that must be accounted for when tallying the total energy cost of the computation. We analyze the energy cost associated with error-correction in more detail in Section 12.4.4.

What can we conclude from our study of reversible computation? There are three key ideas. First, reversibility stems from keeping track of every bit of information; irreversibility occurs only when information is lost or erased. Second, by doing computation reversibly, we obviate the need for energy expenditure during computation. All computations can be done, in principle, for zero cost in energy. Third, reversible computation can be done efficiently, without the production of garbage bits whose value depends upon the input to the computation. That is, if there is an irreversible circuit computing a function f, then there is an efficient simulation of this circuit by a reversible circuit with action $(x, y) \rightarrow (x, y \oplus f(x))$.

What are the implications of these results for physics, computer science, and for quantum computation and quantum information? From the point of view of a physicist or hardware engineer worried about heat dissipation, the good news is that, in principle, it is possible to make computation dissipation-free by making it reversible, although in practice energy dissipation is required for system stability and immunity from noise. At an even more fundamental level, the ideas leading to reversible computation also lead to the resolution of a century-old problem in the foundations of physics, the famous problem of *Maxwell's demon*. The story of this problem and its resolution is outlined in Box 3.5 on page 162. From the point of view of a computer scientist, reversible computation validates the use of irreversible elements in models of computation such as the Turing machine (since using them or not gives polynomially equivalent models). Moreover, since the physical world is fundamentally reversible, one can argue that complexity classes based upon reversible models of computation are more natural than complexity classes based upon irreversible models, a point revisited in Problem 3.9 and 'History and further reading'. From the point of view of quantum computation and quantum information, reversible computation is enormously important. To harness the full power of quantum computation, any classical subroutines in a quantum computation must be performed reversibly and without the production of garbage bits depending on the classical input.

Exercise 3.32: (From Fredkin to Toffoli and back again) What is the smallest number of Fredkin gates needed to simulate a Toffoli gate? What is the smallest number of Toffoli gates needed to simulate a Fredkin gate?

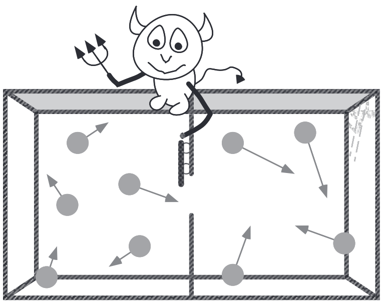
3.3 Perspectives on computer science

In a short introduction such as this chapter, it is not remotely possible to cover in detail all the great ideas of a field as rich as computer science. We hope to have conveyed to you something of what it means to *think* like a computer scientist, and provided a basic vocabulary and overview of some of the fundamental concepts important in the understanding of computation. To conclude this chapter, we briefly touch on some more general issues, in order to provide some perspective on how quantum computation and quantum information fits into the overall picture of computer science.

Our discussion has revolved around the Turing machine model of computation. How does the computational power of unconventional models of computation such as massively parallel computers, DNA computers and analog computers compare with the standard

Box 3.5: Maxwell's demon

The laws of thermodynamics govern the amount of work that can be performed by a physical system at thermodynamic equilibrium. One of these laws, the second law of thermodynamics, states that the *entropy* in a closed system can never decrease. In 1871, James Clerk Maxwell proposed the existence of a machine that apparently violated this law. He envisioned a miniature little 'demon', like that shown in the figure below, which could reduce the entropy of a gas cylinder initially at equilibrium by individually separating the fast and slow molecules into the two halves of the cylinder. This demon would sit at a little door at the middle partition. When a fast molecule approaches from the left side the demon opens a door between the partitions, allowing the molecule through, and then closes the door. By doing this many times the total entropy of the cylinder can be *decreased*, in apparent violation of the second law of thermodynamics.



The resolution to the Maxwell's demon paradox lies in the fact that the demon must perform *measurements* on the molecules moving between the partitions, in order to determine their velocities. The result of this measurement must be stored in the demon's memory. Because any memory is finite, the demon must eventually begin erasing information from its memory, in order to have space for new measurement results. By Landauer's principle, this act of erasing information increases the total entropy of the combined system – demon, gas cylinder, and their environments. In fact, a complete analysis shows that Landauer's principle implies that the entropy of the combined system is increased *at least as much* by this act of erasing information as the entropy of the combined system is decreased by the actions of the demon, thus ensuring that the second law of thermodynamics is obeyed.

