Introduction

Michael E. Cuffaro and Samuel C. Fletcher

Our modern understanding of computation stems in large part from Alan Turing's formalization of the mathematical activity of following an effective method for computing a function. Thus the roots of computer science (at least as traditionally construed) are in this sense those of an essentially mathematical science. The science of Physics, on the other hand, ultimately aims to describe the characteristics of concrete systems as they exist in the natural world. It is thus a nontrivial question to ask whether, and how, physics can illuminate computer science and vice versa.

Indeed, the possible questions one may ask regarding the connections between computation and physics are many, varied, and multi-faceted. For the purposes of a philosophical investigation into these connections they can be usefully characterized as falling into two main categories. On the one hand, there are those questions related to the connections between computational and physical systems, and on the other hand there are those questions related to the connections between computational and physical theory in general. These two main categories can further be subdivided into two sub-categories each, which together comprise the four major parts of this volume:

- Interrelations between computational and physical systems
    - I. The computability of physical systems and physical systems as computers
    - II. The implementation of computation in physical systems
- Interrelations between computational and physical theory
    - III. Physical perspectives on computer science
    - IV. Computational perspectives on physical theory

In the remainder of this introductory chapter, we will summarize each of these parts and the particular contributions of this volume that fall under them. Before we do so, however, it will be useful to review some of the basic concepts which will generally be taken for granted in the rest of the book.

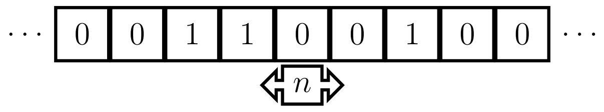

Figure 1: A representation of a Turing machine with read/write head in state n and tape entries "0" and "1" representing blank and marked squares, respectively.

1. Computability theory and the Church-Turing thesis

Intuitively, computability theory concerns which tasks can be completed in principle by following a completely explicit set of instructions. These instructions must be definite, in the sense that they allow no procedural interpretation or flexibility, and self-contained, in the sense that they require no input other than what is provided in the description of the task itself. Such a set of instructions, called an *effective procedure*, hence demands no creativity of whoever (or whatever) executes it.

Effective procedures have found their greatest application in the mathematical domain, many of whose problems can be reduced to the computation of a function of natural numbers. A function $f : \mathbb{N}^k \to \mathbb{N}$ is said to be *effectively computable* when there is an effective procedure for calculating its value for any argument. For example, the familiar elementary arithmetic functions of addition and multiplication are clearly effectively computable, as are functions composed from them. Further, the effective computability of a mathematical decision problem, such as "Is n prime?", can be encoded into a function whose range is $\{0, 1\}$, corresponding with the "no" and "yes" answers.

To make this informal concept of effective computation formally tractable, myriad models for computation have been proposed, inspired variously from logic, arithmetic, and mechanics.[1] One might naturally expect that these different proposals, various as they are in their starting points, lead to formalizations of differing computational strength. So it is remarkable that they in fact determine extensionally the same class of functions as being computable.

The most important and influential of these proposals, on which we focus in this introduction, is that of the Turing machine (TM). A TM is a type of abstract state machine, consisting of the following components (an example of which is illustrated in Figure 0.1):

- An arbitrarily long tape, divided into sequential squares that can be blank or contain a mark.

- A read/write head, which sits atop a particular square, can read whether it contains a mark, and can perform the following actions: print a mark on the square if it doesn't have one, erase the mark on the square if it has one, move one square to the right, and move one square to the left.

- A program, or finite set of instructions, for the read/write head, each of which has the following form:

[1]Examples include representability in a formal system, the λ-calculus, recursive function theory, Markov algorithms, register machines, and Turing machines (what we focus on below). See Epstein and Carnielli (2008, ch. 8E) for brief descriptions and references to these various approaches.

TM Instruction Form In state n, if the current square is [blank/marked], perform [action] and transition to state m. In abbreviated form: $(n, [0/1], [\text{action}], m)$.

The tape is the medium for the input and output to a proposed calculation by the machine, as well as for all the intermediate work required to transition between them. It typically encodes these in binary notation, for example with blank and marked squares representing the numerals "0" and "1," respectively. The read/write head performs the steps leading to the computation through its fixed set of actions. The program encodes an effective procedure for the TM to follow:

1. A TM begins in some pre-specified state at some pre-specified location on the tape.
2. The read/write head reads its current square, then performs the action (if any) specified by the program according to what's read and the current state.
3. It then transitions to another state, according to the program, whereupon step two is repeated.

A program need not have an action and state transition specified for every state and input from the current square. If it does not, then the read/write head halts, indicating the end of the computation, at which point the contents of the tape represent the computation's output.[2]

Suppose now that an encoding of inputs and outputs of natural numbers on the tape and a starting location for the printing head on the tape has been fixed. One then says that a function $f : \mathbb{N}^k \to \mathbb{N}$ is *Turing-computable* if and only if there is a TM that for all $(n_1, \ldots, n_k) \in \mathbb{N}^k$ eventually halts with output encoding $f(n_1, \ldots, n_k)$ when begun with input encoding $(n_1, \ldots, n_k)$.

