y) \in \mathbb{R}^2 : x < y\}$$

(iv) A bank may represent the relationship between the set of accounts and the set of customers by a relation. The implementation of a bank account may be a positive integer with at most eight decimal digits.
The relationship between accounts and customers may be done with a relation $R \subseteq A \times B$, with the set A chosen to be the set of natural numbers, and the set B chosen to be the set of all human beings alive or dead. The set

$$A \text{ could also be chosen to be } A = \{n \in \mathbb{N} : n < 10^8\}$$

A relation $R(A, B)$ may be represented pictorially. This is referred to as the graph of the relation, and it is illustrated in the diagram below. An arrow from x to y is drawn if (x, y) is in the relation. Thus, for the height relation R given by $\{(a, p), (a, r), (b, q)\}$ an arrow is drawn from a to p, from a to r and from b to q to indicate that (a, p), (a, r) and (b, q) are in the relation R.



The pictorial representation of the relation makes it easy to see that the height of a is greater than the height of p and r; and that the height of b is greater than the height of q.

An n-ary relation $R\ (A_1, A_2, \ldots A_n)$ is a subset of $(A_1 \times A_2 \times \cdots \times A_n)$. However, an n-ary relation may also be regarded as a binary relation $R(A, B)$ with $\mathrm{A} = A_1 \times A_2 \times \cdots \times A_{n-1}$ and $B = A_n$.

3.3.1 Reflexive, Symmetric and Transitive Relations

(i) A binary relation on A may have additional properties such as being reflexive, symmetric or transitive. These properties are defined as a relation on a set A is *reflexive* if $(a, a) \in R$ for all $a \in A$.
(ii) A relation R is *symmetric* if whenever $(a, b) \in \mathrm{R}$ then $(b, a) \in R$.
(iii) A relation is *transitive* if whenever $(a, b) \in R$ and $(b, c) \in \mathrm{R}$ then $(a, c) \in R$.

A relation that is reflexive, symmetric and transitive is termed an *equivalence relation*.

Example 3.5 (*Reflexive Relation*) A relation is reflexive if each element possesses an edge looping around on itself. The relation in Fig. 3.2 is reflexive.

Example 3.6 (*Symmetric Relation*) The graph of a symmetric relation will show for every arrow from a to b an opposite arrow from b to a. The relation in Fig. 3.3 is symmetric, i.e. whenever $(a, b) \in R$ then $(b, a) \in R$.

Example 3.7 (*Transitive relation*) The graph of a transitive relation will show that whenever there is an arrow from a to b and an arrow from b to c that there is an arrow from a to c. The relation in Fig. 3.4 is transitive, i.e. whenever $(a, b) \in R$ and $(b, c) \in R$ then $(a, c) \in R$.

Example 3.8 (*Equivalence relation*) The relation on the set of integers $\mathbb{Z}$ defined by $(a, b) \in R$ if $a - b = 2k$ for some $k \in \mathbb{Z}$ is an equivalence relation, and it partitions the set of integers into two equivalence classes, i.e. the even and odd integers.

Domain and Range of Relation

The domain of a relation $R\ (A, B)$ is given by $\{a \in A \mid \exists b \in B \text{ and } (a, b) \in R\}$. It is denoted by **dom** R. The domain of the relation $\mathrm{R} = \{(a, p), (a, r), (b, q)\}$ is $\{a, b\}$.

The range of a relation $R\ (A, B)$ is given by $\{b \in \mathrm{B} \mid \exists a \in A \text{ and } (a, b) \in \mathrm{R}\}$. It is denoted by **rng** R. The range of the relation $R = \{(a, p), (a, r), (b, q)\}$ is $\{p, q, r\}$.

