# The Myth of the Turing Machine

The Failings of Functionalism and Related Theses

*February, 2002*

*Submitted to JETAI*

*Chris Eliasmith*
*Dept. of Philosophy*
*University of Waterloo*
*eliasmith@uwaterloo.ca*

**Abstract**

The properties of Turing's famous 'universal machine' has long sustained functionalist intuitions about the nature of cognition. Here, I show that there is a logical problem with standard functionalist arguments for multiple realizability. These arguments rely essentially on Turing's powerful insights regarding computation. In addressing a possible reply to this criticism, I further argue that functionalism is not a useful approach for understanding what it is to have a mind. In particular, I show that the difficulties involved in distinguishing implementation from function make multiple realizability claims untestable and uninformative. As a result, I conclude that the role of Turing machines in philosophy of mind needs to be reconsidered.

# The Myth of the Turing Machine

## The Failings of Functionalism and Related Theses

### 1. Introduction

Since its invention in the late thirties, the Turing machine has heavily influenced characterizations of human cognition. The Turing machine often plays a central role in debates about the nature of cognition because many characterizations of cognitive systems are computational and Turing (1937) showed that the machine could be used to characterize all computable functions.[1] Functionalists, in particular, have used Turing's results to argue for the 'multiple realizability' of the mental; i.e., for the thesis that what it takes to be a mind is independent of physical realization. A typical argument for multiple realizability goes something like this:

1. Systems with minds are cognitive systems.
2. Cognitive systems are computational systems.
3. Turing machines can completely describe *any* computational system.
4. Therefore, Turing machines can completely describe any cognitive system (2. and 3.).
5. Turing machines are defined independently of implementation (*i.e.* functionally).
6. Therefore, cognitive systems can be defined independently of implementation (4. and 5.).
7. Therefore, *systems with minds can be defined independently of implementation* (1. and 6.).

This conclusion, associated with the philosophical position of 'functionalism' (or sometimes 'psychofunctionalism'), has become a mainstay in philosophy of mind. However, if we consider the argument closely, especially in light of alternate (though closely related) characterizations of computation, we will see that it is not valid. As a result, the multiple realizability thesis and functionalism itself needs to be rethought. Before considering the argument in detail, I briefly describe Turing machines and their relation to functionalism. I then consider the argument and show that it relies on an equivocation. Subsequently, I consider a way to avoid this equivocation, but this attempt

to revive the argument ultimately fails because it relies on a notion of equivalence that cannot do the conceptual work functionalists need it to. The reason, I argue, is that implementation and function are not as easily parted as multiple realizability claims presume. As a result, such claims are untestable and uninformative. By considering a more implementation-conscious characterization of computation, I suggest that the relevance of Turing machines and the functionalist thesis in philosophy of mind needs to be reconsidered.

**2. Turing Machines on the Brain**

The Turing machine has been crucial to 20th century mathematics, in part because it has played a central role in theories of computation and computability. Specifically, the Turing machine helps provide a rigorous definition of an algorithm or method. Turing was able to show that his 'machine' describes a mechanical process that can perform all of the operations a person working with a logical system can perform. Subsequently, Alonzo Church formulated the Church-Turing thesis; namely, the thesis that all definitions of computability are equivalent. Together, these results demonstrate that Turing machine computable functions are all the computable functions there are. Thus, Turing had succeeded in specifying a *universal* computer.

Turing machines are as simple as they are familiar: they consist solely of a tape, a read/write head and a finite table of state changes (see table 1). The tape is divided into discrete boxes, each of which may have either a zero or a one in it. The head will read a zero or one from the current tape square and, depending on the current state and what is in the current tape square it will write a zero or one, move left or right and proceed to the next state in the table. Consider, for example, the state table below (table 1). This Turing machine description defines an algorithm for a simple adder. Given two strings of ones separated by a zero, it will output a single string that is the same length as the sum of the original two strings. So, for example, the string [ 1 0 1 ] becomes [ 1 1 ] and [ 1 1 0 1 1 1 ] becomes [ 1 1 1 1 1 ].

| State Number | Input | Output | Next State |
|---|---|---|---|

| | | | |
|---|---|---|---|
| 1 | 0<br>1 | R1<br>R1 | 2<br>1 |
| 2 | 0<br>1 | L0<br>R1 | 3<br>2 |
| 3 | 0<br>1 | L0<br>L0 | 3<br>4 |
| 4 | 0<br>1 | R0<br>L1 | HALT<br>4 |

**Table 1**: A Turing machine state table for a simple adder.

Turing himself argued that a Turing machine could achieve whatever the human brain could achieve (Turing 1950). Because he had shown that Turing machines can compute any function that is computable (assuming that both the tape and time are infinite), and given the further claim that human cognition is a product of biological computation, Turning concluded that all of our cognitive behaviors can be captured in the language of the Turing machine. So, for any mental process there must be a Turing machine description that will have the same input/output relations.