Because the TM's tape represents the inputs and outputs of the function it computes and the functionality of the read/write head is fixed, the real source of variability amongst TMs comes from the program. TMs may thus be enumerated by their programs, which can be represented by sets of ordered quadruples of the above specified TM Instruction Form. For example, the set $\{(1, 0, 1, 1), (1, 1, \to, 1)\}$ defines a program (i.e., a TM) with only one state, in which it writes a mark if the current square is empty and otherwise moves to the right. Since every Turing-computable function must be computable by some TM, it follows immediately that the Turing-computable functions are enumerable. But because there are uncountably many functions of natural numbers, there must be functions that are not Turing-computable – infinitely more, in fact, than those that are.

Two further remarkable facts build upon such an enumeration.[3] First is the existence of *universal* TMs, ones that can simulate the computation of any other TM. In other words, there exist TMs such that, when given input encoding $(m, n_1, \ldots, n_k)$, they eventually halt with output $f_m(n_1, \ldots, n_k)$, where f_m is the Turing-computable function computed by the mth TM. Second is the specification of concrete non-Turing-computable functions. Most

[2]Despite the physically evocative story involving "components" and so on, a concrete mechanism for implementing or constructing an actual TM is neither provided nor necessary. This is the sense in which the TM is an *abstract* machine providing a *mathematical* definition of computation, rather than a schematic for a physical machine providing an empirical account of computation; cf. Section 4 of this Introduction.

[3]For more on TMs, see Barker-Plummer (2016) and references cited therein.

famous of these is the halting function $h : \mathbb{N}^2 \to \mathbb{N}$, which is equal to 1 if TM m halts on input n, and is equal to 2 otherwise.

The theory of computability can be developed much further,[4] but to close this section we circle back to the original motivation for TMs: does Turing-computability adequately formalize the concept of effective computability? Clearly every Turing-computable function is effectively computable, for the action of a TM that computes such a function is given by an effective procedure. The statement that the converse is also true is known as either *Turing's Thesis* or the *Church-Turing Thesis*.

Church-Turing Thesis (CTT) Every effectively computable function of natural numbers is Turing computable.

The truth of the CTT would imply that one can identify or replace the extension of the informal concept of effective computability with that of the formal concept of Turing-computability, thereby establishing a completely adequate explication (cf. Carnap 1947, sec. 2; Carnap 1950b, ch. 1) of the former. There is a large literature on the status and interpretation of the CTT,[5] but it is fair to say that it is widely accepted among computer scientists and beyond. That said, the theory of computability and the CTT only make claims about what is possible *in principle* to compute, given the idealizations of arbitrarily large temporal, spatial, and material resources – computing steps, tape squares, and the incorruptible functioning of Turing machinery – that abstract away from their actual abundance. When an accounting of these resources is brought to bear, as in the next section of this Introduction, one can distinguish not just between computable and non-computable functions, but, among the computable ones, those of various degrees of difficulty.

2. Computational complexity theory

In computational complexity theory, computational problems are classified based on their resource costs, i.e., those in time and space. We will focus on time, which is the more important measure. Arguably the most basic distinction within the theory is that between those decision problems (i.e., yes-or-no questions) that are "easy" (a.k.a. "feasible," "efficiently solvable," "tractable," etc.) and those that are not (i.e., "hard"). According to the Cobham-Edmonds thesis (Dean 2016b), a decision problem is easy if it is solvable in "polynomial time," i.e., if it can be solved in a number of steps bounded by a polynomial function of its input size, n. Problems so solvable on a deterministic Turing machine (DTM) comprise the complexity class P.

Formally, one can conceive of a decision problem as one of determining whether a given string x of length n is in the "language" L. For example, determining whether x is prime amounts to determining whether it is in the language $\{10, 11, 101, 111, 1011, 1101, 10001, 10011, \ldots\}$ (the set of binary representations of prime numbers). Now, call a language L a member of the class DTIME$(T(n))$ if and only if there is a DTM for deciding membership in L whose running time, $t(n)$, is "on the order of $T(n)$," or in symbols: $O(T(n))$. Here, $T(n)$ represents an upper bound for the growth rate of $t(n)$ in the sense that, by definition, $t(n)$

[4]See, for instance, Immerman (2016) and references cited therein.
[5]See, for instance, Copeland (2015) and references cited therein.

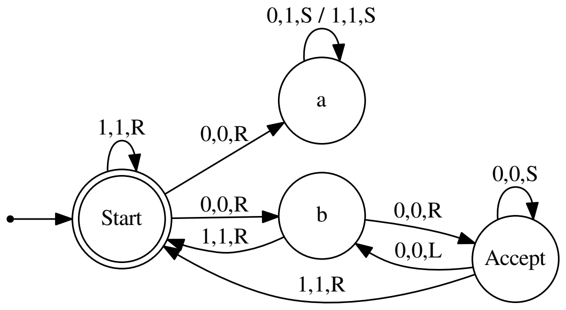

Figure 2: This NTM accepts binary strings ending in "00," since for a given such x, there exists a series of transitions which end in "Accept." But this is not guaranteed. The machine is guaranteed, on the other hand, to reject any string not ending in "00." An edge from s_1 to s_2 labeled α, β, P is read as: In state s_1, read α from the tape, overwrite α with β, move the read/write head to P (L = to the left, R = to the right, S = same), and finally transition to state s_2.