Turing machine model of computation and, implicitly, with quantum computation? Let's begin with parallel computing architectures. The vast majority of computers in existence are serial computers, processing instructions one at a time in some central processing unit. By contrast, parallel computers can process more than one instruction at a time, leading to a substantial savings in time and money for some applications. Nevertheless, parallel processing does not offer any fundamental advantage over the standard Turing machine model when issues of efficiency are concerned, because a Turing machine can simulate a parallel computer with polynomially equivalent total physical resources – the total space and time used by the computation. What a parallel computer gains in time, it loses in the total spatial resources required to perform the computation, resulting in a net of no essential change in the power of the computing model.

An interesting specific example of massively parallel computing is the technique of *DNA computing*. A strand of DNA, deoxyribonucleic acid, is a molecule composed of a sequence (a polymer) of four kinds of nucleotides distinguished by the bases they carry, denoted by the letter A (adenine), C (cytosine), G (guanine) and T (thymine). Two strands, under certain circumstances, can anneal to form a double strand, if the respective base pairs form complements of each other (A matches T and G matches C). The ends are also distinct and must match appropriately. Chemical techniques can be used to amplify the number of strands beginning or ending with specific sequences (polymerase chain reaction), separate the strands by length (gel electrophoresis), dissolve double strands into single strands (changing temperature and pH), read the sequence on a strand, cut strands at a specific position (restriction enzymes), and detect if a certain sequence of DNA is in a test tube. The procedure for using these mechanisms in a robust manner is rather involved, but the basic idea can be appreciated from an example.

The directed Hamiltonian path problem is a simple and equivalently hard variant of the Hamiltonian cycle problem of Section 3.2.2, in which the goal is to determine if a path exists or not between two specified vertices j_1 and j_N in a directed graph G of N vertices, entering each vertex exactly once, and following only allowed edge directions. This problem can be solved with a DNA computer using the following five steps, in which x_j are chosen to be unique sequences of bases (and $\bar{x}_j$ their complements), DNA strands $x_j x_k$ encode edges, and strands $\bar{x}_j \bar{x}_j$ encode vertices. (1) Generate random paths through G, by combining a mixture of all possible vertex and edge DNA strands, and waiting for the strands to anneal. (2) Keep only the paths beginning with j_1 and ending with j_N, by amplifying only the double strands beginning with $\bar{x}_{j_1}$ and ending with $\bar{x}_{j_N}$. (3) Select only paths of length N, by separating the strands according to their length. (4) Select only paths which enter each vertex at least once, by dissolving the DNA into single strands, and annealing with all possible vertex strands one at a time and filtering out only those strands which anneal. And (5) detect if any strands have survived the selection steps; if so, then a path exists, and otherwise, it does not. To ensure the answer is correct with sufficiently high probability, x_j may be chosen to contain many (≈ 30) bases, and a large number ($\approx 10^{14}$ or more are feasible) of strands are used in the reaction.

Heuristic methods are available to improve upon this basic idea. Of course, exhaustive search methods such as this only work as long as all possible paths can be generated efficiently, and thus the number of molecules used must grow exponentially as the size of the problem (the number of vertices in the example above). DNA molecules are relatively small and readily synthesized, and the huge number of DNA combinations one can fit into a test tube can stave off the exponential complexity cost increase for a while – up to a few dozen vertices – but eventually the exponential cost limits the applicability of this method. Thus, while DNA computing offers an attractive and physically realizable model of computation for the solution of certain problems, it is a classical computing technique and offers no essential improvement in principle over a Turing machine.

Analog computers offer a yet another paradigm for performing computation. A computer is analog when the physical representation of information it uses for computation is based on continuous degrees of freedom, instead of zeroes and ones. For example, a thermometer is an analog computer. Analog circuitry, using resistors, capacitors, and amplifiers, is also said to perform analog computation. Such machines have an infinite resource to draw upon in the ideal limit, since continuous variables like position and voltage can store an unlimited amount of information. But this is only true in the absence of noise. The presence of a finite amount of noise reduces the number of *distinguishable states* of a continuous variable to a finite number – and thus restricts analog computers to the representation of a finite amount of information. In practice, noise reduces analog computers to being no more powerful than conventional digital computers, and through them Turing machines. One might suspect that quantum computers are just analog computers, because of the use of continuous parameters in describing qubit states; however, it turns out that the effects of noise on a quantum computer can effectively be *digitized*. As a result, their computational advantages remain even in the presence of a finite amount of noise, as we shall see in Chapter 10.