Central to the power of Turing's formulation of the Turing machine is the fact that it provides a computational description *without any reference to the physical makeup of the computer*. In table 1 there are clearly no physical constraints placed on the implementation of the state table. So, to *fully characterize* the behavior of a Turing machine at a time we need specify only three things: 1) the input at that time; 2) the state of the machine at that time; and 3) the state table. Notably, what makes a machine state the type of state it is (e.g. an adder), are functional relations between inputs, outputs, and other machine states.

Following Turing's lead, functionalists analogously hold that what individuates mental states are functional relations between inputs, outputs, and other mental states (Putnam 1975). Functionalists thus suggest that cognitive functions can be completely characterized by high-level descriptions, abstracted from (i.e., independent of) their implementation. Coupled with a notion of functional equivalence (or 'functional isomorphism' to use Putnam's phrase), this characterization of mental states establishes the cherished multiple realizability thesis. That is, if two systems have functionally

equivalent descriptions, then they have the same set of mental states (if any). Because functional descriptions are independent of implementation, two different implementations can have functionally equivalent descriptions. Thus, two different implementations can have the same set of mental states (if any); this just is the thesis of multiple realizability. Note the central role played in this argument by the notion of 'functional equivalence' – it is this notion that, I will argue, is ultimately unable to do the work functionalists need it to (see section 4). [2]

It is hardly necessary to argue for the prevalence of these positions in current philosophy of mind (see Block 1980; Fodor 1981; Cummins 1983; Putnam 1994; Stillings, Weisler et al. 1995). Ned Block has, in fact, claimed that functionalism and multiple realizability are obvious enough to be considered default positions in philosophy of mind (Block 1980, p. 178):

> [I]t is a simple matter to dream up a nomologically possible machine that satisfies [a given] machine table but does not have the designated physical state. Of course, it is one thing to say this and another thing to prove it, but the claim has such overwhelming prima facie plausibility that the burden of proof is on the critic to come up with reasons for thinking otherwise.

Indeed, the majority of philosophers of mind embrace some form of functionalism (and with it the multiple realizability thesis) and so it will take some effort to show how, precisely, functionalism fails.

### 3. A Logical Slip

Perhaps the most obvious assumption of the multiple realizability argument outlined in section 1 is that cognitive systems are computational. I wish to grant this assumption – in a sense. In precisely this sense: all systems we consider to be cognitive systems are things-that-compute. What is avoided by this formulation is the ambiguity of the term 'computational system'. The term could also be taken to mean 'a system of computations'; i.e., the abstract set of input to output transformations. Noticing this difference, it is clear that premise 3 is true only under the second interpretation. Thus, Turing machines completely describe *computations*, not the *physical systems* that are

performing computations. It is clear that the antecedent premise (premise 2) is using the term 'computational system' to refer to things-that-compute, since real cognitive systems are physical systems and systems of computations are not. So, there is an equivocation on the terms 'computational system', rendering the argument invalid.

Of course, it is still possible to save this Turing machine-motivated argument, and perhaps functionalists would make the necessary additional claims. In particular they may claim that the equivocation noted above is irrelevant. They may argue that the equivocation does *not* nullify the fact that there are *systems of computations* that could be implemented by different things-that-compute. That is, we *can* construct different implementations that have equivalent (or even identical) Turing machine descriptions. If, by completely describing the system of computations, we *have* completely described the cognitively relevant aspects of those things-that-compute, then the argument will survive since nothing would ride on the equivocation. I argue in the next section that the standard notion of equivalent Turing machines does not, in the end, support this conjecture. And, because the notion of equivalence fails in this context, the claim that the equivocation is irrelevant cannot be sustained.