Fig. 3.2 Reflexive relation

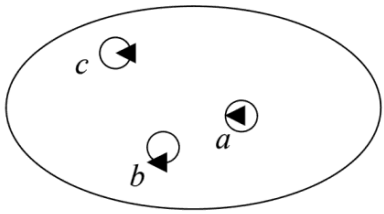


Fig. 3.3 Symmetric relation

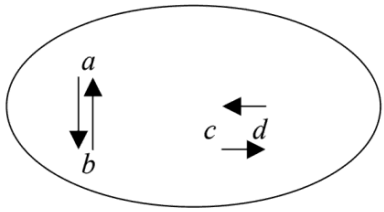
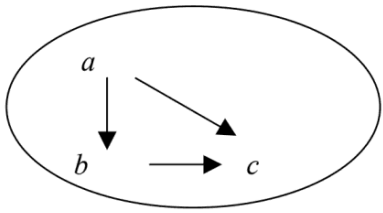


Fig. 3.4 Transitive relation



Inverse of a Relation

Suppose $R \subseteq A \times B$ is a relation between A and B then the inverse relation $R^{-1} \subseteq B \times A$ is defined as the relation between B and A and is given by

$$b\,R^{-1}\,a \text{ if and only if } a\,R\,b$$

That is,

$$R^{-1} = \{(b, a) \in B \times A : (a, b) \in R\}$$

Example 3.9 Let R be the relation between $\mathbb{Z}$ and $\mathbb{Z}^+$ defined by *mRn* if and only if $m^2 = n$. Then $R = \{(m, n) \in \mathbb{Z} \times \mathbb{Z}^+ : m^2 = n\}$ and $R^{-1} = \{(n, m) \in \mathbb{Z}^+ \times \mathbb{Z} : m^2 = n\}$.

For example, $-3\ R\ 9$, $-4\ R\ 16$, $0\ R\ 0$, $16\ R^{-1}\ -4$, $9\ R^{-1}\ -3$, etc.

Partitions and Equivalence Relations

An equivalence relation on A leads to a partition of A, and vice versa for every partition of A there is a corresponding equivalence relation.

Let A be a finite set and let $A_1, A_2, \ldots, A_n$ be subsets of $A_{i \neq \varnothing \text{ for all } i, A_i} \cap A_j = \varnothing$ if $i \neq j$ and $A = \cup_i^n A_i = A_1 \cup A_2 \cup \ldots \cup A_n$.

The sets A_i partition the set A, and these sets are called the classes of the partition (Fig. 3.5).

Theorem 3.2 (Equivalence Relation and Partitions) *An equivalence relation on A gives rise to a partition of A where the equivalence classes are given by Class (a) = {x | x ∈ A and (a, x) ∈ R}. Similarly, a partition gives rise to an equivalence relation R, where (a, b) ∈ R if and only if a and b are in the same partition.*

Proof Clearly, $a \in \text{Class}(a)$ since R is reflexive and clearly the union of the equivalence classes is A. Next, we show that two equivalence classes are either equal or disjoint.

Suppose $\text{Class}(a) \cap \text{Class}(b) \neq \varnothing$. Let $x \in \text{Class}(a) \cap \text{Class}(b)$ and so (a, x) and $(b, x) \in$ R. By the symmetric property $(x, b) \in$ R and since R is transitive from (a, x) and (x, b) in R we deduce that $(a, b) \in$ R. Therefore $b \in \text{Class}(a)$. Suppose y is an arbitrary member of Class (b) then $(b, y) \in$ R; therefore, from (a, b) and (b, y) in R we deduce that (a, y) is in R. Therefore, since y was an arbitrary member of Class (a) we deduce that $\text{Class}(b) \subseteq \text{Class}(a)$. Similarly, $\text{Class}(a) \subseteq \text{Class}(b)$ and so Class $(a) = \text{Class}(b)$.

This proves the first part of the theorem and for the second part we define a relation R such that $(a, b) \in$ R if a and b are in the same partition. It is clear that this is an equivalence relation.

