Expanding the Concept of Computation[1]

Bruce J. MacLennan

Department of Computer Science

1122 Volunteer Blvd.

University of Tennessee

Knoxville, TN 37996-3450, USA

maclennan@cs.utk.edu

Abstract

My goal in this paper is to recontextualize the concept of computation. I review the historical roots of Church-Turing computation to show that the theory exists in a *frame of relevance*, which underlies the assumptions on which it rests and the questions it is suited to answer. Although this frame of relevance is appropriate in many circumstances, there are many important applications of the idea of computation for which it is not relevant, especially in natural computation. As a consequence we need, not so much to abandon the Church-Turing model of computation, as to supplement it with new models based on different assumptions and suited to answering different questions. In this alternative frame of relevance, more suited to natural computation, the central issues include real-time response, continuity, indeterminacy, and parallelism. Once we understand computation in a broader sense than the Church-Turing model, we can see new possibilities for using natural 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. They will also help us to understand computational processes in nature.

Keywords: Church-Turing thesis, natural computation, theory of computation, model of computation

Abbreviations: MIPS = millions of instructions per seconds, VLSI = very large scale integration

---

[1] Preprint of article submitted to *Natural Computation*.

The Frame of Relevance of Church-Turing Computation

It is important to keep in mind that the Turing machine is a *model* of computation. Like all models, its purpose is to facilitate describing or reasoning about some other class of phenomena of which it is a model. A model accomplishes this purpose by being similar to its object in relevant ways, but different in other, irrelevant ways, and it is these differences that make the model more tractable than the original phenomena. But how do we know what is relevant or not? Every model is suited to pose and answer certain classes of questions but not others, which we may call the *frame of relevance* of the model. Although a model's frame of relevance often is unstated and taken for granted, we must expose it and make it explicit in order to understand the range of applicability of a model and to evaluate its effectiveness within its frame of relevance. What, then, is the frame of relevance of the Turing-machine model of computation?

As we know, the Church-Turing theory of computation was developed to address issues of formal calculability and provability in axiomatic mathematics. The assumptions that underlie Church-Turing computation are reasonable in that context, but we must consider them critically before applying the model in other contexts. In its context, for example, for addressing the questions of what is, in principle, effectively calculable or formally provable, it is reasonable to require only that there be a finite number of steps requiring finite resources. As a consequence, according to this model, something is computable if it can be accomplished *eventually, given unlimited but finite resources*.

Another consequence of the historical roots of the Church-Turing theory of computation is its definition of computing power in terms of classes of functions. A function is computable if, given an input, we will get the correct output eventually, given unlimited but finite resources. Of course, the theory can address questions of computation time and space, but the framework of the theory limits its applicability to asymptotic complexity, polynomial-time reducibility, and so forth. The roots of the idea that computation is a process of taking an *input*, calculating for a while, and producing an *output* can be found in the theory of effective calculability as well as in contemporary applications of the first computers, such as ballistics calculations, code breaking, and business accounting.

The Church-Turing model of computation, like all models, makes a number of *idealizing assumptions* appropriate to its frame of relevance. Many of these assumptions are captured by the idea of a calculus, but a phenomenological analysis of this concept is necessary to reveal the background of assumptions (MacLennan 1994a). Although there is not space here to discuss them in detail, it may be worthwhile to mention them briefly (for more see MacLennan 2003, 2004).

The model of information representation derives from mathematical formulas. As idealized in calculi, representations are *formal*, *finite*, and *definite*. We assume that *tokens* can be definitively discriminated from the background and from one another, and we assume that they can be classified by a mechanical procedure into exactly one of a finite number of *types*. The texts, or as we might say, data structures, are required to be finite in size, but also *finite in depth*; that is, as we divide it into parts, we will eventually reach its smallest, atomic constituents, which are tokens. These and other assumptions are taken for granted by the Church-Turing model of computation. Although the originators of the model discussed some of them, many people today do not think of them as idealizing assumptions, which might not be appropriate in some other frames of relevance.

