Chapter I

Introduction

A   What is unconventional computation?

Unconventional computation and its synonym, *nonstandard computation*, are negative terms; they refer to computation that is not "conventional" or "standard." Conventional computation is the kind we are all familiar with, which has dominated computer science since the 1950s. We can enumerate its common characteristics:

- Information representation and processing is digital and in fact binary.

- Computers are organized according to the von Neumann architecture, which separates memory from a processor that fetches data from memory for processing and then returns it to memory.

- Memory is organized as one or more arrays of units (bytes or words) indexed by natural numbers.

- The processor executes instructions from memory, sequentially, in discrete steps, and one at a time; therefore instructions are encoded in binary.

- The sequential binary logic of the processor is implemented by means of electronically-controlled electronic switches.

- Programs are written as hierarchically organized sequences of instructions, which perform computation, input, output, and control of program execution.

- Programs can decide between alternative sequences of instructions.

- Programs can repeat sequences of instructions until some condition is satisfied.

- Programs can be hierarchically organized into subprograms, subprograms of subprograms, etc.

Unconventional computers, then, are different in at least one of these characteristics. For example, analog computers represent information by continuous physical quantities rather than bits, and some analog computers operate in continuous time (obeying differential equations) rather than in discrete sequential steps. Nonelectronic computers may use light, fluids, microorganisms, or DNA and other molecules as a computational medium. In molecular computers, vast numbers of operations take place asynchronously in parallel. Ballistic computers implement reversible computation, so they may dissipate arbitrarily small amounts of energy, but cannot use common programming operations, such as storing into a memory. Functional programming does not use rewritable variables, and computers designed for functional programming, such as reduction machines, do not have a programmer-accessible storable memory. Quantum computers do not do one thing at a time, but execute an exponentially large number of operations simultaneously in quantum superposition. Alternative computational paths can be executed in superposition and otherwise iterative processes can be executed all at once. Neuromorphic computers, which are inspired by the brain, don't have an addressible memory in the usual sense (they learn by analog changes in vast numbers of connection strengths, so that individual memories are distributed over large overlapping sets of connections). They are also massively parallel and use analog computation. Programs for unconventional computers can take unconventional forms, such as recursive function definitions, ordinary or partial differential equations, quantum operations, neural network architectures, sequences of chemical reactions, genetic regulatory networks, or even continuous images. These are just a few of the ways in which computation can be "unconventional" or "nonstandard." You may be asking, however, why we would want to compute in these unconventional ways, given the outstanding success of conventional computation. That is our next topic.
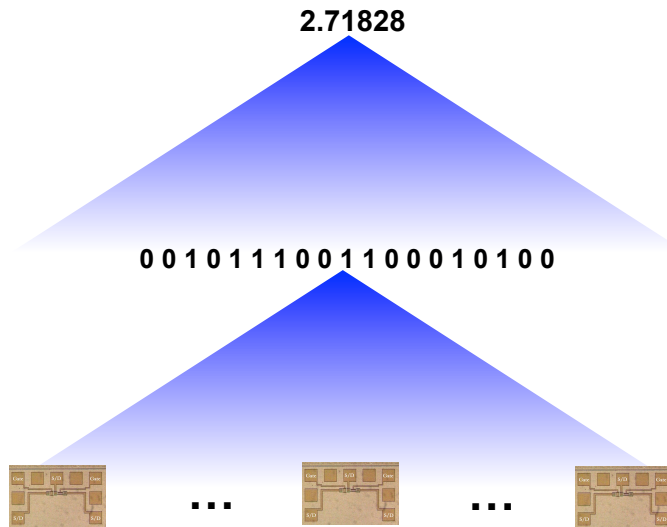
Figure I.1: Hierarchy of spatial scales in conventional computing.

B   Post-Moore's Law computing

Although estimates differ, it is clear that the end of Moore's Law is in sight; there are physical limits to the density of binary logic devices and to their speed of operation.[1] This will require us to approach computation in new ways, which will present significant challenges, but can also broaden and deepen our concept of computation in natural and artificial systems.

In the past there has been a significant difference in scales between computational processes and the physical processes by which they are realized. For example, there are differences in spatial scale: the data with which programs operate (integers, floating point numbers, characters, pointers, etc.) are represented by large numbers of physical devices comprising even larger numbers of physical particles (Fig. I.1). Also, there are differences in time scale: elementary computational operations (arithmetic, instruction sequencing, memory access, etc.), are the result of large numbers of state changes at the device level (typically involving a device moving from one saturated state to another) (Fig. I.2). However, increasing the density and speed of

---

[1]This section is adapted from MacLennan (2008). Moore's Law was presented in Moore (1965).
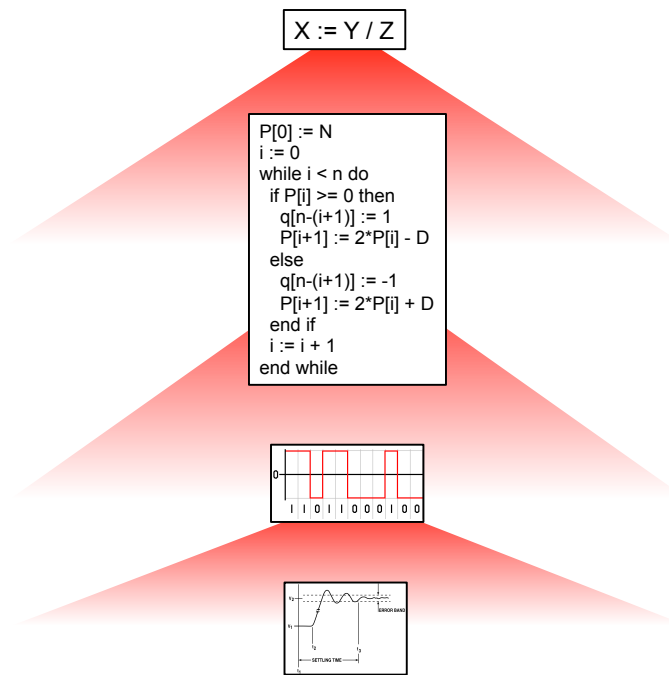
Figure I.2: Hierarchy of temporal scales in conventional computing.

