

Super-Turing or Non-Turing?

Bruce MacLennan

Dept. of Computer Science, University of Tennessee, Knoxville

1 Introduction

“Hypercomputation” is often defined as the ability to transcend Turing computation (TC) in the sense of computing a larger class of functions than can Turing machines. While the possibility of various kinds of hypercomputation, in this sense, is certainly important and interesting, I will argue that this definition limits our perspective, and that there are many other important senses in which we may “transcend Turing computation.”

2 The Limits of Turing Computation

Frames of Relevance

It is important to remember that TC is a *model* of computation, which 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 certain *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 fuzzily) 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 make sense (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 may give good answers (showing that its *actual* frame may be larger than its *intended* frame), but we cannot assume that. 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.

The Frame of Relevance of Turing Computability

It is important to consider the frame of relevance of Turing computation (TC), 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 TC 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 TC model arises from an idealized description of what a mathematician could do with pencil and paper. Although a full analysis of the TC frame of relevance is beyond the scope of this abstract, I will mention a few of the idealizing assumptions.

Within the TC 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 simplified model of arithmetic with paper and pencil. Again, in the context of the formalist programme 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 TC 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 from 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 between them. Thus 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 they 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!

It is important to mention the concept of time presupposed in the TC 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*. This is because the TC 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 questions 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 TC model that is an important concern of this workshop, the assumption that a 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 programme, 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 a while, 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).

Therefore, the natural way to compare the "power" of models of computation was in terms of the classes of functions they could compute, a linear 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). (I should note 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.)

3 New Computational Models

A reasonable position, which many people take explicitly or implicitly, is that the TC 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 TC model exists in a frame of relevance, which delimits the kinds

of questions that it can answer accurately, and, as I hope to show, there are important computational questions that fall outside this frame of relevance.

Natural Computation

Natural computation may be defined as computation occurring in nature or inspired by computation in nature. The information processing and control 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, 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 TC 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 fast 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 for the inputs 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 with the square of the number of inputs to the net; what matters is the absolute number of neurons required for the numbers of inputs there actually are, 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 TC 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* 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 the answers are more indicative of the model 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.

This 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, 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 TC 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 TC frame. For example, 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), 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.

Another 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 TC 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 TC 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).

Nanocomputation

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

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 their interrelationships (e.g., interconnections), and in the structures constructed by nanocomputational processes. 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 temporary equilibria in constant flux. Similarly, unlike in the TC model, nanocomputational operations cannot be assumed to proceed correctly, but 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 always 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 nanocomputation, which should be an essential part of any model of it.

4 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 alternative 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 TC 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 alternative 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 (molecular, biological, optical, quantum) for computation?