A similar array of idealizing assumptions underlie the Church-Turing model of information processing, in particular, that it is *formal*, *finite*, and *definite*. Applicability of basic operations depends only on mechanically determinable properties of the tokens, and there is no ambiguity in the applicability of these operations (although it is permissible for several operations to be applicable). Each basic operation is atomic, and a computation terminates after a finite number of these atomic operations. Also, it is definitely determinable whether a computation has terminated. Again, these are largely unquestioned assumptions of the Church-Turing model.

Alternative Frames of Relevance

It certainly could, and has, been argued that the notion of computation was vague before the pioneering work of Church, Turing, and the other founders of the theory of computation, and that what they did was analyze, define, and express this previously informal notion in more precise terms. However, I want to argue that there are important concerns connected with computation

that are outside the frame of relevance of the Church-Turing model. Therefore we need to consider these alternative frames and the models suited to them.

In this paper I will focus on the frame of relevance of natural computation (see MacLennan 2005 for the related issues in nanocomputation and quantum computation). Natural computation occurs, of course, in the brains of many species, but also in the control systems of microorganisms and in the self-organized collective behavior of groups of animals, such as insect colonies, flocks of birds, and so on. Natural computation is an important research area because robust, efficient, and effective natural systems can show us how to design better artificial computational systems, and our artificial systems, in turn, can suggest models of computational processes in nature. Such fruitful interchange is already apparent in the theories of neural nets and complex systems. However, natural computation occurs in a different frame of relevance from conventional computing, and so it answers questions about the relative power and equivalence of computing systems from a different perspective, and at the same time it asks entirely new questions.

First, natural computation is often more like real-time control than the evaluation of a function. In nature the purpose of computation is frequently to generate continuous control signals in response to continuous sensor inputs. The system continues to compute so long as the natural system exists, and so eventual termination is generally irrelevant in natural computation. Typically, we are concerned with whether a result, such as a decision to act, can be delivered within a *fixed real-time bound*, or whether continuous control signals can be generated in real time. Therefore, we are not concerned with speed in terms of a number of abstract computation steps, but with real time, and therefore with how the rate of computation relates to the rates of the physical processes by which it is implemented. Also, in the usual theory the criterion is whether an *absolutely* correct output will be produced eventually, whereas in natural computation it may be more important to know how closely a preliminary result approximates to the correct one, since the system may be forced by real-time constraints to use the preliminary result.

Asymptotic complexity is not so important in natural computation as in conventional computation, because in applications of natural computation the size of the inputs are often

fixed; for example, they are determined by the physical structure of the sensors. Other bases for comparing computational processes are more relevant in natural computation. For example, for fixed-sized inputs and processing resources, which computational process has a greater *generality* of response, that is, a wider range of inputs to which it responds well?

A closely related criterion relevant to natural computation is how *flexibly* a system responds to novel inputs, that is, to inputs outside its range. For an artificial system this range is the set of inputs it was designed to process correctly; for a natural system it is the set of inputs that it has evolved to process in its environment of evolutionary adaptedness.

Adaptability is another relevant basis of comparison for natural computation systems, that is, how well do they accommodate themselves to changing circumstances. In this regard we are interested in the range, quality, speed, and stability of adaptation.

Finally, the function of natural computation is to allow imperfect physical agents to operate effectively in the extremely complex, unpredictable natural world. Therefore, in natural computation, we must take noise, uncertainty, errors, faults, and damage as givens, not as peripheral issues added on as an afterthought, if considered at all, as is often done in conventional computing. These considerations affect both the structure of natural computations and the criteria by which they are compared.

It will be apparent that the Church-Turing 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.

Computation in General

Defining 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 20th century computation involved the manipulation of mathematical objects by means of physical operations. The familiar examples are arithmetic operations on numbers, but we are also familiar with geometric

operations on spatial objects and with logical operations on formal propositions. Modern computers manipulate a much wider variety of objects, including character strings, images, sounds, and much else. Therefore, when I say that computation uses physical operations to accomplish *the mathematical manipulation of mathematical objects*, I mean it in the broadest sense, that is, the abstract manipulation of abstract objects. In terms of the traditional separation of *form* and *matter*, we may say that computation uses material processes to accomplish formal manipulation of abstract forms.

