On Communication and Computation

PAUL BOHAN BRODERICK
Department of Philosophy, 320 Bowman Hall, Kent State University, Kent, OH 44240, USA;
E-mail: pbohanbr@kent.edu

Abstract. Comparing technical notions of communication and computation leads to a surprising result, these notions are often not conceptually distinguishable. This paper will show how the two notions may fail to be clearly distinguished from each other. The most famous models of computation and communication, Turing Machines and (Shannon-style) information sources, are considered. The most significant difference lies in the types of state-transitions allowed in each sort of model. This difference does not correspond to the difference that would be expected after considering the ordinary usage of these terms. However, the natural usage of these terms are surprisingly difficult to distinguish from each other. The two notions may be kept distinct if computation is limited to actions within a system and communications is an interaction between a system and its environment. Unfortunately, this decision requires giving up much of the nuance associated with natural language versions of these important terms.

1. Introduction

The following paper is an initial report on my investigations into the impact of probabilistic models on philosophical problems. If there is any specific message it is that philosophers should not take Turing Machines quite as seriously as we do. The argument though does not attack any particular failing of Turing Machines. Instead it places this venerable model in a wider context with a richer set of modeling tools available to do philosophical work. Overemphasis on Turing Machines has lead philosophers to miss out on a number of other important phenomena that do the same work in different ways. When considering the appropriate models of algorithmic complexity, Donald Knuth argues against consist use of Turing Machine models, not because of any intrinsic failing, but because they just don't stand up to claims one might have about the veracity of Turing Machines as a model. According to Knuth, "Relevance is more important than realism." (Knuth, 1997, p. 464)

This paper examines the sorts of models that are used to discuss communication and computation. These models, if they are successful, presumably capture something important about the two phenomena in question. The types of models that will receive the most attention below are Turing Machines and Markov Processes. Neither of these make anything more than analogical references to physical structures and the properties embodied by the models are different in important respects from the systems they model. If Turing Machines only really told us about radically unskilled clerks working on ling pieces of paper, it would be less than a joke. Rather both models capture what could be called the informational structures

Minds and Machines **14:** 1–19, 2004.

that define success of communication or computation. These two processes are similar because they are fundamentally informational in ways that combustion or digestion are not. I will be arguing that communication and computation are not as different as one might expect. In this sense, its a contribution to the philosophy of information, both because the discussion is carried on an information level, but also because it explores the models, that is the vocabulary, most commonly used in discussions of information. In other words, it is and elaboration of information and computational methodologies (Floridi, 2001, p.14), where model building is taken as a genus of methodology.

Nancy Cartwright has demonstrated that scientist and engineers with practical concerns on their mind often utilized various, and occasionally incompatible mathematical models, without giving up a commitment to reality of the phenomena they are studying.[1] The models themselves need to be evaluated primarily in terms of their utility.

The utility and beauty of the models we will be considering have inspired their wide use in the sciences and engineering disciplines. The first two specific models are generally also understood to give us some insight into ordinary experience in so far as computation and communication are ordinary experiences. The objects that each of these models are supposed to capture are not exactly equivalent to the aspects of day-to-day experience that we discuss using similar words. The Shannon model of communication has often been looked down upon because it does not capture all of what is ordinarily meant by communication. However, the lack of nuance in the model does not diminish the model's usefulness. The paper will close with a discussion of how the models under consideration, and their similarities are successful at capturing one little noticed ambiguity in how the phrases communication and computation are often used in nontechnical discussion.

2. Two Models and Unifying Framework

In this section I will review the most famous models of computation and communication. Turing Machines and Communication Channels respectively. I will then show, that at least in practical contexts, these two model's aren't so different after all. In the next section, I'll try to show that a similar situation holds in natural language.

2.1. TURING MACHINES

According to standard presentation,[2] a Turing Machine is a 8-tuple M = ($\mathbf{Q}$, $\mathbf{\Sigma}$, $\mathbf{\Gamma}$, $\boldsymbol{\delta}$, $\boldsymbol{q}_0$, $\mathbf{B}$, $\mathbf{F}$, $\mathbf{T}$)

- $\mathbf{Q}$ · the states of the machine
- $\mathbf{\Sigma}$ · the input symbols recognised by the machine
- $\mathbf{\Gamma}$ · the complete set of tape symbols

$\boldsymbol{\delta}$ · the transition function

$\boldsymbol{q}_0$ · the initial state

B · the blank symbol

F · the set of final states

T · a time index

The Turning Machine (TM) has been a staple among philosophers for a half century now. Nonetheless, I'll review the basic notion. The procedures necessary to carry-out a computational process can be modeled in terms of what could be accomplished by a manipulating a pencil and an arbitrarily long piece of paper according to a small set of rules. The list above is meant to specify the limitations on those sets of rules. The states of the machine, Q, identify what rules are operative at time T, Those rules include which symbols can be written to the paper and what to do when a particular state is read off the tape. The response to any symbol is limited to moving the tape, writing/erasing a symbol on the tape and/or moving into a new state.

There have been simpler descriptions than this one. Many presentations make use of fewer components. The time index, for instance, is generally not included, the input symbols must be subset of the complete set of subset symbols, etc. T, the time index coordinates the actions of the Turing Machine. The other members of the tuple define what occurs at each tick of the clock. Because q_0 and B both belong to σ and F is a sub-set of Q, these three items are sometimes left off of the specification of a TM and only explained under their relevant items. Likewise the distinction between Σ and Γ is often made explicitly.