computation will force it to take place on the scale (spatial and temporal) near that of the underlying physical processes. With fewer hierarchical levels between computations and their physical realizations, and less time for implementing computational processes, computation will have to become more like the underlying physical processes. That is, *post-Moore's Law computing* will depend on a greater assimilation of computation to physics.

In discussing the role of physical embodiment in the "grand challenge" of unconventional computing, Stepney (2004, p 29) writes,

> Computation is physical; it is necessarily embodied in a device whose behaviour is guided by the laws of physics and cannot be completely captured by a closed mathematical model. This fact of embodiment is becoming ever more apparent as we push the bounds of those physical laws.

Traditionally, a sort of Cartesian dualism has reigned in computer science; programs and algorithms have been conceived as idealized mathematical objects; software has been developed, explained, and analyzed independently of hardware; the focus has been on the *formal* rather than the *material*. Post-Moore's Law computing, in contrast, because of its greater assimilation to physics, will be less idealized, less independent of its physical realization. On one hand, this will increase the difficulty of programming since it will be dependent on (or, some might say, contaminated by) physical concerns. On the other hand, as will be explored in this book, it also presents many opportunities that will contribute to our understanding and application of information processing in the future.

C Super-Turing vs. non-Turing

In addition to the *practical* issues of producing more powerful computers, unconventional computation poses the question of its *theoretical* power.[2] Might some forms of unconventional computation provide a means of *super-Turing computation* (also called *hypercomputation*), that is, of computation beyond what is possible on a Turing machine? This is an important theoretical question, but obsession with it can divert attention from more significant issues in unconventional computing. Therefore it is worth addressing it before we get into specific unconventional computing paradigms.

_____

[2]This section is adapted from MacLennan (2009b).

C.1   The limits of Turing computation

C.1.a   Frames of Relevance

It is important to remember that Church-Turing (CT) computation is a *model* of computation, and that computation is a physical process taking place in certain physical objects (such as computers). Models are intended to help us understand some class of phenomena, and they accomplish this by making *simplifying assumptions* (typically *idealizing assumptions*, which omit physical details taken to be of secondary importance). For example, we might use a linear mathematical model of a physical process even though we know that its dynamics is only approximately linear; or a fluid might be modeled as infinitely divisible, although we know it is composed of discrete molecules. We are familiar also with the fact that several models may be used with a single system, each model suited to understanding certain aspects of the system but not others. For example, a circuit diagram shows the electrical interconnections among components (qua electrical devices), and a layout diagram shows the components' sizes, shapes, spatial relationships, etc.

As a consequence of its simplifying assumptions, each model comes with a (generally unstated) *frame of relevance*, which delimits (often vaguely) the questions that the model can answer accurately. For example, it would be a mistake to draw conclusions from a circuit diagram about the size, shape, or physical placement of circuit components. Conversely, little can be inferred about the electrical properties of a circuit from a layout diagram.

Within a (useful) model's frame of relevance, its simplifying assumptions are sensible (e.g., they are good approximations); outside of it they may not be. That is, within its frame of relevance a model will give us good answers (not necessarily 100% correct) and help us to understand the characteristics of the system that are most relevant *in that frame*. Outside of its intended frame, a model might give good answers (showing that its *actual* frame can be larger than its *intended* frame), but we cannot assume that to be so. Outside of its frame, the answers provided by a model may reflect the simplifying assumptions of the model more than the system being modeled. For example, in the frame of relevance of macroscopic volumes, fluids are commonly modeled as infinitely divisible continua (an idealizing assumption), but if we apply such a model to microscopic (i.e., molecular scale) volumes, we will get misleading answers, which are a consequence of the simplifying assumptions.

C.1.b   The frame of relevance of Church-Turing computation

It is important to explicate the frame of relevance of Church-Turing (CT) computation, by which I mean not just Turing machines, but also equivalent models of computation, such as the lambda calculus and Post productions, as well as other more or less powerful models based on similar assumptions (discussed below). (Note however that the familiar notions of equivalence and power are themselves dependent on the frame of relevance of these models, as will be discussed.) The CT frame of relevance becomes apparent if we recall the original questions the model was intended to answer, namely questions of effective calculability and formal derivability. As is well known, the CT model arises from an idealized description of what a mathematician could do with pencil and paper. Although a full analysis of the CT frame of relevance is beyond the scope of this chapter (but see MacLennan, 1994b, 2003, 2004), I will mention a few of the idealizing assumptions.