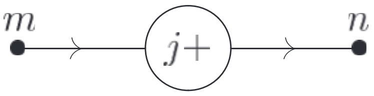
What of the effects of noise on digital computers? In the early days of computation, noise was a very real problem for computers. In some of the original computers a vacuum tube would malfunction every few minutes. Even today, noise is a problem for computational devices such as modems and hard drives. Considerable effort was devoted to the problem of understanding how to construct reliable computers from unreliable components. It was proven by von Neumann that this is possible with only a polynomial increase in the resources required for computation. Ironically, however, modern computers use none of those results, because the components of modern computers are fantastically reliable. Failure rates of 10^{-17} and even less are common in modern electronic components. For this reason, failures happen so rarely that the extra effort required to protect against them is not regarded as being worth making. On the other hand, we shall find that quantum computers are very delicate machines, and will likely require substantial application of error-correction techniques.

Different architectures may change the effects of noise. For example, if the effect of noise is ignored, then changing to a computer architecture in which many operations are performed in parallel may not change the number of operations which need to be done. However, a parallel system may be substantially more resistant to noise, because the effects of noise have less time to accumulate. Therefore, in a realistic analysis, the parallel version of an algorithm may have some substantial advantages over a serial implementation. Architecture design is a well developed field of study for classical computers. Hardly anything similar has been developed along the same lines for quantum computers, but the study of noise already suggests some desirable traits for future quantum computer architectures, such as a high level of parallelism.

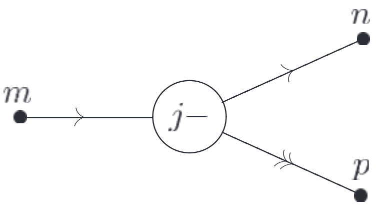
A fourth model of computation is *distributed computation*, in which two or more spatially separated computational units are available to solve a computational problem. Obviously, such a model of computation is no more powerful than the Turing machine model in the sense that it can be efficiently simulated on a Turing machine. However, distributed computation gives rise to an intriguing new resource challenge: how best to utilize multiple computational units when the cost of *communication* between the units is high. This problem of distributed computation becomes especially interesting as computers are connected through high speed networks; although the total computational capacity of all the computers on a network might be extremely large, utilization of that potential is difficult. Most interesting problems do not divide easily into independent chunks that can be solved separately, and may frequently require global communication between different computational subsystems to exchange intermediate results or synchronize status. The field of *communication complexity* has been developed to address such issues, by quantifying the cost of communication requirements in solving problems. When quantum resources are available and can be exchanged between distributed computers, the communication costs can sometimes be greatly reduced.

A recurring theme through these concluding thoughts and through the entire book is that despite the traditional independence of computer science from physical constraints, ultimately physical laws have tremendous impact not only upon how computers are realized, but also the class of problems they are capable of solving. The success of quantum computation and quantum information as a physically reasonable alternative model of computation questions closely held tenets of computer science, and thrusts notions of computer science into the forefront of physics. The task of the remainder of this book is to stir together ideas from these disparate fields, and to delight in what results!

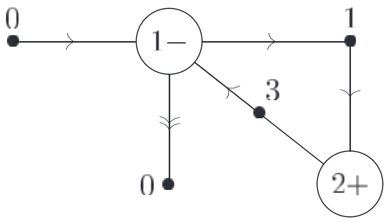
Problem 3.1: (Minsky machines) A *Minsky machine* consists of a finite set of *registers*, $r_1, r_2, \ldots, r_k$, each capable of holding an arbitrary non-negative integer, and a *program*, made up of *orders* of one of two types. The first type has the form:



The interpretation is that at point m in the program register r_j is incremented by one, and execution proceeds to point n in the program. The second type of order has the form:



The interpretation is that at point m in the program, register r_j is decremented if it contains a positive integer, and execution proceeds to point n in the program. If register r_j is zero then execution simply proceeds to point p in the program. The *program* for the Minsky machine consists of a collection of such orders, of a form like:



The starting and all possible halting points for the program are conventionally labeled zero. This program takes the contents of register r_1 and adds them to register r_2, while decrementing r_1 to zero.

(1) Prove that all (Turing) computable functions can be computed on a Minsky machine, in the sense that given a computable function $f(\cdot)$ there is a Minsky machine program that when the registers start in the state $(n, 0, \ldots, 0)$ gives as output $(f(n), 0, \ldots, 0)$.
(2) Sketch a proof that any function which can be computed on a Minsky machine, in the sense just defined, can also be computed on a Turing machine.

Problem 3.2: (Vector games) A *vector game* is specified by a finite list of vectors, all of the same dimension, and with integer co-ordinates. The game is to start with a vector x of non-negative integer co-ordinates and to add to x the first vector from the list which preserves the non-negativity of all the components, and to repeat this process until it is no longer possible. Prove that for any computable function $f(\cdot)$ there is a vector game which when started with the vector $(n, 0, \ldots, 0)$ reaches $(f(n), 0, \ldots, 0)$. (*Hint*: Show that a vector game in $k + 2$ dimensions can simulate a Minsky machine containing k registers.)