One feature that has often been specified is the length of the tape in which the various marks are recorded. The length of this tape is arbitrary and can grow depending on the needs of the particular Turing Machine under consideration.

Even though we haven't yet discussed the features of Markov processes, it might be useful to take a moment to see how these two models might not fit together. First, since a Markov process has a specific relation to prior states, which we will discuss below, it is not exactly possible to model a fully universal Turing Machine as a Markov Process. The second significant difference is that the δ of a Turing Machine is a deterministic function while a Markov process may have probabilistic transition function (including those with a probability of unity). This is equivalent to a Nondeterministic Turing Machine (NTM). NTMs have some interesting properties.[3] In particular they can compute some functions faster than the corresponding deterministic TM. However, this is not directly relevant to our current goal which is to find a vocabulary in which to discuss the similarities and differences of two important phenomena. Finally, with Turing Machines the time index is generally left assumed as the amount of action per unit of time is taken to be fixed. However, for Markov processes, the amount of action per unit of time may vary. Thus, a TM is, in Markovian language, a *homogeneous chain*.

2.2. CHANNELS OF COMMUNICATION

The Shannon model of communication (Shannon and Weaver, 1949) and coding is usually associated with a simple schema of Source-Channel-Distinction, where the channel is taken as a source of noise. Obviously this model does not take into account many of the facets of communication that are typically taken as critical for even the most basic exchanges. What makes the model interesting is the fact that such a powerful and useful theorem as Shannon's Fundemental Theory arises from these considerations.

Although Shannon does not explicitly use the formalism, ergodic information sources are also tuples. $S = (Q, \Gamma, \delta, P, C)$ where

Q · the states, $S_1, \ldots, S_2$, of the machine (Shannon and Weaver, 1949, p. 38).

Γ · a set of symbols that can be transmitted by the channel (Shannon and Weaver, 1949, p. 45). This is analogous to Γ in TM structure given above[4] Shannon does not make a distinction between what I am calling Q and what I am calling S, he does however note the difference in passage I cite. The difference resides in how each can be used in a graphical representation of a communication procedure. The same letter can be associated with different nodes on a graph, but the node is important because it limits what future states a system can evolve into.

δ · the transition function. This is an important difference from the TM formalism.

P · a set of sets of probabilities $p_i(j)$, such for each state i (that is each $q \in Q$), the probability that the machine will enter state j (Shannon and Weaver, 1949, p. 41).[5]

C · the capacity of the channel (Shannon and Weaver, 1949, pp. 36–37).

The Shannon-model of an ergodic source is closely related to the Nondeterministic Turing Machine (Hopcroft et al., 2001, pp. 340-341).

Shannon's is explicitly developed as a Markov process. For this reason I will not devote any time in the next section arguing that a Communication Channel, as modeled by Shannon, can be meaningfully interpreted as a Markov process. The problem for Shannon's model is to determine whether or not these particular Markov processes provide an appropriate model of communication. If the appropriateness of a model is measured in terms of unique fit, or of structural similarity to the modeled phenomena, the this picture of communication is, quite rightly, doomed. On the other hand, if the utility of the model is fundamental, than the various instances of communication in which this model has been deployed to good effect, notably in telephone communication and theoretical cryptography.[6]

2.3. MARKOV PROCESS

Markov models unite the two different sorts of models just described. This will be shown (or at least suggested) later in this section. The fact that both Turing

machines and Communication Channels are, for practical purposes, both instances of another type of model is quite suggestive.

2.3.1. *Example*

Markov processes are usually illustrated with examples such as the following. Assume for a moment that random motion is the overriding influence on the positions of the oxygen molecules in whatever room you now occupy. Furthermore, assume that the state of the room, the state of the oxygen molecules in the room, can be measured and recorded. I will use this example to illustrate how the Markov processes are defined and will use it again below. Finally, in each moment, each molecules can move at most some finite distance. Thus, according to this model, the correct positions of the molecules in this room depend only the positions at the previous moment, adjusted by their speeds and potentially by many collisions.

2.3.2. *Specifications of a Markov Process*

A Markov process can be represented by specifying each of the states through which the process could evolve, symbolized by $X_i = (Q, p)$ where

n · a finite list of variables that will take particular values (enter states) as the process evolves. One way to model the example given would be to provide each molecule with a separate variable to track its position. This value, often known as the time index, can be discrete or continuous, finite ro infinite, though the treatment of each of these cases differs.[7] The subscript i in the equation above varies over the set ot n's. The important point to notice is that this number is fixed by the specification of the process.

Q · the states that the variables may enter, in the example, a large collection of positions distributed over each of the variables. The possibilities would correspond to the possible positions in the room. Depending on how finely grained our measurements are, some of the values may be reused (that is, two or more molecules may be in the same position, assuming the values are widely space).

P · the collection of transition probabilities $P_{i,j}$ from one state, i to another state j. We will represent the complete probabilities that each variable in a particular collection will evolve in a certain way as a matrix, $S_{i,j}$ where each row i represents a particular variable and each column j represent the states into which those variables may evolve. The value associated each position in this matrix indicates the chance that the variable in state Q_1 will evolve into state Q_2.

In general, I have been and will be referring to Markov processes. In those cases where the index takes discrete values, as with Turing Machines, the term Markov chain is also appropriate.