Within the CT frame of relevance, something is computable if it can be computed with finite but unbounded resources (e.g., time, memory). This is a reasonable idealizing assumption for answering questions about formal derivability, since we don't want our notion of a proof to be limited in length or "width" (size of the formal propositions). It is also a reasonable simplifying assumption for investigating the limits of effective calculability, which is a idealized model of arithmetic with paper and pencil. Again, in the context of the formalist research program in mathematics, there was no reason to place an a priori limit on the number of steps or the amount of paper (or pencil lead!) required. Note that these *are* idealizing assumptions: so far as we know, physical resources are not unbounded, but these bounds were not considered relevant to the questions that the CT model was originally intended to address; in this frame of relevance "finite but unbounded" is a good idealization of "too large to be worth worrying about."

Both formal derivation and effective calculation make use of finite formulas composed of discrete tokens, of a finite number of types, arranged in definite structures (e.g., strings) built up according to a finite number of primitive structural relationships (e.g., left-right adjacency). It is further assumed that the types of the tokens are positively determinable, as are the primitive interrelationships among them. Thus, in particular, we assume that there is no uncertainty in determining whether a token is present, whether a configuration is one token or more than one, what is a token's type, or how the tokens are arranged, and we assume that they can be rearranged with

perfect accuracy according to the rules of derivation. These are reasonable assumptions in the study of formal mathematics and effective calculability, but it is important to realize that they are *idealizing assumptions*, for even mathematicians can make mistakes in reading and copying formulas and in applying formal rules!

Many of these assumptions are captured by the idea of a *calculus*, but a phenomenological analysis of this concept is necessary to reveal its background of assumptions (MacLennan, 1994b). Briefly, we may state that both information representation and information processing are assumed to be *formal*, *finite*, and *definite* (MacLennan, 2003, 2004). These and other assumptions are taken for granted by the CT model because they are reasonable in the context of formal mathematics and effective calculability. Although the originators of the model discussed some of these simplifying assumptions (e.g., Markov, 1961), many people today do not think of them as assumptions at all, or consider that they might not be appropriate in some other frames of relevance.

It is important to mention the concept of time presupposed in the CT model, for it is not discrete time in the familiar sense in which each unit of time has the same duration; it is more accurate to call it *sequential time* (van Gelder, 1997). This is because the CT model does not take into consideration the time required by an individual step in a derivation or calculation, so long as it is finite. Therefore, while we can *count* the number of steps, we cannot translate that count into real time, since the individual steps have no definite duration. As a consequence, the only reasonable way to compare the time required by computational processes is in terms of their asymptotic behavior. Again, sequential time is reasonable in a model of formal derivability or effective calculability, since the time required for individual operations was not relevant to the research programme of formalist mathematics (that is, the time was irrelevant in that frame of relevance), but it can be very relevant in other contexts, as will be discussed.

Finally I will mention a simplifying assumption of the CT model that is especially relevant to hypercomputation, namely, the assumption that computation is equivalent to evaluating a well-defined function on an argument. Certainly, the mathematical function, in the full generality of its definition, is a powerful and versatile mathematical concept. Almost any mathematical object can be treated as a function, and functions are essential to the description of processes and change in the physical sciences. Therefore, it was natural, in the context of the formalist program, to focus on functions in

the investigation of effective calculation and derivation. Furthermore, many early applications of computers amounted to function evaluations: you put in a deck of cards or mounted a paper or magnetic tape, started the program, it computed for awhile, and when it stopped you had an output in the form of cards, tape, or printed paper. Input — compute — output, that was all there was to it. If you ran the program again with a different input, that amounted to an independent function evaluation. The only relevant aspect of a program's behavior was the input-output correspondence (i.e., the mathematical function).

This view can be contrasted with newer ones in which, for example, a computation involves continuous, non-terminating interaction with its environment, such as might be found in control systems and autonomous robotics. Some new models of computation have moved away from the idea of computation as the evaluation of a fixed function (Eberbach et al., 2003; Milner, 1993; Milner et al., 1992; Wegner, 1997, 1998; Wegner & Goldin, 2003).

However, in the CT frame of relevance the natural way to compare the "power" of models of computation was in terms of the classes of functions they could compute, a single dimension of power now generalized into a partial order of set inclusions (but still based on a single conception of power: computing a class of functions).[3]

C.2 New computational models

A reasonable position, which many people take explicitly or implicitly, is that the CT model is a perfectly adequate model of everything we mean by "computation," and therefore that any answers that it affords us are definitive. However, as we have seen, the CT model exists in a frame of relevance, which delimits the kinds of questions that it can answer accurately, and, as I will show, there are important computational questions that fall outside this frame of relevance.

C.2.a NATURAL COMPUTATION

Natural computation may be defined as computation occurring in nature or inspired by computation in nature. The information processing and control

---

[3]I note in passing that this approach raises all sorts of knotty cardinality questions, which are inevitable when we deal with such "large" classes; therefore in some cases results depend on a particular axiomatization or philosophy of mathematics.

that occurs in the brain is perhaps the most familiar example of computation in nature, but there are many others, such as the distributed and self-organized computation by which social insects solve complicated optimization problems and construct sophisticated, highly structured nests. Also, the DNA of multicellular organisms defines a developmental program that creates the detailed and complex structure of the adult organism. For examples of computation inspired by that in nature, we may cite artificial neural networks, genetic algorithms, artificial immune systems, and ant swarm optimization, to name just a few. Next I will consider a few of the issues that are important in natural computation, but outside the frame of relevance of the CT model.