3.3.2 Composition of Relations

The composition of two relations $R_1(A, B)$ and $R_2(B, C)$ is given by R_2 o R_1 where $(a, c) \in R_2$ o R_1 if and only there exists $b \in$ B such that $(a, b) \in R_1$ and $(b, c) \in R_2$. The composition of relations is associative, i.e.

$$(R_3 \circ R_2) \circ R_1 = R_3 \circ (R_2 \circ R_1)$$

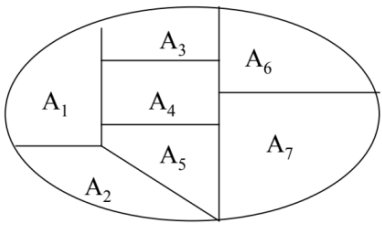


Fig. 3.5 Partitions of A

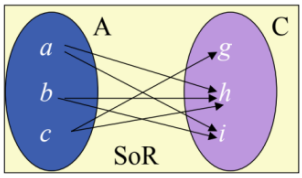
Example 3.10 (*Composition of Relations*) Consider a library that maintains two files. The first file maintains the serial number s of each book as well as the details of the author a of the book. This may be represented by the relation $R_1 = sR_1a$. The second file maintains the library card number c of its borrowers and the serial number s of any books that they have borrowed. This may be represented by the relation $R_2 = c\ R_2s$.

The library wishes to issue a reminder to its borrowers of the authors of all books currently on loan to them. This may be determined by the composition of R_1 o R_2, i.e. $c\ R_1$o $R_2\ a$ if there is book with serial number s such that $c\ R_2\ s$ and $s\ R_1\ a$.

Example 3.11 (*Composition of Relations*) Consider sets $A = \{a, b, c\}$, $B = \{d, e, f\}$, $C = \{g, h, i\}$ and relations $R(A, B) = \{(a, d), (a, f), (b, d), (c, e)\}$ and $S(B, C) = \{(d, h), (d, i), (e, g), (e, h)\}$. Then, we graph these relations and show how to determine the composition pictorially.

S o R is determined by choosing $x \in A$ and $y \in C$ and checking if there is a route from x to y in the graph. If so, we join x to y in S o R. For example, if we consider a and h we see that there is a path from a to d and from d to h and therefore (a, h) is in the composition of S and R (Fig. 3.6).

The union of two relations $R_1(A, B)$ and $R_2(A, B)$ is meaningful (as these are both subsets of $A \times B$). The union $R_1 \cup R_2$ is defined as $(a, b) \in R_1 \cup R_2$ if and only if $(a, b) \in R_1$ or $(a, b) \in R_2$.



Similarly, the intersection of R_1 and R_2 ($R_1 \cap R_2$) is meaningful and is defined as $(a, b) \in R_1 \cap R_2$ if and only if $(a, b) \in R_1$ and $(a, b) \in R_2$. The relation R_1 is a subset of R_2 ($R_1 \subseteq R_2$) if whenever $(a, b) \in R_1$ then $(a, b) \in R_2$.

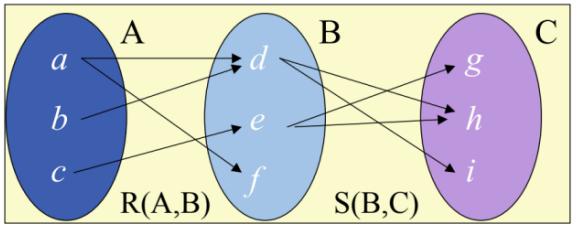


Fig. 3.6 Composition of relations S o R

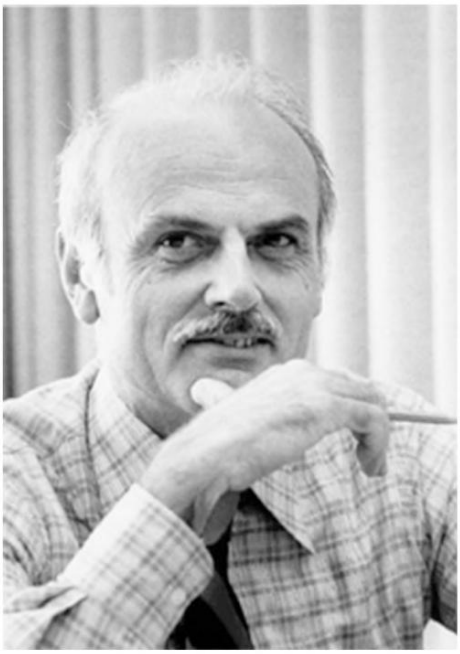


Fig. 3.7 Edgar Codd

The basic relational building block is the domain or data type (often called just type). Each row of the table represents one n-tuple (one tuple) of the relation, and the number of tuples in the relation is the cardinality of the relation. Consider the PART relation taken from Date (1981), where this relation consists of a heading and the body. There are five data types representing part numbers, part names, part colours, part weights and locations where the parts are stored. The body consists of a set of n-tuples, and the PART relation in Fig. 3.8 is of cardinality six.