The evolution of the processes over time is modeled by the successive multiplication of the transition matrix. In some cases, this converges on a final matrix.[8] The analogy between these steps and the various internal states of a Turing Machine

should be fairly clear. The various Q introduced in the section on Communication Channels above were specifically introduced as instances of a Markov process.

In addition to the above components, the Markov process is also limited by the Markov (Completeness) Condition. For our purposes, this means that the transition matrix expressed a complete set of probabilities (each row sums to 1) and that only the transition matrix is used to compute new states (thus only the previous state is used to compute new states, there is no "memory" beyond the most recent state. As we shall see, Turing Machines violate the second part of the Markov Condition, but acceptable approximation is possible.

In the notation I have been discussing, the Markov condition would be formulated as follows:

Markov Condition: Given a chain of events in a process, $X_1 \rightarrow X_2 \rightarrow X_3$, Then the following two conditions hold:

1. $P(X_3|X_2) = P(X_3|X_2 \cdot X_1)$
2. $P(X_3|\neg X_2) = P(X_3|\neg X_2 \cdot X_1)$

The idea behind the condition is that the probability of the current state is determined only by the immediately prior state and whatever path the machine might have taken to get to that state does not effect this probability. If we're choosing to look at Turing Machines we have two relevant states that we might want to model, either the internal state of the machine (Q above) or the total state of the machine) that is Q plus the current contents of the external memory, whatever has been written on the tape. The problem is that the tape has a potentially infinite number of states while Markov chains have only a finite number of states. If we don't include the chain somehow in our model of TMs, then the Markov conditions will be violeted since the prior states of the machine will have and effective way of altering the probability of future states.

Generally, philosopher's have made use of Markov chains to analyse casual relations.[9] Here the point is to develop a general framework for discussing computation and communication. The use of the Markovian framework suggests some ways of loosening the Turing Machine structures to build non-deterministic models of these phenomena. We will now explore the necessary loosening to make Turing Machines into Markov processes. (Again, this is not to ignore the large literature on non-deterministic TMs, but an attempt to see how the pieces fit together.)

2.4. MEMORY

Thus, the Markov process presents a very general and flexible way of talking about how a system can evolve over time. Turing Machines are also an evolution of a sort of states over time. The problem is that the Markov representation does not include any formal way of discussing external memory. A TM effectively has two types of memory, the external or explicit memory stored on the tape and the implicit memory resulting from the finite number of methods of state-evolutions

that the machine could have passed through to achieve its current state. Finite State Machines (FSMs), the less capable cousin of the Turing Machines, utilise only this second sort of memory when computing tasks that require memory. So, for instance a FSM that recognises strings consisting of two characters grouped in sub-strings of equal length. No tape is required to keep track of how many of one symbol has been considered in a particular sub-string. A Markov process has an even more limited form of implicit memory than a Finite State Machine. As presented, in some processes, not every set of previous evolutions can be preceded by a particular current terminal state to the chain.

Consider again the placement of oxygen molecules in this room. They are spread out in a roughly even pattern. Random fluctuations keep this relatively even distribution, since each state have a close resemblance to the immediately prior state. If the movement is really random, what is to prevent the oxygen from being compressed into one corner of the room? There are very few possible states that could, in a single step evolve into this state. Likewise, this state could only evolve into a limited number of successor states. This special state then has a somewhat vague memory since we can draw some conclusion about the prior steps in the chain. For instance, the steps prior to the oxygen gathering in the corner of the room are all steps in which the oxygen is relatively closer to that state than it is to its almost homogenous distribution across the room. Most terminal states would, of course, allow for a much greater range of immediately prior states. In any particular TM, of the other hand, if the initial state of the machine is given, it should be possible to recreate the path that the machine took to get to its current state. If the input is not given, then the path may not be unique, so a collection of paths may the best that could be hoped for.[10]

The true Universal Turing Machine requires a sort of memory that violates the Markov condition. The unlimited nature of the "tape" or explicit memory makes the nth step in the machine's evolution dependent upon the steps before the immediately preceding $(n-1)$ step. However, we aren't interested just in the ideal Turing Machine but in the systems that can be appropriately modeled by them. That is we need a Markov process that is good enough to model any real world computation that a TM is capable of modeling. To do this, allow as many empty variables as the largest practical) real life machine might need and allow for each to take a value of 0 or 1. In practical terms, each could be understood as a box on the machine's tape or as a memory location. So long as we don't need to model the action of a computer with an infinitely large hard drive, this approximation should be good enough.

The preceding sections have assumed that computation and communication can both be adequately modeled by Markov processes or Markov processes plus explicit memory. This assumption is shared by most attempts to build models of "information". Even attempts at semantic-information models (as opposed to Shannon's "syntactic" or engineering model) may be recast on the insights shared by the above (Markovian) formulations. Carnap and Bar-Hillel (1964) consider

the information of a particular sentence rather than a stream of information or the informative value of some time dependent process. The communication measure suggested by Carnap and bar-Hillel maintains the Markovian character of the Shannon model.