**4. Are All Equivalent Systems Created Equal?**

In section 2 we saw that intuitions supporting multiple realizability stem from rely to the notion of the "equivalence" of computational systems. But when, exactly, are Turing machines equivalent? And, what kind of equivalence is necessary to eliminate the equivocation discussed above? Recall that premise 3 of the argument to multiple realizability is: "Turing machines can completely describe *any* computational system." Where by "computational system" we mean things-that-compute. And by "completely describe" we mean something like: a description that captures the cognitively relevant behaviors of a system under a set of conditions. So, if we take any two things-that-compute and they are implementations of equivalent Turing machines, we should expect them to have cognitively similar behavior under similar conditions. Of course, if

they are implementations of *the same* Turing machine, we have good reason to expect their cognitive behavior to be as similar as possible.

I return to this point shortly, but first let us consider more recent results in computational theory that bear directly on considerations of computational equivalence. Kolmogorov has shown how important the *implementation* of a computational system is to its computational character. He has proven that two implementations of a given functional description can not be usefully considered equivalent unless they are almost *identical*. This is because an algorithm running on one implementation can only be run by another implementation with the addition of an *emulator* program of some sort. Running this emulator adds computational complexity[3] making the second implementation significantly different from the first (Le Cun and Denker 1992). It is *only* in the limiting case of *infinite* symbol-strings that this overhead can be ignored (a limiting case often adopted by Turing machine proofs). For *finite* strings, this overhead will significantly affect the performance of the computer – especially if we place *time* and *resource* restrictions on its behavior. Such restrictions seem to be the rule more than the exception in the case of cognitive systems (Newell 1990; van Gelder 1995; van Gelder and Port 1995; Eliasmith 1996).

So, the structure of a computer may not affect what *class* of functions is computable for that computer, however, its structure *will* significantly affect computational complexity. Since an increase in computational complexity necessitates an increase in the time and power needed to perform a computation, the class of actual computable functions *within a given period of time and with a fixed amount of computational resources* will vary for different physical computers. It is in this way that "equivalent" computers are significantly *not* equal. Implementing equivalent or even identical Turing machines does not at all guarantee equivalent computational complexity. In this respect, implementation and function are intimately bound. This is true *even for abstract computational descriptions* if time and resources are included in such descriptions. So, claims of Turing machine equivalence tell us little about computational

complexity and it is computational complexity, not mere function, that is cognitively relevant.

To see this, consider the example of deciding whether an object in the environment is a friend or foe. Suppose we have two different implementations of the function that needs to be computed to successfully achieve this recognition. One of these implementations, Athlon Alan, can compute this function in less than a second given it's architecture, computational primitives, and so on. The other implementation, Intel Alan, takes nearly 10 minutes to perform the same computation because it's architecture, computational primitives, and so on aren't optimized for this kind of computation. In other words, the computational complexity of the algorithm on the second implementation is significantly higher than on the first implementation.[4] Of course, if the object is a foe, Intel Alan may not have the 10 minutes required to make this decision and thus may not ever exhibit this cognitive behavior. As a result, it is computational complexity, not function, that determines possible cognitive behavior. Merely noting the class of abstract functions computable by Intel Alan won't tell us much about Intel Alan's actual cognitive behavior.

**5. The impropriety of Turing's machines**

Because functionalists have insisted that computable functions are cognitively relevant, they have relied on Turing equivalence, a notion that is both *uninformative,* and *untestable* as typically applied to cognitive systems by functionalists. In other words, functionalists have improperly applied Turing's theoretical results to understanding minds.

So see this, let us again consider the Turing machine description in table 1. As previously discussed, this is the state table for an adder; let us call the state table Alan. So we have Alan Turing machine, a *theoretical* entity with four states that can be described by table 1. Of course, Alan and his tape can be implemented in silicon, or with water flowing through a series of pipes. Suppose Alan is implemented in both water and

silicon. Then we can know that silicon Alan and water Alan have equivalent (in fact, identical) Turing machines descriptions.

In trying to understand the brain, we are obviously not provided with the state table and tape that we are trying to characterize. So, to render this thought experiment more amenable to the situation we are in with natural cognitive systems, suppose we *find* water Alan and silicon Alan, without already knowing that they are implementing the same Turing machine. We must now set to work to find out what these two recently discovered systems are computing, i.e., what input, output, and state transition relations hold. If we begin by characterizing these relations at the level of electrons, atoms, or molecules, we will undoubtedly get *different* Turing machine descriptions of the two systems. Such descriptions will be enormously complex. Indeed, we may never find those four simple states that the two Alans' designers were trying to implement. Given these considerations, it is not surprising that we can provide an infinite number of Turing machine descriptions of both systems (Dennett 1978).