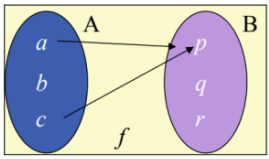
There is more detailed information on the relational model and databases in O'Regan (2018).

P#	PName	Colour	Weight	City
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

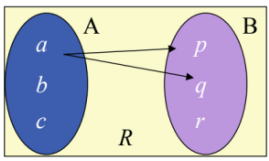
Fig. 3.8 PART relation

3.4 Functions

A function $f : A \rightarrow B$ is a special relation such that for each element $a \in$ A there is exactly (or at most)[8] one element $b \in$ B. This is written as $f(a) = b$.



A function is a relation but not every relation is a function. For example, the relation in the diagram below is not a function since there are two arrows from the element $a \in A$.



The domain of the function (denoted by **dom** f) is the set of values in A for which the function is defined. The domain of the function is A if f is a total function. The co-domain of the function is B. The range of the function (denoted **rng** f) is a subset of the co-domain and consists of

$$\mathbf{rng}\, f = \{r | r \in B \,\text{such that}\, f(a) = r \,\text{for some}\, a \in A\}.$$

Functions may be partial or total. A *partial function* (or partial mapping) may be undefined for some values of A, and partial functions arise regularly in the computing field (Fig. 3.9). *Total functions* are defined for every value in A, and many functions encountered in mathematics are total.

Example 3.13 (*Functions*) Functions are an essential part of mathematics and computer science, and there are many well-known functions such as the trigonometric functions $\sin(x)$, $\cos(x)$ and $\tan(x)$; the logarithmic function $\ln(x)$; the exponential functions e^x; and polynomial functions.

(i) Consider the partial function $f : \mathbb{R} \rightarrow \mathbb{R}$ $f(x) = {}^1/x$ (where $x \neq 0$).

Then, this partial function is defined everywhere except for $x = 0$.

(ii) Consider the function $f : \mathbb{R} \rightarrow \mathbb{R}$ where $f(x) = x^2$

Then this function is defined for all $x \in \mathbb{R}$

[8]We distinguish between total and partial functions. A total function is defined for all elements in the domain, whereas a partial function may be undefined for one or more elements in the domain.

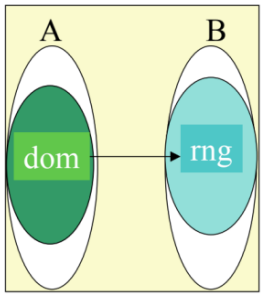


Fig. 3.9 Domain and range of a partial function

Partial functions often arise in computing as a program may be undefined or fail to terminate for several values of its arguments (e.g. infinite loops). Care is required to ensure that the partial function is defined for the argument to which it is to be applied.

Consider a program P that has one natural number as its input and which fails to terminate for some input values. It prints a single real result and halts if it terminates. Then P can be regarded as a partial mapping from $\mathbb{N}$ to $\mathbb{R}$.

$$P : \mathbb{N} \rightarrow \mathbb{R}$$

Example 3.14 How many total functions $f : A \rightarrow B$ are there from A to B (where A and B are finite sets)?

Each element of A maps to any element of B, i.e. there are $|\mathrm{B}|$ choices for each element $a \in A$. Since there are $|\mathrm{A}|$ elements in A the number of total functions is given by

$$\begin{aligned} &|\mathrm{B}|\,|\mathrm{B}|\ldots|\mathrm{B}| \quad (|\mathrm{A}|\text{times}) \\ &= |\mathrm{B}|^{|\mathrm{A}|} \qquad \text{total functions between A and B.} \end{aligned}$$

Example 3.15 How many partial functions $f : A \rightarrow B$ are there from A to B (where A and B are finite sets)?