One of the most obvious issues is that, because computation in nature serves an adaptive purpose, it must satisfy stringent *real-time constraints*. For example, an animal's nervous system must respond to a stimulus — fight or flight, for example — in a fraction of a second. Also, in order to control coordinated sensorimotor behavior, the nervous system has to be able to process sensory and proprioceptive inputs quickly enough to generate effector control signals at a rate appropriate to the behavior. And an ant colony must be able to allocate workers appropriately to various tasks in real time in order to maintain the health of the colony.

In nature, asymptotic complexity is generally irrelevant; the constants matter and input size is generally fixed or varies over a relatively limited range (e.g., numbers of sensory receptors, colony size). Whether the algorithm is linear, quadratic, or exponential is not so important as whether it can deliver useful results in required real-time bounds in the cases that actually occur. The same applies to other computational resources. For example, it is not so important whether the number of neurons required varies linearly or quadratically with the number of inputs to the network; what matters is the absolute number of neurons required for the actual number of inputs, and how well the system will perform with the number of inputs and neurons it actually has.

Therefore, in natural computation, what does matter is how the real-time response rate of the system is related to the real-time rates of its components (e.g., neurons, ants) and to the actual number of components. This means that it is not adequate to treat basic computational processes as having an indeterminate duration or speed, as is commonly done in the CT model. In the natural-computation frame of relevance, knowing that a computation will *eventually* produce a correct result using *finite but unbounded resources*

is largely irrelevant. The question is whether it will produce a *good-enough* result using available resources subject to real-time constraints.

Many of the inputs and outputs to natural computation are *continuous in magnitude* and *vary continuously* in real time (e.g., intensities, concentrations, forces, spatial relations). Many of the computational processes are also continuous, operating in continuous real time on continuous quantities (e.g., neural firing frequencies and phases, dendritic electrical signals, protein synthesis rates, metabolic rates). Obviously these real variables can be approximated arbitrarily closely by discrete quantities, but that is largely irrelevant in the natural-computation frame of relevance. The most natural way to model these systems is in terms of continuous quantities and processes.

If the answers to questions in natural computation seem to depend on "metaphysical issues," such as whether only Turing-computable reals exist, or whether all the reals of standard analysis exist, or whether non-standard reals exist, then I think that is a sign that we are out of the model's frame of relevance, and that the answers are more indicative of the model itself than of the modeled natural-computation system. For models of natural computation, naive real analysis, like that commonly used in science and engineering, should be more than adequate; it seems unlikely that disputes in the foundations of mathematics will be relevant to our understanding how brains coordinate animal behavior, how ants and wasps organize their nests, how embryos self-organize, and so forth.

C.2.b CROSS-FRAME COMPARISONS

This example illustrates the more general pitfalls that arise from *cross-frame comparisons*. If two models have different frames of relevance, then they will make different simplifying and idealizing assumptions; for example objects whose existence is assumed in one frame (such as standard real numbers) may not exist in the other (where all objects are computable). Therefore, a comparison requires that one of the models be translated from its own frame to the other (or that both be translated to a third), and, in doing this translation, assumptions compatible with the new frame will have to be made. For example, if we want to investigate the computational power of neural nets in the CT frame (i.e., in terms of classes of functions of the integers), then we will have to decide how to translate the naive continuous variables of the neural net model into objects that exist in the CT frame. For instance, we might choose fixed-point numbers, computable reals (represented in some

way by finite programs), or arbitrary reals (represented by infinite discrete structures). We then discover (as reported in the literature: e.g., Maass & Sontag, 1999a; Siegelmann & Sontag, 1994a), that our conclusions depend on the choice of numerical representation (which is largely irrelevant in the natural-computation frame). That is, our conclusions are more a function of the specifics of the cross-frame translation than of the modeled systems.

Such results tell us nothing about, for example, why brains do some things so much better than do contemporary computers, which are made of much faster components. That is, in the frame of natural computation, the issue of the representation of continuous quantities does not arise, for it is irrelevant to the questions addressed by this frame, but this issue is crucial in the CT frame. Conversely, from within the frame of the CT model, much of what is interesting about neural net models (parallelism, robustness, real-time response) becomes irrelevant. Similar issues arise when the CT model is taken as a benchmark against which to compare unconventional models of computation, such as quantum and molecular computation.

C.2.c RELEVANT ISSUES FOR NATURAL COMPUTATION

We have seen that important issues in the CT frame of relevance, such as asymptotic complexity and the computability of classes of functions, are not so important in natural computation. What, then, are the relevant issues?

One important issue in natural computation is *robustness*, by which I mean effective operation in the presence of noise, uncertainty, imprecision, error, and damage, all of which may affect the computational process as well as its inputs. In the CT model, we assume that a computation should produce an output exactly corresponding to the evaluation of a well-defined function on a precisely specified input; we can, of course, deal with error and uncertainty, but it's generally added as an afterthought. Natural computation is better served by models that incorporate this indefiniteness a priori.

In the CT model, the basic standard of correctness is that a program correctly compute the same outputs as a well-defined function evaluated on inputs in that function's domain. In natural computation, however, we are often concerned with *generality and flexibility*, for example: How well does a natural computation system (such as a neural network) respond to inputs that are *not* in its intended domain (the domain over which it was trained or for which it was designed)? How well does a neural control system respond

to unanticipated inputs or damage to its sensors or effectors? A related issue is *adaptability*: How well does a natural computation system change its behavior (which therefore does not correspond to a fixed function)?