The forms manipulated by a computation must be materially realized in some way, but the 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 physically realizable, but, on the other, consider the processes independently of any particular physical realization.

These observations provide a basis for determining whether or not a particular physical system (in the brain, for example) is computational (MacLennan 1994b, 2004). If the system could, in principle at least, be replaced by another having the same formal properties and still accomplish its purpose, then it is reasonable to consider the system computational. On the other hand, if a system can fulfill its purpose only by control of particular substances or particular forms of energy, 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 environment; this is a topic I will consider later.

Based on the foregoing considerations, I have proposed the following definition of computation (MacLennan 1994b, 2004):

> **Definition**: *Computation* is a physical process, the purpose of which is the abstract manipulation of abstract objects.

Alternately, we may say that computation accomplishes the formal manipulation of formal objects by means of their material embodiment. Next I will define the relation between the physical and abstract processes:

> **Definition:** A physical system *realizes* a computation if, at the level of abstraction appropriate to its purpose, the abstract manipulation 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. Next I will suggest a definition by which we can classify various systems, both natural and artificial, as computational:

> **Definition:** A physical system is *computational* if its purpose is to realize a computation.

Finally, for completeness:

> **Definition:** A *computer* is an artificial computational system.

Thus I restrict the term "computer" to intentionally constructed 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 they can be improved.

First, these definitions make reference to the *purpose* of a system, but philosophers and scientists are justifiably wary of appeals to purpose, especially in a biological context. However, I claim that the use of purpose in the definition of computation is unproblematic, for in most cases of practical interest, purpose is easy to establish. On the one hand, 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 purpose of biological systems, such as the digestive system and immune system, and make empirically testable hypotheses about their purposes. Ultimately such claims of purpose are reduced to the selective advantage to a particular species in that species' environment of evolutionary adaptedness, but in most cases we can appeal to more

familiar ideas of purpose.

However, I should mention one problem that does arise in biology and can be expected to arise in our biologically-inspired robots. That is, while the distinction between computational and non-computational systems is significant for *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 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 typically 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 purposes, 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 the nervous systems of all animals are from non-computational functions.

Transduction

I have emphasized that the purpose of computation is the abstract manipulation of abstract objects, but obviously this manipulation 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 in order to be useful. In addition to these practical issues, the computational interface to the physical world is also relevant to the *symbol grounding problem*, the philosophical 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.

So we can think of *pure* transduction as changing matter while leaving form unchanged, and computation as transforming form independently of matter. 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.

Classification of Computational Processes

I have tried to frame this definition of computation 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. However, it also includes hybrid processes, incorporating both discrete and continuous computation, so long as they are mathematically consistent. 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 be of the same type, 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, much like 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. (See MacLennan 2004 for more on the classification of computations and realizations.)

Expanding the Range of Physical Computation

General Guidelines

Next I will discuss the prospects for expanding the range of physical processes that can be applied to computation. I will consider some general guidelines, new concepts of general-purpose computers, and natural computation. But why should we want to expand the concept of computation?

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 and technologies, which further promote the success of digital VLSI. Unfortunately this powerful feedback loop is rapidly becoming a vicious cycle. We know that there are limits to digital VLSI technology, and, although estimates differ, we will reach them soon. We have assumed there will always be more bits and more MIPS, but that 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 developed. 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.

Therefore, as a means of breaking out of this vicious cycle, I will 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 my preceding remarks, we can see that *any* mathematical process, that is, any abstract manipulation of abstract objects, is a potential computation. Of course, not all of these are useful, but mathematical models in science and engineering offer many possibilities. Aside from *de novo* applications of mathematical techniques to practical problems, mathematical models of computation in natural systems may be applied to our computational needs. Certainly, a computation must be physically realizable, which means that we need to find at least one physical process for which the desired computation is a good model. Nevertheless, 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 computational technologies is much broader than might be suggested by a narrow definition 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 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 adapt or repair itself. Another consideration that may supersede speed is whether the material substrate 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 the 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.