Problem 3.3: (Fractran) A *Fractran* program is defined by a list of positive rational numbers $q_1, \ldots, q_n$. It acts on a positive integer m by replacing it by $q_i m$, where i is the least number such that $q_i m$ is an integer. If there is ever a time when there is no i such that $q_i m$ is an integer, then execution stops. Prove that for any computable function $f(\cdot)$ there is a Fractran program which when started with 2^n reaches $2^{f(n)}$ without going through any intermediate powers of 2. (*Hint*: use the previous problem.)

Problem 3.4: (Undecidability of dynamical systems) A Fractran program is essentially just a very simple dynamical system taking positive integers to positive integers. Prove that there is no algorithm to decide whether such a dynamical system ever reaches 1.

Problem 3.5: (Non-universality of two bit reversible logic) Suppose we are trying to build circuits using only one and two bit reversible logic gates, and ancilla bits. Prove that there are Boolean functions which cannot be computed in this fashion. Deduce that the Toffoli gate cannot be simulated using one and two bit reversible gates, even with the aid of ancilla bits.

Problem 3.6: (Hardness of approximation of TSP) Let $r \geq 1$ and suppose that there is an approximation algorithm for TSP which is guaranteed to find the shortest tour among n cities to within a factor r. Let $G = (V, E)$ be any graph on n vertices. Define an instance of TSP by identifying cities with vertices in V, and defining the distance between cities i and j to be 1 if (i, j) is an edge of G, and to be $\lceil r \rceil |V| + 1$ otherwise. Show that if the approximation algorithm is applied to this instance of TSP then it returns a Hamiltonian cycle for G if one exists, and otherwise returns a tour of length more than $\lceil r \rceil |V|$. From the **NP**-completeness of HC it follows that no such approximation algorithm can exist unless **P** = **NP**.

Problem 3.7: (Reversible Turing machines)

(1) Explain how to construct a reversible Turing machine that can compute the same class of functions as is computable on an ordinary Turing machine. (*Hint*: It may be helpful to use a multi-tape construction.)

(2) Give general space and time bounds for the operation of your reversible Turing machine, in terms of the time $t(x)$ and space $s(x)$ required on an ordinary single-tape Turing machine to compute a function $f(x)$.

Problem 3.8: (Find a hard-to-compute class of functions (Research)) Find a natural class of functions on n inputs which requires a super-polynomial number of Boolean gates to compute.

Problem 3.9: (Reversible PSPACE = PSPACE) It can be shown that the problem 'quantified satisfiability', or QSAT, is **PSPACE**-complete. That is, every other language in **PSPACE** can be reduced to QSAT in polynomial time. The language QSAT is defined to consist of all Boolean formulae φ in n variables $x_1, \ldots, x_n$, and in conjunctive normal form, such that:

$$\exists_{x_1} \forall_{x_2} \exists_{x_3} \ldots \forall_{x_n} \varphi \text{ if } n \text{ is even;} \tag{3.9}$$

$$\exists_{x_1} \forall_{x_2} \exists_{x_3} \ldots \exists_{x_n} \varphi \text{ if } n \text{ is odd.} \tag{3.10}$$

Prove that a reversible Turing machine operating in polynomial space can be used to solve QSAT. Thus, the class of languages decidable by a computer operating reversibly in polynomial space is equal to **PSPACE**.

Problem 3.10: (Ancilla bits and efficiency of reversible computation) Let p_m be the mth prime number. Outline the construction of a reversible circuit which, upon input of m and n such that $n > m$, outputs the product $p_m p_n$, that is $(m, n) \rightarrow (p_m p_n, g(m, n))$, where $g(m, n)$ is the final state of the ancilla bits used by the circuit. Estimate the number of ancilla qubits your circuit requires. Prove that if a polynomial (in $\log n$) size reversible circuit can be found that uses $O(\log(\log n))$ ancilla bits then the problem of factoring a product of two prime numbers is in **P**.

History and further reading

Computer science is a huge subject with many interesting subfields. We cannot hope for any sort of completeness in this brief space, but instead take the opportunity to recommend a few titles of general interest, and some works on subjects of specific interest in relation to topics covered in this book, with the hope that they may prove stimulating.

Modern computer science dates to the wonderful 1936 paper of Turing[Tur36]. The Church–Turing thesis was first stated by Church[Chu36] in 1936, and was then given a more complete discussion from a different point of view by Turing. Several other researchers found their way to similar conclusions at about the same time. Many of these contributions and a discussion of the history may be found in a volume edited by Davis[Dav65]. Provocative discussions of the Church–Turing thesis and undecidability may be found in Hofstadter[Hof79] and Penrose[Pen89].