Finally, many natural computation systems are not usefully viewed as computing a function at all. As previously remarked, with a little cleverness anything can be viewed as a function, but this is not the simplest way to treat many natural systems, which often are in open and continuous interaction with their environments and are effectively nonterminating. In natural computation we need to take a more biological view of a computational system's "correctness" (better: *effectiveness*). It will be apparent that the CT model is not particularly well suited to addressing many of these issues, and in a number of cases begs the questions or makes assumptions incompatible with addressing them. Nevertheless, real-time response, generality, flexibility, adaptability, and robustness in the presence of noise, error, and uncertainty are important issues in the frame of relevance of natural computation.

C.2.d NANOCOMPUTATION

Nanocomputation is another domain of computation that seems to be outside the frame of relevance of the CT model. By nanocomputation I mean any computational process involving sub-micron devices and arrangements of information; it includes quantum computation (Ch. III) and *molecular computation* (e.g., DNA computation), in which computation proceeds through molecular interactions and conformational changes (Ch. IV).

Due to thermal noise, quantum effects, etc., *error* and *instability* are unavoidable characteristics of nanostructures. Therefore they must be taken as givens in nanocomputational devices and in their interrelationships (e.g., interconnections), and also in the structures constructed by nanocomputational processes (e.g., in algorithmic self-assembly: Winfree, 1998). Therefore, a "perfect" structure is an over-idealized assumption in the context of nanocomputation; defects are unavoidable. In many cases structures are not fixed, but are stationary states occurring in a system in constant flux. Similarly, unlike in the CT model, nanocomputational operations cannot be assumed to proceed correctly, for the probability of error is always non-negligible. Error cannot be considered a second-order detail added to an assumed perfect computational system, but should be built into a model of nanocomputation from the beginning. Indeed, operation cannot even be assumed to proceed uniformly forward. For example, chemical reactions al-

ways have a non-zero probability of moving backwards, and therefore molecular computation systems must be designed so that they accomplish their purposes in spite of such reversals. This is a fundamental characteristic of molecular computation, which should be an essential part of any model of it.

C.2.e   SUMMARY OF ISSUES

In summary, the notion of super-Turing computation, *stricto sensu*, exists only in the frame of relevance of the Church-Turing model of computation, for the notion of being able to compute "more" than a Turing machine presupposes a particular notion of "power." Although it is interesting and important to investigate where unconventional models of computation fall in this computational hierarchy, it is also important to explore *non*-Turing computation, that is, models of computation with different frames of relevance from the CT model. Several issues arise in the investigation of non-Turing computation: (1) What is computation in the broad sense? (2) What frames of relevance are appropriate to unconventional conceptions of computation (such as natural computation and nanocomputation), and what sorts of models do we need for them? (3) How can we fundamentally incorporate error, uncertainty, imperfection, and reversibility into computational models? (4) How can we systematically exploit new physical processes (quantum, molecular, biological, optical) for computation? The remainder of this chapter addresses issues (1) and (4).

C.3   Computation in general

C.3.a   KINDS OF COMPUTATION

Historically, there have been many kinds of computation, and the existence of alternative frames of relevance shows us the importance of non-Turing models of computation. How, then, can we define "computation" in sufficiently broad terms? Prior to the twentieth century computation involved operations on mathematical objects by means of physical manipulation. The familiar examples are arithmetic operations on numbers, but we are also familiar with the geometric operations on spatial objects of Euclidean geometry, and with logical operations on formal propositions. Modern computers operate on a much wider variety of objects, including character strings, images, sounds, and much else. Therefore, the observation that computation uses physical

processes to accomplish mathematical operations on mathematical objects must be understood in the broadest sense, that is, as abstract operations on abstract objects. In terms of the traditional distinction between *form* and *matter*, we may say that computation uses material states and processes to *realize* (implement) formal operations on abstract forms. But what sorts of physical processes?

C.3.b Effectiveness and mechanism

The concepts of *effectiveness* and *mechanism*, familiar from CT computation, are also relevant to computation in a broader sense, but they must be similarly broadened. To do this, we may consider the two primary uses to which models of computation are put: understanding computation in nature and designing computing devices. In both cases the model relates information representation and processing to underlying physical processes that are considered unproblematic within the frame of relevance of the model.

For example, the CT model sought to understand effective calculability and formal derivability in terms of simple processes of symbol recognition and manipulation, such as are routinely performed by mathematicians. Although these are complex processes from a cognitive science standpoint, they were considered unproblematic in the context of metamathematics. Similarly, in the context of natural computation, we may expect a model of computation to explain intelligent information processing in the brain in terms of electrochemical processes in neurons (considered unproblematic in the context of neural network models). Or we may expect a different model to explain the efficient organization of an ant colony in term of pheromone emission and detection, simple stimulus-response rules, etc. In all these cases the explanation is mechanistic, in the sense that it refers to *primary qualities*, which can be objectively measured or positively determined, as opposed to *secondary qualities*, which are subjective or depend on human judgment, feeling, etc. (all, of course, in the context to the intended purpose of the model); measurements and determinations of primary qualities are *effective* in that their outcomes are reliable and dependable.

A mechanistic physical realization is also essential if a model of computation is to be applied to the design of computing devices. We want to use physical processes that are *effective* in the broad sense that they result reliably in the intended computations. In this regard, electronic binary logic has proved to be an extraordinarily effective mechanism for computation. (In

Sec. C.4.b I will discuss some general effectiveness criteria.)