To make things worse, let us suppose that there is another system, Nala, that is an implementation of a state table that has been randomly selected from the vast space of possible Turing machines. Since, once implemented, there are an enormous number of Turing machine descriptions of this system as well, it is likely that we would be able find a Turing machine description for Nala that matched one for either water or silicon Alan. In fact, this would be just as likely as it would be for us to find descriptions for water and silicon Alan that matched. If we did find matching descriptions for Nala and, say, water Alan, we could rightfully claim that Nala and water Alan are Turing machine equivalent.

Together, Alan and Nala can help us discover how Turing equivalence is uninformative and untestable. The reason Turing equivalence is uninformative is that it is both too easy and too hard to find in physical systems. The example of water and silicon Alan shows how equivalence can be too hard to establish – matter simply has too many input, output, and state transition relations to know which are the right ones for a given analysis. The example of Nala shows how equivalence can be too easy to establish

– any matching input, output, state transition mapping supports an equivalence claim. The notion of equivalence is so under-constrained as to provide no means for making principled distinctions in our descriptions of unfamiliar computational systems, of which the brain is presumably one example.[5]

The problem is that the designers of water and silicon Alan could not implement just four states, without implementing an infinite host of others; such is the nature of physical stuff. Any physical system can be described in terms of a near infinity of different virtual machines. As a result, we can't be in a position to tell which one is explanatorily relevant. As the example of Nala shows, Turing machine characterizations are cheap. That means that for unknown systems, they will not help determine how to draw a function/implementation distinction in an explanatorily useful way; all such characterizations (mis-) describe a system equally well. But functionalists assume that there is such a distinction to be drawn; without that distinction, the multiple realizability thesis would make no sense. But the distinction can not be supported *in practice* precisely because real physical systems implement not one, but infinitely many Turing machine defined functions.

A functionalist may claim that this is fine as long as that infinity includes some subset of functions of interest, e.g. the subset necessary for supporting a mind. But the same two problems arise. First, it is likely that we can find whatever subset is specified in a host of implementations we do not want to count as minds. Second, we can never test such an hypothesis since we can never implement *just* that subset of functions. Such claims are thus uninformative (because both things with and without minds will have those functional descriptions) and untestable (because we can not implement just that functional description). Simply put, there is no way around the fact that the set of a system's functions and its implementation are intimately related.

It follows that the function of a given implementation can only be successfully divorced from that implementation in the realms of mathematical theory (although it doesn't need to be, as Kolmogorov's result shows). As soon as we build something (or

even make design decisions), the materials we choose to build it with and the way we put it together will determine what functions it can realize. For example, if we wish to build a machine to catch a baseball, we cannot simply specify the states through which the machine must move; those states *have* to be moved through in a given period of time (i.e. with a specified computational complexity given certain resource constraints), or that function cannot be *actually* performed.

More generally, if we add to a functional description some function whose performance depends on a dimension along which two implementations are non-identical, they will no longer both be able to implement the *exact same* functions. The reason is that, *whereas Turing machine descriptions are dimensionless, Turing machine implementations are not*. The significance of this realization is most evident once we acknowledge that many functions have a distinctly temporal nature. In such cases, the speed with which a given system can compute helps determine those functions computable by that system[6]. If implementation affects computational speed, it affects computable functions; and there is a huge range of factors, from computational complexity to mass, which affect computational speed.

**6. Conclusion**

Without a robust notion of equivalence, functionalism as standardly construed is not a useful means of understanding what it is to have a mind. Turing machine equivalence, I have argued, is not a robust notion in this context. However, this is the notion that functionalists typically adopt. As a result, the functionalist argument to multiple realizability fails because Turing machine-type functional descriptions simply are not *complete enough* descriptions of physical implementations.

As a result, the role that Turing machines play in our understanding of what it takes to be a cognitive system needs to reevaluated. While Turing's insights can be useful for specifying a class of potentially computable functions, this class is unlikely to tell us much about the set of real systems we deem to count as cognitive. Instead, we need a

notion of (Kolmogorov) equivalence based on considerations of computational complexity and computability. Specifying boundaries on temporal and/or computational complexity within which a system must fall to count as cognitive is essential to a useful taxonomy of minds. This is because there is no getting around the fact that in the real world the set of functions realized by a given physical system is going to depend significantly on the physics of that system.