2.5. MARKOV PROCESSES AND THE MODELS OF COMPUTATION AND COMMUNICATION

In order to draw out the similarities between TMs and relevant models of communication, I am going to treat both of them *as if* they were special sorts of Markov process.[11]

The most famous Markovian model of communication was developed by Claude Shannon (Shannon and Weaver, 1949). It is more unusual to consider Turing Machines as Markov process. There are good reasons to avoid this treatment. First, and most importantly, Markov processes have no external memory. No tape to which one can write intermediate steps in a computation. Thus, is would seem, Markov processes should be useful for modeling Finite State Machines (FSMs), and they can be so used. Unfortunately, there's a world of difference between FSM and an TM. Further, a Markov representation adds unnecessary complexity to an elegant construction. Also, TM define deterministic procedures while the theory of Markov processes are designed to account for probabilistic processes.

A TM, on my reading, is a Markov process with deterministic state-transitions and no variation in either step duration or probability distribution[12] that is supplemented with appropriate types of memory. The transition probabilities of a Turing Machine are all set to 1. One way of putting this is that the transition matrix δ_{ij} is simply a transition vector δ_i. On each possible input, there is only one possible state into which the machine transitions. There is no need for the other columns of the transition matrix of a more complex Markov Process. Since 1 is a valid probability, this move does not do too much damage to the to the Markov formalism. Also, since each row of the vector sums to 1 (*is* 1), the Markov completeness is not violated.

3. Our Computation

In the previous section, I commented on how Turing Machine's can be placed next to Shannon's Communication streams as applications of a broadly Markovian framework. These models for computation and communication turned out to be more similar than they might have at first seemed. If the models are similar, does this imply that perhaps the phenomena to be modeled might have similar analogies. In this section, I will attempt to show that what we typically mean by computation and communication do share some affinities. To evade difficulties involved with

the appropriate definition of computation over symbols.[13] I will start by trying to define computation broadly as "whatever it is that a computer does".

3.1. DEFINITION OF COMPUTER

The suitability of Turing Machines as models of cognitive processes has long been in doubt. Turing Machines are traditionally the most popular models for electronic computers. Writers, such as Donald Knuth who address modeling of saved program computers have often preferred to avoid TMs in favour of less "comparatively realistic" (Knuth, 1997, p. 464) models. The differences between these are generally taken to be negligible. It is exactly these differences, that I have been attempting to exploit in this paper by considering these three different types of models. The success of the various models can only be evaluated if one has some ideas about what the models are supposed to model. We will now address this problem by attempting to figure out what we mean by computer without reference to particular electronic consumer goods.

DEFINITION 1. A *computer* is any member of that class of machines that are capable of effectively computing a recursive function.

Each component of this definition shall be discussed in turn.[14] An *effective procedure*, in this case, an effective computation, is one that may be performed with perfect success. Although this notion may have interesting theoretical consequences,[15] it is possible to read too much into it. Effective procedures are extremely common, since anything that does not allow for degrees of success (such as walking through a door, passing a test, or living to 30) is an effective procedure. Limiting our attention to effective procedure suppresses the possibility of borderline cases. Thus, this sort of description breaks up the range of possible completions of the described procedure into a set of distinct, mutually exclusive possibilities. Hence, this definition has a certain prejudice for digital machines over analog machines as the proper exemplars of "computer". Some notion of effectiveness is necessary to rule out cases that arbitrarily compute functions,[16] that is, that don't compute anything in a meaningful way, but may be described as computing any functions at all because of the lack of conditions on what should count as a computations. Effectiveness does rule this out, or least sets up the possibility of measuring a successful computation once the specifics of the system have been determined.

The definition is not equivalent to the closely related notion of an *instantiation of a Universal Turing Machine* (which has often been used as a definition of a computer) since there aren't any such things. Turing Machine perhaps, but not universal ones. True universality depends on the allowance of an arbitrary amount of time and memory for the execution of programs. Not only are these circumstances not available in the real world, but all systems that might be modeled in interesting ways by finite state machines have been optimized, by either their

designers or natural selection, to execute particular functions efficiently with a corresponding loss of efficiency in other realms. This loss of efficiency is strong enough to prevent the execution of some tasks. Hence, a system optimized for survival and reproduction is not doubt so inefficient in running spreadsheets that they are arguable beyond that systems capacity. The same can be applied to the spread-sheet optimized system's capacity to find prey. I've already made use of this unrealistic aspect of TM's models in the argument above.

3.1.1. *Recursion*

The notion of *recursive function* possesses a deceptive simplicity. In computer science,[17] the notion of recursion is often applied in a broad fashion as the complement of iteration. In this sense, recursion is a procedure which must be called as part of itself, while an iterative procedure is performed multiple times as part of a larger procedure, but never calls itself as part of itself. For instance, a recursive procedure for finding the factorial of a number may be "multiply that number by the factorial of the number which is one less than it". The factorial of that number is computed in the same fashion. The computation at any particular application of the procedure cannot be completed until the number one is reached and then the recursively defined procedure can begin working its way "back out". An iterative process for finding factorials is also possible. The iterative process is more complex in that it requires two variables (one to keep track of the stage of the computation, the other holds a value that approaches the final value of the factorial) while the recursive definition only requires one. On the other hand, each step in the iterative process requires the same amount of memory, so long as the number is within the bounds that the machine can deal with, while the recursive definition requires an amount of memory proportional to the size of the number to be computed. In this broad sense, a recursive procedure is one which can be defined form a kernel of basic scheme and a limited set of procedures to be performed on them. The results of any application of the procedures to either the kernel or to previously defined procedures counts as a procedure. The number of times that these procedures can be carried out to define new procedures is limited only by practical concerns.