is $O(T(n))$ if for every sufficiently large n, $t(n) \leq k \cdot T(n)$ for some constant k.[6] We can now formally characterize P (Arora and Barak 2009, p. 25) as:

$$\mathrm{P} = \bigcup_{k \geq 1} \mathrm{DTIME}(n^k). \tag{1}$$

A *nondeterministic Turing machine* (NTM) is such that it may "choose," when in a given state, which one of a set of possible successor states to transition to; see Figure 0.2. It is said to accept a string x if and only if there exists a path through its state space that, given x, leads to an accepting state. It rejects x otherwise. NTIME($T(n)$) is now defined, analogously to DTIME($T(n)$), as the set of languages for which an NTM exists to decide, in $O(T(n))$ steps, whether a given string x of length n is in L. The class "NP" is defined as:[7]

$$\mathrm{NP} =_{df} \bigcup_{k \geq 1} \mathrm{NTIME}(n^k). \tag{2}$$

Exactly how an NTM "chooses" to follow one path rather than another is not defined. In a *probabilistic Turing machine* (PTM), in contrast, we associate a particular probability with each possible transition. We can then define the class BPP (bounded-error probabilistic polynomial time) as the class of languages such that there exists a polynomial-time PTM which, on any given run, will correctly determine whether a string x is in the language L with probability $\geq 2/3$.[8]

It has been conjectured that any language L decidable under a given "reasonable" (i.e. physically realizable) machine model $\mathfrak{M}$ is "efficiently simulable" by a PTM in the sense that a PTM to decide L exists which requires at most a polynomial number of extra time steps

[6]By "for every sufficiently large n" it is meant that there exists some $n_0 \geq 1$ such that $t(n) \leq k \cdot T(n)$ whenever $n \geq n_0$.

[7]Equivalently (Arora and Barak 2009, p. 42), NP is the set of languages for which one can construct a polynomial-time *deterministic* TM to verify, for any x, that $x \in L$, given a polynomial-length string u (called a "certificate" for x).

[8]The particular threshold probability 2/3 is inessential. Any probability $p_{min} \geq 1/2 + n^{-k}$, with k a constant, will yield the same class (Arora and Barak 2009, p. 132).

compared to a machine of type $\mathfrak{M}$. This is known as the "strong" or "extended" Church-Turing thesis (ECT).[9,10] Over the last three decades, however, evidence has been mounting against the ECT, primarily as a result of the advent of quantum computing (Aaronson 2013, chs. 10, 15), which we will discuss in more detail in the next section.

3. Quantum computing

The best known classical algorithm for factoring arbitrary integers, the number field sieve (Lenstra et al. 1990), requires $O(2^{(\log N)^{1/3}})$ steps to factor a given integer N. Shor's *quantum* factoring algorithm requires only a number of steps that is polynomial in $\log N$ – an exponential speedup over the number field sieve. This and other quantum algorithms provide evidence that the ECT is false, for they seem to show that BQP, the class of languages probabilistically decidable by a quantum computer in polynomial time, is strictly larger than BPP. Note, however, that although the evidence furnished by Shor's algorithm is strong, it is still an open question whether factoring is in BPP.[11]

The state of a classical digital computer, whether deterministic or probabilistic, is describable as a sequence of bits. A bit can be directly instantiated by any two-level classical physical system, such as a circuit that can be open or closed. In a *quantum* computer, the basic unit of representation is not the bit but the qubit. To directly instantiate it, one uses a two-level quantum system such as an electron (specifically: its spin). Like a bit, a qubit can be "on": $|0\rangle$, or "off": $|1\rangle$. In general, however, a qubit's state can be expressed as a normalized linear superposition:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \tag{3}$$

where the complex "amplitudes" α and β satisfy the normalization condition: $|\alpha|^2 + |\beta|^2 = \alpha\bar{\alpha} + \beta\bar{\beta} = 1$, with $\bar{c}$ the complex conjugate of c. We refer to $|\psi\rangle$ as the "state vector" for the qubit.

Unlike a bit, not all states of a qubit can be observed directly. In particular, one never observes a qubit in a linear superposition with respect to a particular measurement basis. For example, a "computational basis" measurement – "$|0\rangle$ or $|1\rangle$?" – will never reveal a qubit state of the form of equation 3, aside from the trivial case where one of α or β is 0. In general, given the initial state in equation 3, such a measurement on a qubit will find it in the state $|0\rangle$ with probability $|\alpha|^2$ and in state $|1\rangle$ with probability $|\beta|^2$.

Quantum computers can efficiently simulate classical probabilistic computers, since it is "easy" in a complexity-theoretic sense to simulate a fair classical coin toss. For example, one can instantiate the transition Q, defined as:

$$Q|0\rangle \rightarrow \frac{i}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle, \qquad Q|1\rangle \rightarrow \frac{1}{\sqrt{2}}|0\rangle + \frac{i}{\sqrt{2}}|1\rangle,$$

[9]For further discussion of the ECT and related issues, see Dean (2016a,b,c).

[10]ECT is sometimes defined with respect to the TM rather than the PTM model, for there has been mounting evidence that P = BPP (Arora and Barak 2009). For our purposes the choice of TM or PTM is inessential; a TM is a special case of a PTM for which transition probabilities are always either 0 or 1. Moreover, defining ECT with respect to the PTM model is convenient when comparing classical computation in general with quantum computation, which is probabilistic.