There are many excellent books on algorithm design. We mention only three. First, there is the classic series by Knuth[Knu97, Knu98a, Knu98b] which covers an enormous portion of computer science. Second, there is the marvelous book by Cormen, Leiserson, and Rivest[CLR90]. This huge book contains a plethora of well-written material on many areas of algorithm design. Finally, the book of Motwani and Raghavan[MR95] is an excellent survey of the field of randomized algorithms.

The modern theory of computational complexity was especially influenced by the papers of Cook[Coo71] and Karp[Kar72]. Many similar ideas were arrived at independently in Russia by Levin[Lev73], but unfortunately took time to propagate to the West. The classic book by Garey and Johnson[GJ79] has also had an enormous influence on the field. More recently, Papadimitriou[Pap94] has written a beautiful book that surveys many of the main ideas of computational complexity theory. Much of the material in this chapter is based upon Papadimitriou's book. In this chapter we considered only one type of reducibility between languages, polynomial time reducibility. There are many other notions of reductions between languages. An early survey of these notions was given by Ladner, Lynch and Selman[LLS75]. The study of different notions of reducibility later blossomed into a subfield of research known as *structural complexity*, which has been reviewed by Balcázar, Diaz, and Gabarró[BDG88a, BDG88b].

The connection between information, energy dissipation, and computation has a long history. The modern understanding is due to a 1961 paper by Landauer[Lan61], in which Landauer's principle was first formulated. A paper by Szilard[Szi29] and a 1949 lecture by von Neumann[von66] (page 66) arrive at conclusions close to Landauer's principle, but do not fully grasp the essential point that it is the *erasure* of information that requires dissipation.

Reversible Turing machines were invented by Lecerf[Lec63] and later, but independently, in an influential paper by Bennett[Ben73]. Fredkin and Toffoli[FT82] introduced reversible circuit models of computation. Two interesting historical documents are Barton's May, 1978 MIT 6.895 term paper[Bar78], and Ressler's 1981 Master's thesis[Res81], which contain designs for a reversible PDP-10! Today, reversible logic is potentially important in implementations of low-power CMOS circuitry[YK95].

Maxwell's demon is a fascinating subject, with a long and intricate history. Maxwell proposed his demon in 1871[Max71]. Szilard published a key paper in 1929[Szi29] which anticipated many of the details of the final resolution of the problem of Maxwell's demon. In 1965 Feynman[FLS65b] resolved a special case of Maxwell's demon. Bennett, building on Landauer's work[Lan61], wrote two beautiful papers on the subject[BBBW82, Ben87] which completed the resolution of the problem. An interesting book about the history of Maxwell's demon and its exorcism is the collection of papers by Leff and Rex[LR90].

DNA computing was invented by Adleman, and the solution of the directed Hamiltonian path problem we describe is his[Adl94]. Lipton has also shown how 3SAT and circuit satisfiability can be solved in this model[Lip95]. A good general article is Adleman's *Scientific American* article[Adl98]; for an insightful look into the universality of DNA operations, see Winfree[Win98]. An interesting place to read about performing reliable computation in the presence of noise is the book by Winograd and Cowan[WC67]. This topic will be addressed again in Chapter 10. A good textbook on computer architecture is by Hennessey, Goldberg, and Patterson.[HGP96].

Problems 3.1 through 3.4 explore a line of thought originated by Minsky (in his beautiful book on computational machines[Min67]) and developed by Conway[Con72, Con86]. The Fractran programming language is certainly one of the most beautiful and elegant universal computational models known, as demonstrated by the following example, known as PRIMEGAME[Con86]. PRIMEGAME is defined by the list of rational numbers:

$$\frac{17}{91};\frac{78}{85};\frac{19}{51};\frac{23}{38};\frac{29}{33};\frac{77}{29};\frac{95}{23};\frac{77}{19};\frac{1}{17};\frac{11}{13};\frac{13}{11};\frac{15}{2};\frac{1}{7};\frac{55}{1}. \tag{3.11}$$

Amazingly, when PRIMEGAME is started at 2, the other powers of 2 that appear, namely, $2^2, 2^3, 2^5, 2^7, 2^{11}, 2^{13}, \ldots$, are precisely the prime powers of 2, with the powers stepping through the prime numbers, in order. Problem 3.9 is a special case of the more general subject of the spatial requirements for reversible computation. See the papers by Bennett[Ben89], and by Li, Tromp and Vitanyi[LV96, LTV98].