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

I believe that natural computation provides the best opportunity. Nature provides 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 implement 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. (I am thinking of 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. Therefore, one important step toward a more broadly based computer technology will be a library 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 in 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.

General-purpose Computation

An important lesson learned from digital computer technology is the value of programmable general-purpose computers for prototyping of special-purpose computers as well as for use in production system. 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 such 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 particular application area, we must consider in what ways the programmed general-purpose computer must behave the same as the computer it is emulating, and in what ways 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.

Fortunately there has been some work in this area. For example, theoretical analysis of general-purpose analog computation goes back to Claude Shannon (1941), with more recent work by Pour-El (1974) and Rubel (1981, 1993). In the area of neural networks we have several theorems based on Sprecher's (1965) improvement of the Kolmogorov superposition theorem, which defines one notion of universality for feed-forward neural networks, although perhaps not a very useful one. Also, I have done some work on general-purpose field computers (MacLennan 1987, 1990, 1999). 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* might be used to govern a gradient descent process (MacLennan 1995, 2004). We are, indeed, quite far from universal Turing machines and the associated notions of programs and computing, but non-Turing models are often more relevant in natural computation and other new domains of computation.

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 natural computation. Therefore we need to supplement it with new models based on different assumptions and suited to answering different questions. Central issues include continuity, indeterminacy, and parallelism. Once we understand

computation in a broader sense than the Church-Turing model, we begin to see new possibilities for using natural 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. They will also help us to understand computational processes in nature.

References

Harnad S (1990) The symbol grounding problem. Physica D 42: 335–346

Harnad S (1993) Grounding symbols in the analog world. Think 2: 12–78

MacLennan BJ (1987) Technology-independent design of neurocomputers: The universal field computer. In: Caudill M & Butler C (eds) Proceedings IEEE First International Conference on Neural Networks, Vol. 3, IEEE Press, pp 39–49

MacLennan BJ (1990) Field computation: A theoretical framework for massively parallel analog computation, parts I–IV. Technical report UT-CS-90-100, Department of Computer Science, University of Tennessee, Knoxville

MacLennan BJ (1993) Grounding analog computers. Think 2: 48–51

MacLennan BJ (1994a) Continuous symbol systems: The logic of connectionism. In: Levine DS & Aparicio IV M (eds) Neural Networks for Knowledge Representation and Inference, Lawrence Erlbaum, Hillsdale, pp 83–120

MacLennan BJ (1994b) "Words lie in our way." Minds and Machines 4: 421–437.

MacLennan BJ (1995) Continuous formal systems: A unifying model in language and cognition. In: Proceedings of the IEEE Workshop on Architectures for Semiotic Modeling and Situation Analysis in Large Complex Systems, August 27–29, 1995, Monterey, CA, pp. 161–172

MacLennan BJ (1999) Field computation in natural and artificial intelligence. Information Sciences 119: 73–89

MacLennan BJ (2003) Transcending Turing computability. Minds and Machines 13: 3–22

MacLennan BJ (2004) Natural computation and non-Turing models of computation. Theoretical Computer Science 317: 115–145

MacLennan BJ (2005) The nature of computing — Computing in nature. Technical report UT-

CS-05-565, Department of Computer Science, University of Tennessee, Knoxville

Pour-El MB (1974) Abstract computability and its relation to the general purpose analog computer (some connections between logic, differential equations and analog computers). Transactions of the American Mathematical Society 199: 1–29

Rubel LA (1981) A universal differential equation. Bulletin (New Series) of the American Mathematical Society 4: 345–349

Rubel LA (1993) The extended analog computer. Advances in Applied Mathematics 14: 39–50

Shannon CE (1941) Mathematical theory of the differential analyzer. Journal of Mathematics and Physics of the Massachusetts Institute Technology 20: 337–354

Sprecher DA (1965) On the structure of continuous functions of several variables. Transactions of the American Mathematical Society 115: 340–355.