Comments on Gualtiero Piccinini's 'The mind as neural software'

Presented at the Southern Society for Philosophy and Psychology
Annual Meeting, April 5-7, 2007, Atlanta

Whit Schonbein
Department of Philosophy
College of Charleston
schonbeinw@cofc.edu

To begin, thanks to Dr. Piccinini for the stimulating paper; it offers a number of welcome clarifications and criticisms of the often misunderstood claim that 'the mind is the software of the brain'. However I think a bit more clarification is in order, and in these comments I'll draw attention to two issues that emerge from the various types of computational functionalism described by Piccinini.

In the first half of the paper Piccinini is concerned with rejecting the 'standard formulation' of computational functionalism (Standard-CF) on the grounds that it relies on a mistaken notion of software implementation. The notion used by Standard-CF leads to certain problems, such as allowing that a single system could execute multiple programs simultaneously. In its place, he offers a revised version that attempts to make sense of the informal claim that the mind is the software of the brain: program-execution computational functionalism (Program-Execution-CF). He then goes on to generalize Program-Execution-CF so that it no longer requires an appeal to program execution; call this final version 'generalized computational functionalism' (Generalized-CF).

As I understand it, these three versions of computational functionalism are:

> *Standard-CF*: The mind is the software of the brain, where software is implemented according to a mapping relation between physical descriptions and computational descriptions.
>
> *Program-Execution-CF*: The mind is the software of the brain, where software is implemented via a mechanism that executes a program, in the sense familiar from the theory of computation (i.e., in the same sense that a universal Turing machine simulates another Turing machine).
>
> *Generalized-CF*: The mind is the computational organization of the brain.

So, Piccinini's argument is that (i) the notion of software implementation appealed to in Standard-CF is incorrect, and that (ii) software implementation is more appropriately understood in terms of Program-Execution-CP. Furthermore, once this distinction is appreciated, we can articulate a theory of computational functionalism that does not rely on program execution.

In these comments I will highlight several concerns I have about the relationship between these theories.

1. Piccinini argues that Standard-CF is flawed, and offers as a replacement Program-Execution-CF. However, it seems to me that Program-Execution-CF implies Standard-CF, and consequently, if Standard-CF fails, so does Program-Execution-CF. Here is why: First, when we say that the mind is the software of the brain, the description relevant to understanding mentality is the description of the software, not a description of the hardware that executes it (for if the relevant description were the hardware, then clearly it is not the software that is responsible for mentality). Second, the theory of computation tells us that for every program executed by a universal machine, there is a non-universal machine that instantiates (rather than executes) that program. For example, suppose a universal Turing Machine U simulates a Turing Machine T. Then there is a Turing Machine that doesn't simulate T – it just *is* T. Furthermore, the description of the program executed by U is true of T as well, because the input to U just *is* a description of T. In other words: If there is a machine the executes program P, then there is a machine that instantiates program P without running it. P is a description of both the program executed by the universal machine, and a description of the machine that instantiates P without running it.

This means that, if an entity has a mind by virtue of executing a program, there is another possible entity that instantiates that very same program without executing it. Since the description of the program is what is relevant to capturing mentality, and that description is true of both entities, then it seems that both of them – the program-execution entity and the program-instantiation entity – have minds. In other words, anyone who endorses Program-Execution-CF should likewise endorse Standard-CF.

Unfortunately, in that case the problems afflicting Standard-CF will plague Program-Execution-CF as well. For example, one purported problem with S-CF is that the implementation relation presupposed by that hypothesis allows for the same system to satisfy multiple computational descriptions (Piccinini, p. 4). This problem appears to recur for Program-Execution-CF, because the program being run by a universal machine will also satisfy multiple computational descriptions, just as that program being instantiated by a non-universal machine does. Suppose, for example, we describe the program using the language of C++. Since Pascal is equivalent to C++, there is a description of the program in the language of Pascal. Similarly for other equivalent programming languages, actual or possible. So, if the problem of multiple program instantiation is a problem for computational functionalism, then it remains a problem, even if we get what it means to execute a program right.

To keep the two theses distinct, what the proponent of Program-Execution-CF needs is a way to deny that the program-instantiation entity has a mind. That is, there must be something special about *executing a program* that is relevant to mentality, something that is lacked by a machine that simply instantiates that same program. An obvious way to do this is to deny the truth of the initial premise: When we say that the mind is the software of the brain, we do not really mean that the description of the program captures all that matters for mentality. Instead, other architectural (i.e., implementational) requirements must be satisfied as well: The entity must have a physical construction capable of genuinely executing programs rather than simply

realizing them. That is, the machine must be a 'universal' machine rather than the machine it simulates.

Piccinini seems to go both ways on this issue, sometimes writing as if Program-Execution-CF does include a claim about the relevant architecture in addition to the standard claim about running the right program, and at other times writing as if the program is all that matters for mentality. For example, he says that explanations in Program-Execution-CF are a kind of mechanistic explanation, where "the postulated mechanism is one that possess a program (or set of programs) and one or more processors, which together determine how the mechanism processes its data" (p. 13). Here it seems as if mentality requires more than merely the right program – it also requires an architecture capable of executing programs – an appropriate mechanism. In contrast, he also writes on the same page, "It may surprise some readers that when interpreted literally, computational functionalism entails that the mind is a physical component (or a stable state of a component) of the brain, in the same sense in which computer programs are physical components (or stable states of components) of computers." (p. 13) This second quotation appears to assert that what is important for mentality is the program description alone: The program is stored in memory, which is only part of the architecture of a machine that can execute programs, and hence the mind is really only part of the brain.[1]

So my first question is this: Does Program-Execution-CF contain an architectural requirement (namely, that the machine be a universal machine), or is mentality still tied solely to the program? And if the latter, wouldn't objections to Standard-CF apply to Program-Execution-CF as well?