[11]For further discussion, see Cuffaro (in press).

and then measure in the computational basis.

Unlike classical bits, qubits can sometimes exhibit "interference effects." For example, upon applying the "Q-gate" twice to a qubit in the initial state $|0\rangle$, "destructive" and "constructive" interference is exhibited between the complex amplitudes associated with the $|0\rangle$ and $|1\rangle$ components of the state vector, respectively:

$$|0\rangle \xrightarrow{Q} \Big(\frac{i}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\Big) \xrightarrow{Q} \Big(-\frac{1}{2}|0\rangle + \frac{i}{2}|1\rangle + \frac{1}{2}|0\rangle + \frac{i}{2}|1\rangle\Big) = i|1\rangle.$$

A computational basis measurement on the qubit will now yield $|1\rangle$ with certainty. This ability to exhibit interference effects is held by some to be the key to understanding the source of the power of quantum computers (Fortnow 2003; Aaronson 2013).

The combined state of two or more qubits is said to be separable if it can be expressed as a product state:

$$|\alpha\rangle \otimes |\beta\rangle \otimes |\gamma\rangle \ldots.$$

The state

$$|\psi\rangle = |0\rangle \otimes |0\rangle + |0\rangle \otimes |1\rangle + |1\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle = (|0\rangle + |1\rangle) \otimes (|0\rangle + |1\rangle)$$

is an example. Not all states of more than one qubit are separable states. The following is an entangled state; it cannot be expressed as a product state:

$$|\Phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}.$$

The ability of qubits to form entangled states when combined is another oft-cited source of the power of quantum computers (Steane 2003). Entanglement has been shown to be necessary for achieving quantum speedup when using pure states (Jozsa and Linden 2003). However in the same paper, Jozsa and Linden argue that it may not be a sufficient resource. For an in-depth discussion, see Cuffaro (2017).

Other purported sources of quantum speedup include: massive quantum parallelism achievable through the ability of qubits to realize superposition states (Pitowsky 2002; Duwell, this volume); the same but with an additional ontological posit of many computational worlds (Hewitt-Horsman 2009; Cuffaro 2012); quantum contextuality (Howard et al. 2014); and the structure of quantum logic (Bub 2010). For an overview and further discussion, see Hagar and Cuffaro (2017).

4. Computational implementation and the physical Church-Turing theses

The previous sections of this Introduction concerned what could be computed and how efficiently, largely abstracting from most of the physical, mechanical, and engineering details that would be necessary to describe adequately a concrete computer executing a concrete computation (but noting the possibly differences between classical and quantum for complexity theory). This abstraction of computability and computational complexity theory is perfectly unproblematic when the latter are considered as branches of mathematics. But their application to putative concrete computers and computations demands an account of how their abstract objects adequately represent physical objects and processes, or how their descriptions of computation are adequate abstractions from concrete ones. What, in other words, would it take for a physical system to implement a computation?

There is no consensus about the correct account of computational implementation, but it will be helpful for the remainder to keep several different proposals in mind.[12] The *simple mapping* account states, roughly, that a physical system performs a computation when there is a mapping from the sequential states of the system to the computational states of a computational model (say, the state and tape contents of a TM) such that physical state transitions get mapped to computational state transitions. This account is very liberal, in that it designates multitudinous physical processes as implementing multitudinous computations. It is thus often associated with the thesis of (unlimited) *pancomputationalism*, that (nearly) all physical processes implement all (or many non-equivalent) computations, in some sense.[13] Because many take (unlimited) pancomputationalism to be implausible, many other accounts of computational implementation add extra conditions to the mappings of the simple mapping account to make computation less abundant. Causal, counterfactual, and dispositional accounts require that the state transitions support various modal conditions. Semantic and syntactic accounts take seriously the idea of computation as manipulation of meaningful symbols, requiring respectively that the mappings be representational, according to some account of proper representation, or syntactical, according to an account of what it means for states and changes thereof to be syntactically structured. Mechanistic accounts demand that the physical system or process implementing a computation does so in terms of a functional mechanism, an organization of the components of the system suited to the task of manipulating computational vehicles, those components whose states are mapped to computational states.

Regardless of how the issue of computational implementation is settled, it raises the further question of an analog of the CTT for physical computations. Recall from section 1 that the CTT subsumed the extension of an informal concept – effective computability – under a formal one – Turing computability. Effective computability, though vague, concerns in some idealized sense what can be in principle computed by a human agent aided only with simple memory aides such as paper and pencil. Physical computability, by contrast, concerns what can be computed by any physical process made eligible by one of the above accounts. So a physical version of the CTT would subsume a presumably wider set of physical processes under Turing computation, and the converse of a physical version should easily follow from an argument similar to that for the converse of the CTT.

Several versions of a physical CTT have been proposed. Following Piccinini (2011, 2015), it is helpful to distinguish between two classes of physical CTT:

Modest Physical CTT Any function of natural numbers that is physically computable is Turing computable.

Bold Physical CTT Any physical process is Turing computable.

The modest version, like the CTT itself, focuses on the computation of the values of numerical functions. Sometimes Gandy (1980) is interpreted as having advanced such a thesis:

[12]See Piccinini (2017, sec. 2) for a more thorough review of proposals for computational implementation.
[13]See Piccinini (2017, sec. 3) for more on the different types of pancomputationalism.