Turing Machines are commonly used to study recursion, especially in the context of computation. Other methods are, of course, available. The varieties of recursion were given their canonical formulation by Stephen Kleene (1964),[18] following the work of Gödel and others. At the base of the hierarchy are six schema (five presented at the beginning of the paper, one derived). Any function is primitively recursive if it is strictly equivalent to a binary function whose definition is expressible in one of these schemes. General recursive functions involve a set of recursively defined equations returning a single numeric value (not necessarily 0 or 1 as in primitive recursion). Partial recursion refers to functions that may not be defined over the entire range of possible arguments or are equivalent to sets of equations that may not return any numeric value under certain circumstances. Sets

and predicates can also be recursive. Each of these sorts of recursion are actually special cases of the general sort of recursion informally outlined just above. They are recursions working from a kernel constituted by Kleene's schema.[19]. Note that this relationship implies that the definition of computer currently under discussion would not be endangered by a potential disproof of the Church–Turing Thesis. That thesis states, in essence, that any possible set of basic procedures can, at most, define the complete set of procedures defined using Kleene's (or any equivalent) method. Should Church–Turing fail, the definition would be held to work in the more general, less formal sense glossed above. In practice, the more formal sense is assumed as a propaedeutic to design (of software, machines and algorithms). Nonetheless, the potential disproof of Church–Turing would raise some questions about what it would mean for an artifact to implement such things. The guides to the practice of Computer Science, such as Abelson et al. (1996), do not seem to recognize any problem of that sort.

This sense of recursion was originally developed to address specific questions that had developed in mathematical logic. It entered into this discussion because it forms the central element of a common, perhaps the most common, definition of computer. Somehow, this notion must be able to bridge the gap from the abstract realm of decision problems to the more rough and tumble one where physical machinery is designed and constructed from material components. Design principles make this connection possible and those principles can be formulated most effectively using some form of the analog-digital distinction.

Von Neumann Architecture. Within the category of computational artifacts, the case of von Neumann architecture conformant machines deserve special attention. In ordinary speech, we can avoid linguistic contortions by using von Neumann machines as a rough plan for what we mean by 'computer'. The problem is that even though most commercially available computers are von Neumann compliant, it would not be too difficult to replace one with a similar beige box that carries out the same functions but fails to meet the specifics of the von Neumann specifications.

Most, though not all, modern computers are examples of the von Neumann architecture.[20] The von Neumann Architecture could be considered a definition of this class of computers. The von Neumann Architecture definition contrasts with definition 1 in that it defines a variety of computer rather than trying to provide a unifying notion of the entire class of computers. Von Neumann Architecture *only* describes design features of computers, such as the use of registers rather than accumulators in local memory,[21] the explicit presence of at least two sorts of memory[22] and, most importantly, a central processor that performs all manipulations. In short, the von Neumann Architecture is a blueprint for instantiating a computer according to the definition given above under constraints of available technology since about 1950.[23] Production technology has changed a great deal since that time, making other possible architectures feasible by economies of scale and institutional momentum have prevented the alternatives from being as widely

accepted as the von Neumann Architecture. There is no powerful indication that the von Neumann architecture is broken, so there is little chance that any suggested fixes to this architecture will be adopted.

3.1.2. *The Strengths of the Proposed Definition of Computer*

One strength of this definition is that it gives no clue about the actual construction of such a machine.[24] The sort of agnosticism is appropriate because it includes both analog and digital computers in all of their variety, including machines that conform to the von Neumann Architecture, as well as those that don't; connectionist machines, massively parallel machines, Alan Turing's Automatic Computing Engines and the various information appliances that have come to inhabit our world. The definition does admit of some borderline conditions. The humble steel coil thermostat has often been called upon as an example of a machine that has some interesting properties. As it 'computes' a function from the position if its input device (either and analog dial or digital keypad) to the room temperature, usually by implicity varying quantities hidden from view in the furnace. The problem with the definition of a computer is also that it does not give any hint about how such a machine could be constructed, the machine has been characterize at too high a level of abstraction for that purpose. When components have been established as having either analog or digital character, constraints have been placed upon the design problem, the sorts of constraints necessary for planning a physical construction.

3.1.3. *Problems with this Definition of Computer*

Since the definition makes reference to the abstract features by which computers are most frequently defined, it is in danger of lapsing into the unfortunate position that either everything identifiable is a computer or that there are no computers. I will refer to this as the "vacuity objection". The vacuity objection has been raised against most attempts to develop a working definition of what a computer is.

Many attacks[25] on the relevance of computational/informational pictures of real-world phenomena accuse them of vacuity. This seems a non-vicious vacuity, so long as we are capable of learning something from the picture. There is a real danger *if* the phenomena may only be identified through these descriptions. That shouldn't be a problem in the definition of computers, since there are other ways of identifying computers, specifically by identifying them as computers of a certain type. Since the various types of computers are generally tied to some sort of physical description there should,'t be a problem with the use of more abstract pictures of how these systems operate. One way around the vacuity objection would be to limit the reference of 'computer' to those that clearly conform to an architecture such as the von Neumann Architecture.