C.3.c   MULTIPLE REALIZABILITY

Although the forms operated upon by a computation must be materially realized in some way, a characteristic of computation that distinguishes it from other physical processes is that it is independent of *specific* material realization. That is, although a computation must be materially realized in *some* way, it can be realized in *any* physical system having the required formal structure. (Of course, there will be practical differences between different physical realizations, but I will defer consideration of them until later.) Therefore, when we consider computation *qua computation*, we must, on the one hand, restrict our attention to formal structures that are mechanistically realizable, but, on the other, consider the processes independently of any particular mechanistic realization.

These observations provide a basis for determining whether or not a particular physical system (in the brain, for example) is computational (MacLennan, 1994c, 2004). If the system could, in principle at least, be replaced by another physical system having the same formal properties and still accomplish its purpose, then it is reasonable to consider the system computational (because its formal structure is sufficient to fulfill its purpose). On the other hand, if a system can fulfill its purpose only by control of particular substances or particular forms of energy (i.e., it is not independent of a specific material realization), then it cannot be purely computational. (Nevertheless, a computational system will not be able to accomplish its purpose unless it can interface properly with its physical environment; this is a topic I will consider in Sec. C.3.f.)

C.3.d   DEFINING COMPUTATION

Based on the foregoing considerations, we have the following definition of computation (MacLennan, 1994c, 2004, 2009b):

Definition 1 Computation *is a mechanistic process, the purpose of which is to perform abstract operations on abstract objects.*

Alternately, we may say that computation accomplishes the formal transformation of formal objects by means of mechanistic processes operating on the objects' material embodiment. The next definition specifies the relation between the physical and abstract processes:

Definition 2 *A mechanistic physical system* realizes *a computation if, at the level of abstraction appropriate to its purpose, the abstract transformation of the abstract objects is a sufficiently accurate model of the physical process. Such a physical system is called a* realization *of the computation.*

That is, the physical system realizes the computation if we can see the material process as a sufficiently accurate embodiment of the formal structure, where the sufficiency of the accuracy must be evaluated in the context of the system's purpose. Mathematically, we may say that there is a homomorphism from the physical system to the abstract system, because the abstract system has some, but not all, of the formal properties of the physical system (MacLennan, 1994a, 2004). The next definition classifies various systems, both natural and artificial, as computational:

Definition 3 *A physical system is* computational *if its function (purpose) is to realize a computation.*

Definition 4 *A* computer *is an artificial computational system.*

Thus the term "computer" is restricted to intentionally manufactured computational devices; to call the brain a computer is a metaphor. These definitions raise a number of issues, which I will discuss briefly; no doubt the definitions can be improved.

C.3.e PURPOSE

First, these definitions make reference to the *function* or *purpose* of a system, but philosophers and scientists are justifiably wary of appeals to purpose, especially in a biological context. However, the use of purpose in the definition of computation is unproblematic, for in most cases of practical interest, purpose is easy to establish. (There are, of course, borderline cases, but that fact does not invalidate the definition.) On the one hand, in a technological context, we can look to the stated purpose for which an artificial system was designed. On the other, in a biological context, scientists routinely investigate the purposes (functions) of biological systems, such as the digestive system and immune system, and make empirically testable hypotheses about their purposes. Ultimately such claims of biological purpose may be reduced to a system's selective advantage to a particular species in that species' environment of evolutionary adaptedness, but in most cases we can appeal to more immediate ideas of purpose.

On this basis we may identify many natural computational systems. For example, the function of the brain is primarily computational (in the sense used here), which is easiest to see in sensory areas. For example, there is considerable evidence that an important function of primary visual cortex is to perform a Gabor wavelet transform on visual data (Daugman, 1993); this is an abstract operation that could, in principal, be realized by a non-neural physical system (such as a computer chip). Also, pheromone-mediated interactions among insects in colonies often realize computational ends such as allocation of workers to tasks and optimization of trails to food sources. Likewise DNA transcription, translation, replication, repair, etc., are primarily computational processes.

However, there is a complication that arises in biology and can be expected to arise in our biologically-inspired robots and more generally in post-Moore's Law computing. That is, while the distinction between computational and non-computational systems is significant to *us*, it does not seem to be especially significant to *biology*. The reason may be that we are concerned with the *multiple realizability* of computations, that is, with the fact that they have alternative realizations, for this property allows us to consider the implementation of a computation in a different technology, for example in electronics rather than in neurons. In nature, typically, the realization is given, since natural life is built upon a limited range of substances and processes. On the other hand, there is often selective pressure in favor of exploiting a biological system for as many purposes as possible. Therefore, in a biological context, we expect physical systems to serve multiple functions, and therefore many such systems will not be *purely* computational; they will fulfill other functions besides computation. From this perspective, it is remarkable how free nervous systems are of non-computational functions.

C.3.f   Transduction

The purpose of computation is the abstract transformation of abstract objects, but obviously these formal operations will be pointless unless the computational system interfaces with its environment in some way. Certainly our computers need input and output interfaces in order to be useful. So also computational systems in the brain must interface with sensory receptors, muscles, and many other noncomputational systems to fulfill their functions. In addition to these practical issues, the computational interface to the physical world is relevant to the *symbol grounding problem*, the philosophi-

cal question of how abstract symbols can have real-world content (Harnad, 1990, 1993; MacLennan, 1993). Therefore we need to consider the interface between a computational system and its environment, which comprises *input* and *output transducers.*

The relation of transduction to computation is easiest to see in the case of analog computers. The inputs and outputs of the computational system have some physical dimensions (light intensity, air pressure, mechanical force, etc.), because they must have a specific physical realization for the system to accomplish its purpose. On the other hand, the computation itself is essentially dimensionless, since it manipulates pure numbers. Of course, these internal numbers must be represented by *some* physical quantities, but they can be represented in any appropriate physical medium. In other words, computation is *generically* realized, that is, realized by any physical system with an appropriate formal structure, whereas the inputs and outputs are *specifically* realized, that is, constrained by the environment with which they interface to accomplish the computational system's purpose.