Each element of A may map to any element of B or to no element of B (as it may be undefined for that element of A). In other words, there are $|\mathrm{B}| + 1$ choices for each element of A. As there are $|\mathrm{A}|$ elements in A, the number of distinct partial functions between A and B is given by

$$\begin{aligned} &(|\mathrm{B}|+1)(|\mathrm{B}|+1)\ldots(|\mathrm{B}|+1) \quad (|\mathrm{A}|\text{times}) \\ &= (|\mathrm{B}|+1)^{|\mathrm{A}|} \end{aligned}$$

Two partial functions f and g are equal if

1. dom f = dom g
2. $f(a) = g(a)$ for all $a \in$ dom f.

A function f is less defined than a function g ($f \subseteq$ g) if the domain of f is a subset of the domain of g, and the functions agree for every value on the domain of f.

1. dom $f \subseteq$ dom g
2. $f(a) = g(a)$ for all $a \in$ dom f.

The composition of functions is similar to the composition of relations. Suppose f: A → B and g : B → C then $g \; o \; f$: A → C is a function, and it is written as $g \; o \; f$ (x) or $g(f(x))$ for $x \in$ A.

The composition of functions is not commutative and this can be seen by an example. Consider the function $f : \mathbb{R} \to \mathbb{R}$ such that $f(x) = x^2$ and the function $g : \mathbb{R} \to \mathbb{R}$ such that $g(x) = x + 2$. Then

$$g \, o f(x) = g\left(x^2\right) = x^2 + 2.$$
$$f \, o \, g(x) = f(x+2) = (x+2)^2 = x^2 + 4x + 4.$$

Clearly, $g \; o \; f(x) \neq f \; o \; g(x)$ and so composition of functions is not commutative. The composition of functions is associative, as the composition of relations is associative and every function is a relation. For f: A → B, g : B → C, and h : C → D we have

$$h \, o \, (g \, o f) = (h \, o \, g) \, o f$$

A function f : A →B is *injective* (*one to one*) if

$$f(a_1) = f(a_2) \Rightarrow a_1 = a_2.$$

For example, consider the function $f : \mathbb{R} \to \mathbb{R}$ with $f\ (x) = x^2$. Then $f\ (3) = f\ (-3) = 9$ and so this function is not one to one.

A function f: A →B is *surjective* (*onto*) if given any $b \in$ B there exists an $a \in$ A such that $f(a) = b$. Consider the function $f : \mathbb{R} \to \mathbb{R}$ with $f(x) = x + 1$. Clearly, given any $r \in \mathbb{R}$ then $f\ (r - 1) = r$ and so f is onto (Fig. 3.10).

A function is *bijective* if it is one to one and onto (Fig. 3.11). That is, there is a one-to-one correspondence between the elements in A and B, and for each $b \in B$ there is a unique $a \in A$ such that $f(a) = b$.

The inverse of a relation was discussed earlier, and the relational inverse of a function f: A →B clearly exists. The relational inverse of the function may or may not be a function.

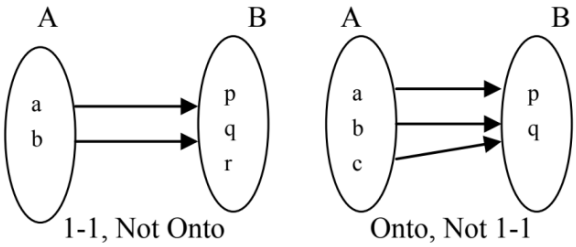


Fig. 3.10 Injective and surjective functions

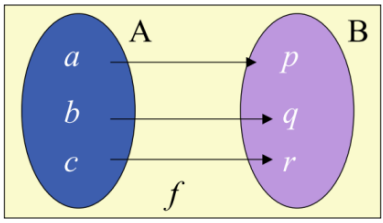


Fig. 3.11 Bijective function (one to one and onto)

However, if the relational inverse is a function it is denoted by $f^{-1} : \mathrm{B} \rightarrow \mathrm{A}$. A total function has an inverse if and only if it is bijective whereas a partial function has an inverse if and only if it is injective.