Gandy's Thesis M Any function of natural numbers computable by a discrete deterministic mechanical assembly (DDMA) is Turing computable.

A DDMA is any physical device of which an adequate theoretical description uses discrete dynamics for finitely many parts of bounded complexity that affect each other only locally and deterministically. Gandy (1980) then proves that under a certain formalization of DDMAs, Thesis M follows. This thesis is physical in the sense that DDMAs are intended to be models for arbitrary machines that humans might construct to aid them in computations. However, unless one has an essentially anthropocentric account of computational implementation, it is less plausible that physical computations are exhausted by such machines. Accordingly, Thesis M sits conceptually between the CTT and the modest physical CTT.

The bold version of the physical CTT requires a bit of interpretation: what does it mean for a process to be computable? Typically, this means that an adequate theoretical description of the physical process can be simulated by a TM, in the sense that there is an injective map from the physical states of the system undergoing dynamical evolution to the computational states of a TM. A version of this thesis has been advocated by Deutsch (1985):

Deutsch's Principle Every finitely realizable physical system can be perfectly simulated by a universal model computing machine operating by finite means.

The qualification "finitely realizable" is made in explicit reference to Gandy's formalization of DDMAs: it demands that, at any time, finitely many parts of bounded complexity change their state according to local dynamics. Nevertheless, Deutsch's Principle outstrips Thesis M in at least three ways: it does not assume discrete dynamics, nor does it restrict its scope to machines or (more generally) systems adequately described as satisfying the formal requirements for DDMAs; and "universal model computing machine operating by finite means" is intended not to pick out TMs in particular, but a broader class whose operation includes stochastic elements. He argues that this class includes any machines computationally equivalent to universal quantum computers.[14] Unlike the standard CTT, the truth of Deutsch's Principle or related versions of the physical CTT could constrain physical theory,[15] ruling out proposals that allowed for the existence of physical systems implementing *hypercomputation*, i.e., the computation of non-Turing computable functions. Various schemes for hypercomputation have been proposed,[16] but it is a matter of controversy whether those proposals are successful at refuting an interesting version of the physical CTT in the context of a plausible account of computational implementation.

5. Landauer's principle and the thermodynamics of computation

Besides providing a possible constraint on physical theory via some version of the physical CTT, computational ideas have also been invoked in explanatory contexts in thermal physics

[14]Deutsch claims that universal quantum computers outstrip the simulating power of TMs at least because of quantum non-locality, but there is no reason why non-local correlations must be simulated by non-local correlations on a classical stochastic system.

[15]For descriptions of some of these, see Piccinini (2017, sec. 4).

[16]See Piccinini (2017, sec. 4.3) for an overview and further references.

and in investigations of whether a physical computation requires some entropic or energetic cost solely in virtue of its formal properties. Perhaps the main historical origin for these connections is in Maxwell's demon, a thought experiment originally intended to illustrate how an atomic foundation for thermodynamics would entail only statistical validity for its second law – the impossibility of cyclically extracting work solely from the heat energy of a thermal body. In that thought experiment, Maxwell (1871) considered an insulated box of gas separated in two by a partition with a hole just large enough for one gas molecule to pass through. An intelligent demon, equipped with a frictionless shutter for the hole, observes the gas and, whenever a faster-than-average molecule approaches the hole from, say, the left-hand side of the box, opens the shutter to allow it to pass, and similarly for slower-than-average molecules approaching from the right-hand side. Continuing in this way, the gas on the right-hand side of the box becomes hotter and that on the left-hand side cooler, the resulting temperature difference becoming exploitable for extracting work (i.e., by pushing a piston).

Szilard's (1972 [1929]) analysis of this thought experiment focused on what it would take to save the validity of the second law. He considered a simplified version thereof in which the box was not insulated but in contact with a thermal bath and contained only a single molecule of gas. As the molecule bounces around, energetically equilibrating with the walls, an intelligence can frictionlessly place a movable partition in the middle of the box, configured so that when the gas molecule strikes it, it moves, doing work through a series of pulleys. This requires the intelligence to know which side of the box the gas molecule is on and hence measure the gas, so Szilard proposed an energetic cost of $kT \ln 2$, where T is the temperature and k is Boltzmann's constant, to measurement that would balance out the maximum work extractable. Once the energy loss associated with the act of measurement is taken into account, he suggested, the second law is not violated.

Landauer (1961) reversed the operation of Szilard's "engine" to argue that "erasing" a memory unit rather than reading (measuring) it has a minimum energetic cost of $kT \ln 2$. To see how this works, note that one can view the one-molecule gas with the partition as a kind of one-bit memory: if the gas molecule is on the left (resp. right), then the memory reads "0" (resp. "1"). Regardless of the location of the gas molecule, the memory may be reset to "0" by removing the partition, inserting it on the right-hand side of the box, and then doing any work necessary to push it against the gas molecule to the center. What is deemed essential to the "reset" is that the computational process it implements is irreversible, i.e., *not* an injective map on computational states. Thus any adequate physical implementation of the computation cannot be injective either. Landauer argued – for what has become known as his eponymous principle – that the energetic costs to resetting a bit in any physical implementation of a computation must be bounded below by this amount, $kT \ln 2$. Assuming the validity of the second law of thermodynamics (instead of using Landauer's principle to save it), no cyclic process can convert heat energy solely into work, so any thermodynamical process associated with resetting a bit must also generate a minimum amount of heat. Connecting computational processes and concepts of information with thermodynamics, Landauer's principle remains as provocative as it is controversial.[17]

[17]See Maroney (2009a) for a more thorough review of the controversial issues.