In *Representation and Reality* (Putman, 1998), Hilary Putman develops a set of arguments that are meant to refute the position commonly known as functionalism. The technical strength of this argument speaks directly to the distinction

between the practical construction and theoretical description of procedures. On one reading of this argument, Putnam would be showing that not only is any thing a computer, but everything is every computer. Philosophical discussions of computation become almost inevitably entangled with issues that more properly concern cognition. Putnam observes, succinctly though perhaps with the wistfulness of one who has gone down a path that in retrospect they might rather no have taken,

> With the use of computer science, and entirely new paradigm of what a scientific realist account of intentionality might look like presented. (Putman, 1998, p.108)

The point of *Representation and Reality* is that the development of symbol-based computation does not, in fact, herald the development of a final-theory of human psychology. I agree with this assessment, the rise of computer science does not constitute a revolution in psychology. This implies that either computer science does not raise any issues of philosophic importance,[26] or that problems raised are more novel than a rehash of traditional philosophical problems concerning human cognition, such as the mind-body problem. Computers have interesting features, grounded in their symbol manipulating capabilities, that are not shared by, say, a rock.[27] Putnam presents a major challenge to this assertion when he suggests that every finite automata is implemented by every open system, including rocks.

Putnam's argument opposes not only the possibility of an ideal psychology, or even a theory of intentionality, being constructed from a theory of computation but of any sort of useful theory of computation. According to Putnam, theories of computation deal in radically under-determined phenomena. That is, any computational description may be, with the appropriate interpretation, applied to any physical system. The appendix to *Representation and Reality* contains a proof based around the construction of just such an interpretation for an arbitrary system (or open section of space-time). In Putnam's words, "Every ordinary open system is a realization of every abstract finite automata" (Putnam, 1998, p. 122). Assuming that traditional theories of computation are appropriate tools to study the operation of computers (in the sense of electronic artifacts), these theories must work because people have chosen, for whatever reason, the appropriate interpretation to connect operation with physical construction.[28] This is somewhat counter-intuitive proposal. The argument in Putnam's appendix runs as follows. Take any system, arbitrarily defined so long as it is open to the rest of the universe (that contains objects operating as clocks) and operating according to something like Newtonian physics. The exterior clocks may be used to divide the evaluation of the system into a requisite number of time periods (1 for every state through which the system is supposed to evolve) and the states of the system in each period are assigned a state name that can be identified with the state name in the automata that is supposed to be described. Note that in the system, time marks and primitive states of the system ar all arbitrary. In fact, the system states are really names for whatever condition the system is in at time t, rather than descriptions of those states. Presumably, this should be unproblematic, since theories of computation describe the transition between states, and

not only any "content" or internal constitution of these states. Putnam's models of computations are, strictly speaking, Markov rather than Turning type models of computation, as discussed in the first half of this paper.

In order to be successfully described using the vocabulary of computation theory, a system must possess the characteristic of effectiveness. To be effective, a system must have procedures that can be carried out with perfect success (positivity) and repeatability (reliability).[29] This may seem to beg the question of arbitrary description since *any* physical process might, under appropriate interpretation, be described as effective. That is false. Some systems show relevant details at all levels of resolution. Arbitrary systems make very poor computers, no matter what sort of interpretation is used. Other systems, however, may be described using another sort of description since details relevant to their macro-level description and behaviour are manifest on the whole range of possible resolutions.

Putnam's arguments speak to a central issue: computational theories are not literal theories of the physical world, they are under-determined by physical reality. In some trivial cases, this would be unavoidable since, for instance, vacuous components can be added to a description in order to make a new theory that also describes the same system.[30] There is a stronger problem though in that any computational description can be taken to describe any physical system.

David Chalmers (1996) offers a response to Putnam. Chalmers notes, correctly, that the sorts of casual relations between states implicit to Putnam's arguments are not reliable.[31] Because Putnam discusses systems that are open to arbitrary environmental influences (this is one of his assumptions) they will exhibit a sensitivity to initial conditions, the hallmark of an analog system in the physical state. Chalmers also argues that Putnam's proof ignores the requirement that a physical system implement the unrealized states that are explicitly modeled by a computational representation. That is, computational models make counter-factual assertions that are not captured by Putnam's open physical system. Of course, this leads quickly into a consideration of Putnam's metaphysics that would fail outside the scope of this dissertation. Chalmers seeks to reformulate the traditional computational models (specifically Turing Machines) in terms of vectors that capture the sorts of casual relations that would cause a physical system to properly instantiate a computational system in a way that does not fail prey to Putnam's proof. These CSA (Casual State Automata) are theoretically interesting, in that they do provide an answer to Putnam. Practically speaking, they make automatas more complicated and there seems to be little need for more complexity. An understanding of the bridging principles between computational and physical systems should be sufficient. The conditions under which a particular machine can be held to implement a particular model are pretty clear. Chalmer's models captures a few of them, for instance a machine might be held to realize a particular automata equivalent to an algorithm for addition, say, rather than a different automata which implements the same process because the casual relations within the machine more closely follow the steps in the first automata's model.

4. On Communication

The standard models used to describe the actions of computers apply as well to to other sorts of devices, especially those used for communication. Under some interpretations, the definition of computer collapses into that of communication device[32] since the machine's reality is seen as a social construct that enhances human communication, and nothing else. While a telephone (at least of the 'analog' variety) is not a computer, it is an information processing device and is usefully subject to all of the same descriptive models as the computer. Since any communication system with more than two nodes will have to incorporate some variety of switching mechanism, telephones can even be described as Turing Machines.