Therefore we can think of *(pure) transduction* as changing matter (or energy) while leaving form unchanged, and of *computation* as transforming form independently of matter (or energy). In fact, most transduction is not pure, for it modifies the form as well as the material substrate, for example, by filtering. Likewise, transductions between digital and analog representations transform the signal between discrete and continuous spaces.

C.3.g CLASSIFICATION OF COMPUTATIONAL DYNAMICS

The preceding definition of computation has been framed quite broadly, to make it *topology-neutral*, so that it encompasses all the forms of computation found in natural and artificial systems. It includes, of course, the familiar computational processes operating in discrete steps and on discrete state spaces, such as in ordinary digital computers. It also includes continuous-time processes operating on continuous state spaces, such as found in conventional analog computers and field computers (Adamatzky, 2001; Adamatzky et al., 2005; MacLennan, 1987, 1999). However, it also includes hybrid processes, incorporating both discrete and continuous computation, so long as they are mathematically consistent (MacLennan, 2010). As we expand our computational technologies outside of the binary electronic realm, we will have to consider these other topologies of computation. This is not so much a problem as an opportunity, for many important applications, especially in

natural computation, are better matched to these alternative topologies.

In connection with the classification of computational processes in terms of their topologies, it is necessary to say a few words about the relation between computations and their realizations. A little thought will show that a computation and its realizations do not have to have the same topology, for example, discrete or continuous. For instance, the discrete computations performed on our digital computers are in fact realized by continuous physical systems obeying Maxwell's equations. The realization is approximate, but exact enough for practical purposes. Conversely a discrete system can approximately realize a continuous system, analogously to numerical integration on a digital computer. In comparing the topologies of the computation and its realization, we must describe the physical process at the relevant level of analysis, for a physical system that is discrete on one level may be continuous on another. (The classification of computations and realizations is discussed in MacLennan, 2004.)

C.4   Expanding the range of computing technologies

C.4.a   A VICIOUS CYCLE

A powerful feedback loop has amplified the success of digital VLSI technology to the exclusion of all other computational technologies. The success of digital VLSI encourages and finances investment in improved tools, technologies, and manufacturing methods, which further promote the success of digital VLSI. Unfortunately this feedback loop threatens to become a vicious cycle. We know that there are limits to digital VLSI technology, and, although estimates differ, we will reach them soon (see Ch. II). We have assumed there will always be more bits and more MIPS, but that assumption is false. Unfortunately, alternative technologies and models of computation remain undeveloped and largely uninvestigated, because the rapid advance of digital VLSI has surpassed them before they could be adequately refined. Investigation of alternative computational technologies is further constrained by the assumption that they must support binary logic, because that is the only way we know how to compute, or because our investment in this model of computation is so large. Nevertheless, we must break out of this vicious cycle or we will be technologically unprepared when digital VLSI finally, and inevitably, reaches its limits.

C.4.b GENERAL GUIDELINES

Therefore, as a means of breaking out of this vicious cycle, let us step back and look at computation and computational technologies in the broadest sense. What sorts of physical processes can we reasonably expect to use for computation? Based on the preceding discussion, we can see that any mathematical process, that is, any abstract transformation of abstract objects, is a potential computation. Therefore, in principle, *any reasonably controllable, mathematically described, physical process can be used for computation.* Of course, there are practical limitations on the physical processes usable for computation, but the range of possible technologies is much broader than might be suggested by a narrow conception of computation. Considering some of the requirements for computational technologies will reveal some of the possibilities as well as the limitations.

One obvious issue is speed. The rate of the physical process may be either too slow or too fast for a particular computational application. That it might be too slow is obvious, for the development of conventional computing technology has been driven by speed. Nevertheless, there are many applications that have limited speed requirements, for example, if they are interacting with an environment with its own limited rates. Conversely, these applications may benefit from other characteristics of a slower technology, such as energy efficiency; insensitivity to uncertainty, error, and damage; and the ability to be reconfigured or to adapt or repair itself. Sometimes we simply want to slow a simulated process down so we can observe it. Another consideration that may supersede speed is whether the computational medium is suited to the application: Is it organic or inorganic? Living or nonliving? Chemical, optical, or electrical?

A second requirement is the ability to implement the transducers required for the application. Although computation is theoretically independent of its physical embodiment, its inputs and outputs are not, and some conversions to and from a computational medium may be easier than others. For example, if the inputs and outputs to a computation are chemical, then chemical or molecular computation may permit simpler transducers than electronic computation. Also, if the system to be controlled is biological, then some form of biological computation may suit it best.

Finally, a physical realization should have the accuracy, stability, controllability, etc. required for the application. Fortunately, natural computation provides many examples of useful computations that are accomplished by

realizations that are not very accurate, for example, neuronal signals have at most about one digit of precision. Also, nature shows us how systems that are subject to many sources of noise and error may be stabilized and thereby accomplish their purposes.