6. Chapter summaries

Part I: The computability of physical systems and physical systems as computers

Part I (Chapters 1–3) of the present volume addresses the relationship between physical systems and computational systems: In what senses are physical systems computable, and which of them are computers or perform computations? The widest positive answer to the latter question – namely, *all* physical systems – is found in the astounding thesis of pancomputationalism (cf. Section 4 of this Introduction). In "Ontic pancomputationalism," Gualtiero Piccinini and Neal G. Anderson clarify and consider arguments for the titular strong form of this thesis, which asserts that all physical systems literally and fundamentally perform a computation. That is to say, according to ontic pancomputationalism, the most basic description and dynamics of a physical system is as a computational system performing a single, specific computation. Piccinini and Anderson point out that ontic pancomputationalism also entails an empirical thesis, that a computational formalism ought to be adequate for describing the physical world. The demands on this formalism depend on whether the computational model is discrete or quantum, both of which Piccinini and Anderson review. They argue that, in spite of the provocative nature of ontic pancomputationalism, its empirical aspects are not well supported by our current evidence, while its metaphysical aspects either collapse into triviality or cannot explain the variety of physical systems actually observed.

Jack Copeland, Oron Shagrir, and Mark Sprevak continue the discussion of the relation between physical and computational systems in their chapter, "Zuse's thesis, Gandy's thesis, and Penrose's thesis." The titular three theses concern various ways in which physical systems are or are not supposed to be computers or computable. Concerning Zuse's thesis, which states that the universe literally is a digital computer – in particular, a cellular automaton – Copeland et al. focus on what they call the implementation problem: What kind of ontology could the hardware implementing the universe's computational process have? Surveying four options to the problem – novel (or even non-physical) entities, instrumentalism, anti-realism, and epistemic humility – they conclude that each is unsatisfactory, either failing to answer the question or doing so only by the exorbitant postulation of new ontology. Gandy's thesis, meanwhile, concerns the computational capabilities of machines (cf. Section 4 of this Introduction). Copeland et al. show that Gandy's axiomatic formalization of machines as DDMAs contains a hidden premise asserting a strong form of determinism, effectively ruling out hypercomputational machines operating in relativistic spacetimes. Thus it is not even successful in characterizing the computability of simple sorts of machines. Finally, they consider the oppositely minded Penrose's thesis, which asserts that the processes of the brain are *not* Turing computable. Penrose's argument for his thesis depends upon a controversial application of Gödel's incompleteness theorems. Copeland et al. point out that if Penrose's argument form is valid, then it in fact can be used to prove a much stronger thesis, that the computational power of the human mind exceeds that of any hypercomputer. They show that this *reductio ad absurdum* holds even when certain plausible modifications are made to the thesis. But regardless of rationalistic arguments made to support it, they observe, it is ultimately an empirical hypothesis for which we have yet little evidence.

Rossella Lupacchini continues the discussion of the computability of physical systems in

"Church's thesis, Turing's limits, and Deutsch's principle" by tracing some of the historical and conceptual threads that lead from Hilbert's program for the foundations of mathematics through the CTT on to Deutsch's principle (for which see Sections 1 and 4 of this Introduction, respectively). She shows how Hilbert's influential quest for objective understanding in mathematics through the formalization of metamathematical notions such as "proof" engaged Gödel, Herbrand, and Church in their early attempts at formalizing the concept of effective calculability. It was only with the work of Post and Turing, which shifted focus from defining which functions are supposed to be computable to which (calculative) processes are supposed to be effective, that the connection between computability and physical processes (however idealized) that could be harnessed by human users was made secure. As Post wrote, "to mask this identification [of effective calculability with recursiveness or λ-definability] under a definition hides the fact that a fundamental discovery in the limitations of the mathematicizing [*sic*] power of *Homo Sapiens* has been made and blinds us to the need of its continual verification" (Post 1936, p. 291). This perspective makes Deutsch's Principle – and his suggestion that a quantum computer ought to be the model for a universal finite computer – quite natural once one substitutes a quantum physical substratum for computers for a classical one.

Part II: The implementation of computation in physical systems

Chapters 4–6 of the volume concern issues related to the way that computations are implemented in physical systems. In his chapter "How to make orthogonal positions parallel: Revisiting the quantum parallelism thesis," Armond Duwell begins this part of the book with a discussion of *quantum computation*, that is, computation as it is implemented in quantum mechanical systems and that takes advantage of the particular physical resources that those systems provide. As was mentioned earlier, there is actually no consensus regarding which physical resources are responsible for quantum speedup. Duwell considers two rival explanations. The first, the *quantum parallelism thesis* (Duwell 2007a), asserts that quantum computers are able to outperform classical computers by computing many values of a function simultaneously. Seemingly opposed to this is the idea that quantum computers can outperform classical computers because the quantum logic associated with the state space instantiated by a quantum system allows it to complete a computational task by performing fewer, not more, computations than a classical system (Bub 2010).