The identity function $1_{\mathrm{A}} : \mathrm{A} \rightarrow \mathrm{A}$ is a function such that $1_{\mathrm{A}}(a) = a$ for all $a \in \mathrm{A}$. Clearly, when the inverse of the function exists then we have that $f^{-1}\ o\ f = 1_{\mathrm{A}}$ and $f^{-}\ o\ f^{-1} = 1_{\mathrm{B}}$.

Theorem 3.3 (Inverse of Function) *A total function has an inverse if and only if it is bijective.*

Proof Suppose $f : A \rightarrow B$ has an inverse f^{-1}. Then we show that f is bijective.

We first show that f is one to one. Suppose $f(x_1) = f(x_2)$ then

$$\begin{aligned} & f^{-1}(f(x_1)) = f^{-1}(f(x_2)) \\ & \Rightarrow f^{-1}\ of\ (x_1) = f^{-1}\ of\ (x_2) \\ & \Rightarrow 1_{\mathrm{A}}(x_1) = 1_{\mathrm{A}}(x_2) \\ & \Rightarrow x_1 = x_2 \end{aligned}$$

$$
\begin{aligned}
z &= \mathrm{sqr}\, p / \mathrm{sqr}\, q \\
\mathrm{sqr}\, k &= k * k \\
p &= a + b \\
q &= a - b \\
a &= 10 \\
b &= 5
\end{aligned}
$$

The scope of a formal parameter (e.g. the parameter k above in the function sqr) is limited to the definition of the function in which it occurs.

One of the most common data structures used in Miranda is the list. The empty list is denoted by [ ], and an example of a list of integers is given by [1, 3, 4, 8]. Lists may be appended to by using the '++' operator. For example,

$$[1,3,5] + + [2,4] \text{ is } [1,3,5,2,4].$$

The length of a list is given by the '#' operator:

$$\#[1,3] = 2$$

The infix operator ':' is employed to prefix an element to the front of a list. For example,

$$5 : [2,4,6] \text{ is equal to } [5,2,4,6]$$

The subscript operator '!' is employed for subscripting: For example,

$$\text{Nums} = [5,2,4,6] \quad \text{then Nums!0 is } 5.$$

The elements of a list are required to be of the same type. A sequence of elements that contains mixed types is called a tuple. A tuple is written as follows:

$$\text{Employee} = (\text{``Holmes''}, \text{``221B Baker St.London''}, 50, \text{``Detective''})$$

A tuple is similar to a record in Pascal, whereas lists are similar to arrays. Tuples cannot be subscripted but their elements may be extracted by pattern matching. Pattern matching is illustrated by the well-known example of the factorial function:

$$
\begin{aligned}
&\mathrm{fac}\, 0 = 1 \\
&\mathrm{fac}(n+1) = (n+1) * \mathrm{fac}\ n
\end{aligned}
$$

The definition of the factorial function uses two equations, distinguished by using different patterns in the formal parameters. Another example of pattern matching is the reverse function on lists:

$$\text{reverse}\,[\,] = [\,]$$
$$\text{reverse}\,(a : x) = \text{reverse}\,x + + [a]$$

Miranda is a higher order language, and it allows functions to be passed as parameters and returned as results. Currying is allowed and this allows a function of n-arguments to be treated as n applications of a function with 1-argument. Function application is left associative, i.e. f x y means (f x) y. That is, the result of applying the function f to x is a function, and this function is then applied to y. Every function with two or more arguments in Miranda is a higher order function.

3.6 Number Theory

Number theory is the branch of mathematics that is concerned with the mathematical properties of the natural numbers and integers. These include properties such as the parity of a number, divisibility, additive and multiplicative properties, whether a number is prime or composite, the prime factors of a number, the greatest common divisor and least common multiple of two numbers, and so on.

Number theory has many applications in computing including cryptography and coding theory. For example, the RSA public-key cryptographic system relies on its security due to the infeasibility of the integer factorization problem for large numbers.