C.4.c LEARNING TO USE NEW TECHNOLOGIES

A key component of the vicious cycle is our extensive knowledge about designing and programming digital computers. We are naturally reluctant to abandon this investment, which pays off so well, but as long as we restrict our attention to existing methods, we will be blind to the opportunities of new technologies. On the other hand, no one is going to invest much time or money in technologies that we don't know how to use. How can we break the cycle?

In many respects natural computation provides the best opportunity, for nature offers many examples of useful computations based on different models from digital logic. When we understand these processes in computational terms, that is, as abstractions independent of their physical realizations in nature, we can begin to see how to apply them to our own computational needs and how to realize them in alternative physical processes. As examples we may take information processing and control in the brain, and emergent self-organization in animal societies, both of which have been applied already to a variety of computational problems (e.g., artificial neural networks, genetic algorithms, ant colony optimization, etc.). But there is much more that we can learn from these and other natural computation systems, and we have not made much progress in developing computers better suited to them. More generally we need to increase our understanding of computation in nature and keep our eyes open for physical processes with useful mathematical structure (Calude et al., 1998; Calude & Paun, 2001). Therefore, one important step toward a more broadly based computer technology will be a knowledge-base of well-matched computational methods and physical realizations.

Computation in nature gives us many examples of the matching of physical processes to the needs of natural computation, and so we may learn valuable lessons from nature. First, we may apply the actual natural processes as realizations of our artificial systems, for example using biological neurons or populations of microorganisms for computation. Second, by understanding the formal structure of these computational systems in nature, we may realize

them in alternative physical systems with the same abstract structure. For example, neural computation or insect colony-like self-organization might be realized in an optical system.

C.4.d General-purpose computation

An important lesson learned from digital computer technology is the value of programmable general-purpose computers, both for prototyping special-purpose computers as well as for use in production systems. Therefore to make better use of an expanded range of computational methodologies and technologies, it will useful to have general-purpose computers in which the computational process is controlled by easily modifiable parameters. That is, we will want generic computers capable of a wide range of specific computations under the control of an easily modifiable representation. As has been the case for digital computers, the availability of such general-purpose computers will accelerate the development and application of new computational models and technologies.

We must be careful, however, lest we fall into the "Turing Trap," which is to assume that the notion of universal computation found in Turing machine theory is the appropriate notion in all frames of relevance. The criteria of universal computation defined by Turing and his contemporaries was appropriate for their purposes, that is, studying effective calculability and derivability in formal mathematics. For them, all that mattered was whether a result was obtainable in a finite number of atomic operations and using a finite number of discrete units of space. Two machines, for example a particular Turing machine and a programmed universal Turing machine, were considered to be of the same power if they computed the same function by these criteria. Notions of equivalence and reducibility in contemporary complexity theory are not much different.

It is obvious that there are many important uses of computers, such as real-time control applications, for which this notion of universality is irrelevant. In some of these applications, one computer can be said to emulate another only if it does so at the same speed. In other cases, a general-purpose computer may be required to emulate a particular computer with at most a fixed extra amount of a computational resource, such as storage space. The point is that in the full range of computer applications, in particular in natural computation, there may be considerably different criteria of equivalence than computing the same mathematical function. Therefore, in any partic-

ular application area, we must consider in what respects the programmed general-purpose computer must behave the same as the computer it is emulating, and in what respects it may behave differently, and by how much. That is, each notion of universality comes with a frame of relevance, and we must uncover and explicate the frame of relevance appropriate to our application area.

There has been limited work on general-purpose computers in the non-Turing context. For example, theoretical analysis of general-purpose analog computation goes back to Claude Shannon (1941), with more recent work by Pour-El, Lipshitz, and Rubel (Pour-El, 1974b; Lipshitz & Rubel, 1987; Rubel, 1993; Shannon, 1941, 1993). In the area of neural networks we have several theorems based on Sprecher's improvement of the Kolmogorov superposition theorem (Sprecher, 1965), which defines one notion of universality for feed-forward neural networks, although perhaps not a very useful one, and there are several "universal approximation theorems" for neural networks and related computational models (Haykin, 2008). Also, there are some CT-relative universality results for molecular computation (Calude & Paun, 2001) and for computation in nonlinear media (Adamatzky, 2001). Finally, we have done some work on general-purpose field computers (MacLennan, 1987, 1990, 1999, 2009a) and on general-purpose computation over second-countable metric spaces (which includes both analog and digital computation) (MacLennan, 2010). In any case, much more work needs to be done, especially towards articulating the relation between notions of universality and their frames of relevance.

It is worth remarking that these new types of general-purpose computers might not be programmed with anything that looks like an ordinary program, that is, a textual description of rules of operation. For example, a *guiding image*, such as a potential surface, might be used to govern a gradient descent process or even a nondeterministic continuous process (MacLennan, 1995, 2004). We are, indeed, quite far from universal Turing machines and the associated notions of programs and computation, but non-Turing models are often more relevant in natural computation and other kinds of unconventional computation.

C.5   Conclusions

The historical roots of Church-Turing computation remind us that the theory exists in a *frame of relevance*, which is not well suited to post-Moore's

Law unconventional computation. Therefore we need to supplement it with new models based on different assumptions and suited to answering different questions. Central issues include real-time response, generality, flexibility, adaptability, and robustness in the presence of noise, uncertainty, error, and damage. Once we understand computation in a broader sense than the Church-Turing model, we begin to see new possibilities for using physical processes to achieve our computational goals. These possibilities will increase in importance as we approach the limits of electronic binary logic as a basis for computation, and they will also help us to understand computational processes in nature.