Appealing to Gualtiero Piccinini's (2015) *mechanistic conception* of computation, Duwell argues that the seemingly opposite orientations of these positions stem from conflicting intuitions about how to appropriately describe the quantum systems that perform computational tasks. He argues that these positions do not disagree, however, about the fundamental features of quantum systems that give rise to quantum speedup. Guided by this insight, Duwell argues that the quantum parallelism thesis can be formulated in a way that is both true and does not appeal to controversial computational descriptions.

In "How is there a physics of information? On characterizing physical evolution as information processing," Owen J. E. Maroney and Christopher G. Timpson continue Part II by focusing on the field called "The Physics of Information." One of the core claims of this field is that for every information-processing task there exists a fundamental physical resource cost that is associated with it intrinsically. A significant challenge for the Physics of Information is that the existence of alternative physical models of information processing seems, in fact, to make it impossible to associate such intrinsic physical resource costs. Rather, the existence of alternative models seems to show that the details of physical instantiation cannot be ignored. This threatens to undermine the very basis of an implementation-independent Physics of Information.

To make sense of intrinsic resource cost claims, Maroney and Timpson propose a five-fold criterion for determining when an information-processing task has been physically instantiated: (i) the task's logical states are adequately represented by physical states; (ii) these physical states can be reliably initialized, and in fact have been on any particular occasion in which a computation can be said to have taken place; (iii) the physical states evolve equivalently, in a relevant sense, to the logical states they represent; (iv) the final physical output of the task fixes its logical output and is of a kind that is readable to someone; and (v) the process exhibits a certain amount of error tolerance. Maroney and Timpson take criteria (ii) and (iv) to be especially crucial, and on that basis argue that there is a Physics of Information, not because information itself is physical, but because physically embodied agents are able to carry out information-processing tasks.

Part II ends with "Abstraction/Representation theory and the natural science of computation," by Dominic Horsman, Viv Kendon, and Susan Stepney. As with the previous chapter, Horsman et al. aim to present a number of criteria for determining when a physical process instantiates a computation. However they also argue that these criteria unite computation with prediction and engineering. Their framework, *Abstraction/Representation (AR) theory*, contains the following components: scientific theory, computational theory, encoding and decoding of a scientific problem in a computational model, and instantiation of a computational problem in a physical system. These components take on different forms depending on the kind of computation one is discussing, so that AR theory is capable of describing vastly different kinds of physical computation. For example, it can describe classical digital computation, including how transistors and other hardware components implement classical computational models such as the von Neumann architecture, and how programming languages, compilers, and other software components can make use of this hardware. And it can also describe the unconventional model of "slime-mold computation," wherein how a slime mold reacts to food sources being placed in its vicinity implements the abstract problem of finding the shortest path through a maze.

According to AR theory, physically carrying out a computation is analogous to doing science, in the sense that both practices involve a representational relation between physical objects and abstract mathematico-logical objects. In each case this presupposes a representational entity. In the natural sciences this can be an experimenter, or perhaps a theorist. In the case of a computational process this can be, for example, a programmer, high-level designer, or end-user. In seeming contrast to the view of Maroney and Timpson, however, the representational entity need not embody a rich conception of agency.

Part III: Physical perspectives on computer science

Chapters 7–9 of this volume concern the question of how physical theory can illuminate the theory of computation. In "Physics-like models of computation," Klaus Sutner argues that insight into the nature of computability theory can be gained by carefully studying what he refers to as “physics-like” computational models. Such models have been constructed with an eye to their possible physical realization and include, for example, the ordered partition automaton and its variants. Sutner’s aim is not to argue that computer science should become a part of physics. He rather makes the compelling argument that the study of physical computation is potentially very fruitful for computer science, and that it has not received sufficient attention to date.

Sutner contrasts such studies of physics-like computation with classical recursion theory. He argues that classical recursion theory is not only isolated from physics but also from much of mathematics (including discrete mathematics), in that there has been little exchange of results and techniques between the two disciplines. On the other hand, Sutner argues that there are problems in the theory of computation that stand to profit much from the study of physical computation. He shows, in particular, how a physics-like model of computation (the elementary cellular automaton number 110) produced the first instance of a universal system that was discovered, rather than designed. He also shows how considering the issue of physical realization leads to insight into the old problem of the epistemological status of intermediate recursively enumerable degrees.

In the next chapter, “Feasible computation: Methodological contributions from computational science,” Robert H. C. Moir explains how, in contemporary usage, “computational science” has come to refer to a number of different forms of scientific computing. In essence, computational science in this sense is concerned with generating “feasible algorithms” for solving mathematical problems, and unlike traditional computability theory, it can involve numerical methods and hybrid symbolic-numeric methods in addition to symbolic computation. Also, and importantly, unlike traditional computability theory, the feasible algorithms of computational science generally involve the use of approximation extensively.

Moir describes, at length, how the historical roots of approximation methods in computational science can be traced back to the eighteenth- and nineteenth-century techniques developed by physicists to overcome the calculational limitations of their theoretical representations of physical phenomena (for example, in fluid mechanics and astronomy). Moir shows how the use of such approximation methods allowed physicists to extract crucial information from their theoretical models even when exact solutions were not at hand.