DEFINITION 2. *A communication device is a machine designed with the explicit intention of delivering information between a source and a destination.*

Communication devices need to be carefully distinguished from computers because they are easily distinguished only in theory and not so easily in practice. There are computers in modern communication devices and most modern computers have internal communication devices. Examples could probably be found between any possible combination and permutation of these two sorts of devices. Nonetheless, the sorts or arguments put forth in this chapter have depended on features of computers no present in communication devices, so it will be necessary to distinguish the two when looking at illustrative examples. Along with the thermostat and the von Neumann architecture conformant computer, the telephone is a test case used to illustrate the various differences. I have allowed an extra degree of abstraction in the definition of these devices because there is simply no other of making sense of them. Thus, while a computer may happen to be a communication device and a communication device could be a computer, they are very different sorts of things. The most important being that a communication device exists only as part of a larger system, that includes identifiable sources and destinations, while a computer is a system to itself.

4.1. COMPARISON

In everyday life, the distinction between computers and communication devices is, at best, vague.

Either sort of device can be a partial constituent of a device of the other sort. Most computers have devices that communicate among their various parts and most modern system of communication contain many computers.

Both communication devices and computers depend on the same sorts of technology. These technologies may be grouped into at least two categories; those that depend on a direct analogy between media (of transition) and phenomena (or single source) and those that use some sort of heteronomous glossary to circumvent the need for such an analogy. Those are, respectively, analog and digital

technologies. Although both sorts of groundings can be used in both computers and communication devices, their final purposes are quite different. A communication device performs the miraculous task of reliably recreating a single at another point. A computer on the other hand, performs tasks with inferential import. Now the signals require some sort of interpretation to be understood, they require a sort of semiotics. It is not a coincidence that Shannon's monograph was titled *The Mathematical Theory of Communication* and not the theory of computation. The distinction between the communication devices and computers also corresponds to the point at which communication theory leaves off and algorithmic theory begins to be useful. When the added requirements of computers are considered, there is a need for a brand of representation that relies on the physical properties of representing tokens.

5. Conclusions

The analysis of terms in natural language is at best suspect. Without some serious work recording usage in various contexts and without the knowledge of the speakers, there's no way to know if my observations about how these words are used is correct for even a small group of English speakers. Even if it is correct, there's no guarantee that the usage of ordinary words corresponds to the natural structure of the world. Communication and computation may be inexact concepts that don't really reflect operations in the world anymore than the sun actually rises in the morning. Continued uses of these figures of speech maybe harmless, but there's philosophical moral to be drawn, at least not without some support from another source.

Perhaps the most immediate is that without some discussion about the ordinary use of these concepts, the similarities and differences between the associated models is not nearly as surprising. Given what is ordinarily taken as computation and communication it is quite surprising that the most obvious difference between the two is that one has deterministic state transitions and the other does not. This surprise may very well accompany a "gee-whiz" moment. These moments are among philosophy's appeals. At worst, its a sort of philosophical joke. Jokes have their place.

Hopefully, there is a deeper point. The concepts of ordinary language may not be the only ones with questionable connections to any underlying reality. The Turing and Shannon models aren't necessarily completely accurate pictures of the processes that are ordinarily referred to as computation and communication. The Shannon-measure of information, in particular, has received considerable criticism. May critics hold that while Shannon's ordinarily talk about information. Ordinary speech is a check to theoretical musings.

Notes

[1]Cartwright (1981). More recently, Cartwright (1999) has expressed a strong skepticism about exactly the sort of models that I will be discussing. Her claim in this recent paper is that models using the sort of screening conditions I will discuss are not universally valid. I do not intend to make any claims to universal validity. If anything, this paper aims to deflate some claims of universal validity

[2]Notation here is a slight variation of the standard presentation of Hopcroft et al. (2001, p. 309).

[3]For example, see Hopcroft (2001, pp. 340–342).

[4]The analogs of q_0, B and F would all be members of this set, but they do not play any special role in the proofs of the theory, so they are generally not given separate entries. S does include spaces and other potentially spacial characters. See especially pp. 39–44.

[5]This is equivalent to the δ above in that it is not one to one, but one to many. The state that the machine enters after entering a transition is determined by a probability measure.

[6]A good starting point to examine the relation of information theory and cryptography is Bruce (1996, pp. 233–237).

[7]See Isaacson (1976) for details. Shannon makes use of all four cases at various times in his presentation.

[8]Much of this discussion is based on Isaacson (1976) but especially pp. 12–23 for a discussion of how this matrix is found (or not found).

[9]Though not always successfully, see Salmon (1998) for a discussion of a few of the more important cases and some suggestions for future applications in probabilistic casuality

[10]Consider a TM for computing the residue of a number modulo 2, or the remainder of some number after being divided by 2. The output of the computation will be either 1 or 0, but the sequence of states that the machine had to go through to get there will be different depending upon the initial input of the computation. If we don't have the initial input, we can exclude most, but not all, sequences as being the path taken.

[11]A Markov process or chain is not the same thing as a Markov algorithm. A Markov Algorithm is a sort of automata that is equivalent to Turing Machines in building a model of computation. For instance, see Curry (1977, p. 70). It does not help that the A.A. Markov (1903–1979) who developed the theory of Markov algorithms is not the A.A. Markov (1856–1922) behind the Markov processes.