[Addendum: In an email reply to the initial version of these comments, Piccinini confirms that, on his account, Program-Execution-CF does not imply Standard-CF, pointing to the fact that the former involves an appeal to a specific type of mechanism – namely, one that can execute programs. So I take it that on his account it is the case that Program-Execution-CF goes beyond what seems to be the standard notion of computational functionalism by positing an implementational requirement for mentality. This seems to me to go well beyond merely clarifying 'the mind is the software of the brain metaphor', as it adds a substantive and potentially problematic additional requirement – see below.]

Piccinini introduces a new notion of computational functionalism that seems to answer my first question: *dynamic program-execution computational functionalism* (Dynamic-CF). According to this version, "the mind is constituted by the *processes* generated by the brain's software." (p. 14, italics in original). Piccinini does not discuss this version in depth, but one way to interpret the proposal is that satisfying the program description alone is not sufficient for mentality; instead, that program must be *running* on some programmable hardware in the sense that causal[2] processes between those parts of the mechanism that constitute the physical storage of the program and the hardware that runs the program are instantiated. That is, there is something about the causal structure

---

[1] Piccinini's comments on multiple realization (p. 13, n. 16) are another example of treating the program as the locus of mentality.

[2] Here I am assuming the relevant processes are causal, as Piccinini does not discuss it, and it sees like a natural candidate: The difference between a program running and not running is that in the former case the program is involved in the causal structure of ongoing computation whereas in the latter it is not.

described by a program being actually instantiated on a general purpose machine that is important to mentality. This notion of computational functionalism answers my earlier question insofar as it clearly requires that the machine's architecture be of a certain general sort – a non-programmable architecture cannot have a mind, because it cannot execute a program stored in memory, i.e., it lacks the requisite causal processes.

Now, Dynamic-Program-Execution-CF seems to me to present its own special set of problems, chief among them being that it seems to preclude the possibility that minds are multiply realizable. Since different types of programmable hardware running the same program will instantiate different types of causal processes, and mentality resides in these causal processes rather than the program itself, those processes are not multiply realized across different hardware executing the same program. Obviously there are different approaches one could take in response to this issue, such as rejecting (radical) multiple realizability, or by attempting to give a principled account of the relevant causal processes such that all programmable machines share them (hence preserving multiple realizability).

But there's a more fundamental question for the Program-Execution-CF (dynamic or otherwise) proponent: Why should being programmable matter so much for mentality in the first place? On what grounds do we deny that an entity that is hardwired to behave in certain ways has a mind, whereas an entity that behaves in exactly the same way in all the same circumstances, in accordance with the same program description, has one? In other words, why should it be relevant for mentality that a system *could* run a program different than the one it is *actually* running? It seems that on its usual interpretation, regardless of whether the machine is universal or not, it is the program that the machine is actually running that is relevant for mentality. If this is right, then some additional argument is required to establish relevance to mentality of a capacity to run a different program than the one it is currently running.[3]

To sum, my first worry is that either Program-Execution-CF is not conceptually distinct from Standard-CF (and hence is subject to the same criticisms) [but see the bracketed addendum, above], or Program-Execution-CF is unmotivated.

2. Even if the Program-Execution-CF is unmotivated, at least the second interpretation (where creatures with minds must be mechanisms that execute programs) distinguishes between two classes of system: Those that execute programs (e.g., Universal Turing Machines), and those that do not (e.g., regular Turing Machines). However, Piccinini later generalizes Program-Execution-CF to arrive at a version of computational functionalism that no longer requires program execution. Instead, with Generalized-CF, program execution is replaced with "other computational processes" (p. 16), such as those exhibited by connectionist networks or finite state automata.

Without further elaboration on Piccinini's part, it is unclear exactly what these

[3] One possibility is to require that the machine be capable of modifying its own program, in which case the architecture would have to be programmable at some level. But (i) this would amount not only to a restriction on the type of architecture, but also a restriction on the type of program, which is not the thesis described by Piccinini, and (ii) the question recurs: Why is a (potentially unexercised) capacity to rewrite one's own program relevant to mentality? (This is of course a different question than whether humans do in fact have this capacity – even if they do, it still requires additional argument to show that such a capacity is essential to mentality.)

alternative computational processes are. However, relaxing the definition in this way makes me curious as to precisely how the generalized form of computational functionalism is supposed to differ from Standard-CF. In particular, if the motivation to reject Standard-CF comes from problems surrounding the standard notion of program implementation, and Generalized-CF does not have access to the putative solution provided by Program-Execution-CF, then what notion of implementation are we to appeal to? It seems to me that the only(?) option is the standard notion rejected at the start of the paper.

Take, for example, finite state automata (FSA)[4]. If a physical system realizes a FSA, then it either does so by executing a program that realizes the FSA, or it realizes it directly. As I read Piccinini, the former option is possible but not required by Generalized-CF, hence the system could realize it directly. According to common practice in the theory of computation, a system directly realizes a FSA when there is a mapping between states of the system and states of the FSA, and the transitions between states of the FSA are isomorphic to transitions between states of the physical system.[5] But this notion of direct realization just is the standard notion of software implementation identified and rejected by Piccinini early in the paper.

So my second worry is that, by generalizing Program-Execution-CF, we again end up back where we started, with a version of computational functionalism that is vulnerable to the very same objections raised at the beginning of the paper.

[4] As I write this, I have in mind deterministic finite automata in particular, because they are the simplest type of FSA. I don't think that makes any difference in my reasoning, but I thought I'd mention it nonetheless.

[5] This is obviously a very rough account of realization. I left out the details because it would take too long to explain them.