Moir reveals how the central strategy used in computational science for developing approximation algorithms itself has an algorithmic structure, and that a version of this strategy underlies symbolic computing. This has consequences for traditional computability theory; Moir argues that it motivates the development of a theory of “higher-order” computation, leads to a different way of thinking about computational complexity, and points the way to a possible expansion of the theory of computation towards encompassing aspects of the methodology and epistemology of scientific inference.

Part III of the book ends with “Relativistic computation,” by Hajnal Andréka, Judit X. Madarász, István Németi, Péter Németi, and Gergely Székely. Andréka et al. describe the challenge to the physical CTT that is posed by computational systems which take advantage of the possibilities inherent in the spacetimes allowed by general relativity.

In particular, they describe a thought experiment involving a physical computer composed of a TM, on the one hand, and a spaceship carrying a programmer, on the other, operating in the general relativistic spacetime associated with a huge slowly rotating black hole. Features of the spacetime associated with such a black hole make it so that our programmer can survive entry into its inner event horizon intact. Because of the gravitational time dilation effect of general relativity, which causes clocks in stronger gravitational fields to run more slowly, in a sense, than those in weaker such fields, we can design our computational system so that, in the finite time before the programmer enters the inner event horizon, she is able to receive the results of an infinitely long computation carried out by the distant TM.

Fascinatingly, Andréka et al. argue that such a computer could be realized by a conceivable future civilization. In particular, they argue that such a computer could be made error-tolerant, that limitations with respect to the transfer of information between the TM and the programmer can be overcome, and that the spacetimes which make possible such a computational system are physically realistic.

Part IV: Computational perspectives on physical theory

Part IV (Chapters 10–12) concerns the potential illumination – and distraction – that may result from considering physical theory from a computational perspective. In particular, the first two chapters of this part concern the status of Landauer's principle (which is adumbrated in Section 5 of this Introduction). James Ladyman brings accounts of computational implementation to bear on the debate over whether Landauer's principle follows from the second law of thermodynamics in "Intension in the physics of computation: Lessons from the debate about Landauer's principle." Ladyman's focus is on the dialectic between the results of Ladyman et al. (2007), which seek to prove a general, sound version of Landauer's principle, and the critique thereof by Norton (2011). In particular, Ladyman argues that Norton's rebuttal to the proof can be blocked once one requires that the implementation of a computation in a physical system be intensional, or modally robust: It must support counterfactuals describing that the same function or logical operation would have been computed, but with a different input, had the initial state of the physical system been different. This also requires a non-dynamical (or "control theory") perspective on thermodynamics that views the theory – and Landauer's Principle – as concerning what human agents can do to systems with their typically limited knowledge of the system's detailed microstates. Thus, an account of computation that concerns human abilities and knowledge reveals how thermodynamics is bound in similar ways.

In contrast with Ladyman, John D. Norton describes how, in the literature on Maxwell's demon (also described in Section 5 of this Introduction), the historical focus on exorcising the demon by considering its computational powers has been a mistake. In particular, he argues that this focus has distracted us, for decades, from a simpler and more satisfying solution which only makes use of concepts from physical theory. His punning title puts it simply: "Maxwell's demon does not compute." Norton begins by reviewing the history of the demon and its naturalization – versions which use an explicit physical mechanism instead of the fanciful demon – arguing that such naturalized demons, lacking any obvious embodiment of intelligent or computational capacity, controvert claims to a general exorcism using computational ideas. Despite this, influential work by Szilard (1972 [1929]) and Landauer (1961) drew the physics community's attention away from more directly physical ideas towards more speculative ones that, Norton's argues, have produced more heat than light. Norton then extends previous work (Norton 2013a) exorcising the classical demon using Liouville's theorem to a quantum version, comparing the steps of the arguments in detail. Essential to the argument is an analogy between phase space volume, in the classical case, and Hilbert subspace dimension, in the quantum case. If a putative naturalized quantum demon is to act on an equilibrium state occupying a subspace of dimension almost as large as that of its complete Hilbert space, and the intermediate states into which the demon drives the system occupy instead a small subspace, then the demon will always fail in its task.

In the final chapter, "Quantum theory as a principle theory: Insights from an information-theoretic reconstruction," Adam Koberinski and Markus P. Müller argue that recasting physical theory in terms of principles about information and computation yields additional explanatory power about the nature of the (quantum) physical world. In particular, they explicate a recent derivation of finite-dimensional quantum theory from the following principles: the state and time evolution of every physical system can be reversibly encoded into a number of interacting "universal bits"; the time evolution of these systems is reversible and continuous; states of composite systems are uniquely determined by those of their components and the correlations between them; and one universal bit can carry no more than one binary unit of information. This principle-theory framework (*sensu* Einstein [1954]) they argue, provides a *partial* interpretation of quantum theory in the sense that it describes *in part* what the world would be like if the theory were true: The world is fundamentally restricted with respect to how physical systems can transform information (i.e., perform computations). It is only partial, however, at least because it is agnostic on the interpretation of the quantum state. Accordingly, Koberinski and Müller outline three options for completing the interpretation, but argue that each one poses a challenge for interpreting the quantum state as representing a feature of the world rather than a feature of our knowledge about the world (i.e., for so-called ψ-ontic interpretations rather than ψ-epistemic ones). They conclude by suggesting that the principle of continuous reversible time evolution should be given more attention than it has heretofore received, as it may prove surprisingly central in future efforts to characterize what makes quantum theories different from classical ones.