There are several unsolved problems in number theory and especially in prime number theory. For example, Goldbach's[14] Conjecture states that every even integer greater than two is the sum of two primes, and this result has not been proved to date. Fermat's[15] Last Theorem (Fig. 4.12) states that there is no integer solution to $x^n + y^n = z^n$ for $n > 2$, and this result remained unproved for over 300 years until Andrew Wiles finally proved it in the mid-1990s.

The natural numbers $\mathbb{N}$ consist of the numbers $\{1, 2, 3, \ldots\}$. The integer numbers $\mathbb{Z}$ consist of $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$. The rational numbers $\mathbb{Q}$ consist of all numbers of the form $\{{}^{p}/_{q}$ where p and q are integers and $q \neq 0\}$. The real numbers $\mathbb{R}$ are defined to be the set of converging sequences of rational numbers and they are a superset of the rational numbers. They contain the rational and irrational numbers. The complex numbers $\mathbb{C}$ consist of all numbers of the form $\{a + bi$ where $a, b \in \mathbb{R}$ and $i = \sqrt{-1}\}$.

[14]Goldbach was an eighteenth-century German mathematician and Goldbach's conjecture has been verified to be true for all integers $n < 12 * 10^{17}$.

[15]Pierre de Fermat was a seventeenth-century French civil servant and amateur mathematician. He occasionally wrote to contemporary mathematicians announcing his latest theorem without providing the accompanying proof and inviting them to find the proof. The fact that he never revealed his proofs caused a lot of frustration among his contemporaries, and in his announcement of his famous last theorem he stated that he had a wonderful proof that was too large to include in the margin. He corresponded with Pascal, and they did some early work on the mathematical rules of games of chance and early probability theory.

Pythagorean triples are combinations of three whole numbers that satisfy Pythagoras's equation $x^2 + y^2 = z^2$. There are an infinite number of such triples, and an example of such a triple is 3, 4, 5 since $3^2 + 4^2 = 5^2$.

The Pythagoreans discovered the mathematical relationship between the harmony of music and numbers, and their philosophy was that numbers are hidden in everything from music to science and nature. This led to their philosophy that 'everything is number'. We will discuss number theory in more detail in Chap. 5 and show how it is applied to the field of cryptography in Chap. 10.

3.7 Automata Theory

Automata Theory is the branch of computer science that is concerned with the study of abstract machines and automata. These include finite-state machines, pushdown automata, and Turing machines. Finite-state machines are abstract machines that may be in one of a finite number of states. These machines are in only one state at a time (current state), and the input symbol causes a transition from the current state to the next state. Finite-state machines have limited computational power due to memory and state constraints, but they have been applied to several fields including communication protocols, neurological systems and linguistics.

Warren McCulloch and Walter Pitts published early work on finite-state automata in 1943. They were interested in modelling the thought process for humans and machines. Moore and Mealy developed this work further, and their finite-state machines are referred to as the '*Mealy machine*' and the '*Moore machine*'. The Mealy machine determines its outputs through the current state and the input, whereas the output of Moore's machine is based upon the current state alone.

Finite-state automata can compute only very primitive functions, and so they are not adequate as a model for computing. There are more powerful automata such as the Turing machine that is essentially a finite automaton with a potentially infinite storage (memory). Anything that is computable is computable by a Turing machine.

A finite-state machine can model a system that has a finite number of states, and a finite number of inputs/events that can trigger transitions between states. The behaviour of the system at a point in time is determined from the current state and input, with behaviour defined for the possible input to that state. The system starts in an initial state.

Pushdown automata have greater computational power, and they contain extra memory in the form of a stack from which symbols may be pushed or popped. The state transition is determined from the current state of the machine, the input symbol and the element on the top of the stack. The action may be to change the state and/or push/pop an element from the stack.

The Turing machine is the most powerful model for computation, and this theoretical machine is equivalent to an actual computer in the sense that it can compute the same set of functions. The memory of the Turing machine is a tape that consists of a potentially infinite number of one-dimensional cells. The Turing