**7. Acknowledgements**

Special thanks to Charles Anderson, William Bechtel, Andy Clark, Valerie Hardcastle, Pete Mandik, Jesse Prinz, and Chase Wrenn for insightful discussions and comments on earlier drafts of this paper. This work was supported in part by the McDonnell Project in Philosophy and the Neurosciences and the Social Sciences and Humanities Research Council of Canada.

**8. References**

Block, N. (1980). Introduction: what is functionalism? *Readings in philosophy of psychology*. N. Block. Cambridge, MA, Harvard University Press. **1:** 171-184.

Cummins, R. (1983). *The nature of psychological explanation*. Cambridge, MA, MIT Press.

Dennett, D. (1978). *Brainstorms*. Montgomery, VT, Bradford Books.

Eliasmith, C. (1996). "The third contender: a critical examination of the dynamicist theory of cognition." *Philosophical Psychology* **9**(4): 441-463.

Eliasmith, C. (2000). "Is the brain analog or digital?: The solution and its consequences for cognitive science." *Cognitive Science Quarterly* **1**(2): 147-170.

Fodor, J. (1981). *Representations*. Cambridge, MA, MIT Press.

Le Cun, Y. and J. S. Denker (1992). "Natural versus universal probability, complexity, and entropy". *IEEE Workshop on the Physics of Computation*.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA, Harvard University

Press.

Putnam, H. (1975). The nature of mental states. *Mind, language and reality*, Cambridge University Press**:** 429-440.

Putnam, H. (1988). *Representation and reality*. Cambridge, MA, MIT.

Putnam, H. (1994). *Words and Life*. Cambridge, MA, Harvard University Press.

Searle, J. R. (1990). "Is the brain a digital computer?" *Proceedings and Addresses of the American Philosophical Association* **64**: 21-37.

Stillings, N. A., S. E. Weisler, et al. (1995). *Cognitive science: an introduction*. Cambridge, MA, MIT Press.

Turing, A. M. (1937). "On computable numbers, with an application to the entscheidungsproblem." *Proceedings of the London Mathematical Society* **42**: 230-265.

Turing, A. M. (1950). "Computing machinery and intelligence." *Mind* **59**: 433-460.

van Gelder, T. (1995). "What might cognition be, if not computation?" *The Journal of Philosophy* **XCI**(7): 345-381.

van Gelder, T. and R. Port (1995). It's about time: An overview of the dynamical approach to cognition. *Mind as motion: Explorations in the dynamics of cognition*. R. Port and T. van Gelder. Cambridge, MA, MIT Press.

---

[1] Technically, Turing's result only applies to digital, or discrete state, machines. However, I take it that the brain can be successfully characterized as such a machine (Eliasmith 2000).

[2] It is not unanimous amongst teleological functionalists that a notion of equivalence is mandatory. For those who do not commit themselves to a notion of equivalence, the arguments in section 4 should be considered merely a warning.

[3] Computational complexity is roughly a measure of the number of steps it takes to complete a computation.

[4] This is true despite being able to give an abstract characterization of the algorithm. In order for that characterization to be realized, the *actual* algorithm being run on the computer will depend intimately on the computer's architecture.

[5] These arguments are reminiscent of Searle's (1990) rejection of computation all together (see also Putnam 1988). However, I simply wish to show the generality of the notion of Turing equivalence and don't need nearly as strong a result as Searle does. Searle's point is rejected by some because it seems outlandish to suppose that we could ever figure out how our office wall was running a word processor.

My point, in contrast, is that even given a machine that is running a word processor, Turing equivalence can't provide useful constraints for determining that this is so.

[6] More abstractly, consider that a function can be mathematically defined as $F:X \circledR Y$ where $X$, $Y$ and $F$ are the sets $X=\{x_1,\ldots,x_n\}$, $Y = \{y_1,\ldots,y_n\}$, and $F$ is some set of ordered pairs of elements of $X$ and $Y$. We must, in addition, realize that in an implementation all members of $X$ occur at a time $t$, and thus this mapping is really one from vectors, $\mathbf{x}_i \hat{I} X$, to vectors, $\mathbf{y}_i \hat{I} Y$, where $\mathbf{x}_i = \{x_i,t_i\}$, $\mathbf{y}_i = \{y_i,t_i\}$. If we don't preserve *this* mapping, we aren't computing the same function.