[12]These last property means that it is a homogeneous Markov Process.

[13]See, for instance, Fetzer (1994).

[14]Something like this is in wide use. Under most common definitions of algorithm, it reduces to " A computer is a machine that carries out an algorithm".

[15]See, for instance, Haugeland (1998).

[16]Putnam plays on a similar point, the potential ambiguity of what should count as computational processes is a real hurdle to beginning a discussion of computation.

[17]See the discussion at Abelson (1996, pp. 31–50)

[18]Hence, the Kleene Hierarchy.

[19]Alternatively, one could use a different set of basic functions, for example; the zero function, the successor function and various identity functions. See Boolos (1980, p. 73)

[20]This concept is frequently confused with the idea of a von Neumann machine, or self-replicating automata. The two are almost, but not entirely, unrelated.

[21]A register holds values while an accumulator adds a new value to its previous content.

[22]TMs also have two sorts of memory, but the von Neumann Architecture types are distinguished by their access speed not by any sort of theoretical demands.

[23]Though it was not the only solution to the problem. For example Turing's ACE (Turing, 1945) was not von Neumann architecture compliant since it utilised accumulators.

[24]This definition should seem rather strange, almost as if an automobile has been defined as a machine of a particular scale capable of effectively executing surface based navigation and transportation of cargo. It's pretty clear that this sort of definition is better than one depending on the number of wheels or the use of an internal combustion engine. The most appropriate definitions for artifacts refer to capacities to execute rather abstract processes.

[25]Searle (1980) and Putnam (1988) are commonly invoked in this regard.

[26]Although Putnam does not make this claim explicitly, his arguments do point in this direction. The places where he seems to argue that computer science may not raise any philosophical questions are in Chapters 5 and 6, where he develops his attack on functionalism

[27]All references to terrestrial rocks follow Chalmers (1996).

[28]In contrast, I would argue that the theories are important because they are provide appropriate design principles to build the computers. The physical system is constructed to fulfill the interpretation. This would make little sense if any physical system realizes every automata. The difference may only be in how far along the spectrum between pure convention and 'independent' fact the principles of information theory fall.

[29]See Haugeland (1998, pp. 76–78).

[30]Such a move can be illustrated by adding extra states to a TM that the machine will never reach because there are no defined transitions leading to those states.

[31]This is one of the two primary characteristics of digital phenomena, according to Haugeland.

[32]For instance, this would be a consequence of the position developed in Winograd and Flores (1986).

References

Abelson, H., Sussman, G. and Sussman, J. (1996), *Structure and Interpretation of Computer Programs*, Second Edition, Cambridge, MA: MIT Press.

Boolos, G. and Jeffrey, R. (1980), *Computability and Logic*, Second Edition, Cambridge: Cambridge University Press.

Bruce, S. (1996), *Applied Crytography*, New York: Wiley.

Carnap, R. and Bar-Hillel, Y. (1964), 'An Outline of Semantic Information', in *Language and Information*, Reading, MA: Addison-Wesley, pp. 221–274.

Cartwright, N. (1981), 'The Reality of Causes in a World of Instrumental Laws', *PSA* 2, pp. 38–48.

Cartwright, N. (1999), 'Causal Diversity and the Markov Condition', *Synthese* 121, pp. 3–27.

Chalmers, D. (1996), 'Does A Rock Inplement Every Finite-State Automation?' *Synthese* 108, pp. 309–333.

Curry, H. (1977), *Foundations of Mathematical Logic*, Dover.

Fetzer, J. (1994), 'Mental Algorithms Are Minds Computational Systems?' *Pragmatics & Cognition* 2(2), pp. 1–29.

Floridi, L. (2001), 'What Is the Philosophy of Information?' *Metaphilosophy* 33(1/2), pp. 123–45. Also available at http://www.wolfson.ox.ac.uk/∼floridi/index.html

Haugeland, J. (1998), 'Analog and Analog', in *Having Thought: Essays in the Metaphysics of Mind*, Cambridge, MA: Harvard University Press.

Hopcroft, J., Motwani, R. and Ullman, J. (2001), *Introduction to Automata Theory, Languages, and computation*, Second Edition, Boston, MA: Addison-Wesley.

Isaacson, E. and Madsen, R. (1976) *Markov Chains: Theoory and Application*, John Wiley & Sons.

Kleene, S. (1964), 'General Recursive Functions of Natural Numbers', in M. Davis, ed., *The Undecidable*, New York: Raven Press, pp. 237–252.

Knuth, D. (1997), *The Art of Computer Programming*, 3rd Editon, Volume 1, Reading MA: Addison-Wesley.

Putnam, H. (1988), *Representation and Reality*, Cambridge, MA: MIT Press.
Salmon, W. (1998), 'Probablistic Causation', in *Causality and Explanation*, Oxford: Oxford University Press, pp. 208–232.
Searle, J. (1980), 'Minds, Brains and Programs', *Behavorial and Brain Sciences* 3, pp. 417–457.
Shannon, C. and Weaver, W. (1949), *The Mathematical Theory of Communication*, Urbana: The University of Illinois Press.
Turing, A. (1945), 'Proposals for Development in the Mathematics Department of an Automatic Computing Engine', in *Mechanical Intelligence*, Amsterdam: North Holland.
Winograd, T., and Flores, F. (1986), *Understanding Computers and Cognition*, Reading, MA: Addison-Wesley.