Olympia and Other O-Machines*

Colin Klein
Department of Philosophy
Macquarie University
colin.klein@mq.edu.au

Abstract

Against Maudlin, I argue that machines which merely reproduce a pre-programmed series of changes ought to be classed with Turing's O-Machines even if they would counterfactually show Turing Machine-like activity. This can be seen on an interventionist picture of computational architectures, on which basic operations are the primitive loci for interventions. While constructions like Maudlin's Olympia still compute, then, claims about them do not threaten philosophical arguments that depend on Turing Machine architectures and their computational equivalents.

*1505 Draft 2De1, compiled May 22, 2015

1 Olympia

Not everything that behaves like a Turing Machine (TM) counts as one. The dividing line is not obvious. Tricky cases are philosophically interesting. Tim Maudlin's *Olympia* is the most notable of an especially vexing class of examples Maudlin (1989). Olympia implements a TM by standard counterfactual criteria. Yet intuitively, she does not compute, at least in the interesting way that TMs do.

Olympia consists of three parts: a (potentially infinite) tape, an armature, and a slew of auxiliary Turing machines. Left alone, the armature will blindly transform the tape in accordance with pre-programmed information. Such transformations are identical to those that would have been caused by a TM running a program P started on input i. If the starting input is i, the armature makes its rounds, transforming the tape in accordance with pre-programmed information. The result is identical to what a TM running P would have done to i—not just the output, either, but the step-by-step changes to the tape. Started on an input other than i, Olympia's auxiliaries stop the armature and compute P on the actual input by acting as Turing Machines running P. (So described, Olympia is a schema into which many Ps and is could be filled; for convenience, I will speak as if there is a single such instance.)

Maudlin claims, and thinks everyone should agree, that the activity of the armature in isolation would not be sufficient for computation. How could it be? The armature alone is entirely insensitive to the input; its brute activity is a parlor trick, below the level of real computation. Together, however, the armature and the auxiliaries make a formidable machine. If Olympia is started on i, the armature winds out changes in its usual way. Had it not been the case that she was started on i, though, the transformations of the tape would still have been appropriate to P. Olympia as a whole thus *is* a Turing machine, claims Maudlin. The changes in the tape are isomorphic to those produced by an ordinary TM running P, and the appropriate input counterfactuals are true; this is sufficient to be a TM.

If so, a number of interesting consequences follow. The auxiliary TMs are entirely inactive on a run of P on i. So the difference between a TM and something that does not compute (that is, between the bare armature and Olympia as a whole) is a difference in wholly inert machinery. That does not play nicely with various plausible theses about consciousness.

I will not be concerned with the supposed further consequences. Each of them depend on the claim that Olympia is a Turing Machine. Many have found this implausible, precisely because the activity of the armature itself is so thin. Most responses thus attempt to bar Olympia by further restricting implementation;[1] Without some deeper explanation of what's wrong with Olympia, however, even descriptively adequate conditions inevitably seem *ad hoc*.

I think there is a better response to Maudlin. One can concede that Olympia computes, but deny that Olympia is anything like a Turing machine. Indeed, Olympia is something far stranger.

[1] I argued as much in my (2008), and (Bartlett, 2012) has a nice review and critique of such moves.

2 Trurl

Consider a class $\mathcal{C}$ of machines that behave like TMs on all but one special input. On the special input, they transform the tape according to the instructions in a pre-programmed array. Machines in $\mathcal{C}$ can be specified by a triple $\langle P, i, S\rangle$, where P specifies the machine table for a TM, i is the special input, and S is a possibly infinite array $[j_1, k_1, j_2, k_2 \ldots]$. When started on inputs other than i, such machines simply follow P and act as TMs. Started on i, they transform the tape in accordance with S: at the nth computational step, the j_nth tape square is changed to read k_n.

Olympia is a member of $\mathcal{C}$; S is simply a specification of the tape changes performed by the chosen program P on i. Note, however, that $\mathcal{C}$ does not generally restrict S to transformations made by P run on i. S simply specifies a (possibly infinite) ordered set of changes to be made to a tape.

Consider another member of the same class, *Trurl*. For Trurl, let P = the identity function, $i = 0$, and S be defined such that j_n is '1' if the nth Turing machine (on some complete enumeration scheme) halts given n as input, and is '0' otherwise. Is Trurl a Turing Machine? Clearly not. Started on an empty tape, Trurl solves the halting problem. No TM can do that. So Trurl is more powerful than any Turing machine. Similarly so with the class $\mathcal{C}$. Some machines in $\mathcal{C}$ outstrip the power of Turing Machines, while any Turing-computable function can be trivially computed by machines in $\mathcal{C}$. So the class $\mathcal{C}$ is more powerful than the class of TMs.

The result should be unsurprising. The class $\mathcal{C}$ is computationally equivalent to the class of *Oracle machines* (O-machines for short). Oracle machines, as defined by Turing, are Turing machines augmented with a primitive operation (the 'oracle') that returns the value of some particular function on the natural numbers. Oracles work by "unspecified means," and are not restricted to the Turing-computable functions. Hence their power Turing (1938).

What, then, of Olympia? I claim that she is an O-machine, not a Turing machine. The architecture she belongs to is an O-Machine architecture: the built-in transformations available to her place her in this stronger class. We should thus count her among the O-machines (and their equivalents) and not the Turing Machines (and their equivalents).

3 Olympia is an O-Machine

To my claim about Olympia, I anticipate three objections. First, that Olympia is restricted to computing only the Turing-computable functions and therefore does not have the additional power that makes Trurl so problematic. Second, that Olympia's operation is isomorphic to that of a Turing Machine, and that this isomorphism is the only criterion that matters for being a TM. Third, that the Church-Turing thesis prohibits the existence of O-machines in worlds like ours.

To the first objection, I respond that the restrictions on Olympia make no difference to the *type* of computing machine she is. Olympia's activity is not materially different from Trurl's: on at least one input, both rely on the availability of the pre-programmed tape transformation encoded in S. That Olympia's pre-programmed transformations happen to coincide with those which would be performed by a Turing Machine shows only the power of oracles to reproduce such sequences.

Upgrading Olympia to a machine that computed *non*-Turing-Computable functions would be a trivial matter: one would need to change only the value stored in the oracle, not the computational activity Olympia performs. In contrast, no Turing Machine could be so trivially altered: no amount of fiddling with machine tables will suffice to improve a TM to Trurl's level.

Indeed, the point can be made without recourse to especially powerful functions. We classify machines not just to show their maximum power but also, at a more fine-grained level, to make claims about the required time, space, and other properties of algorithms run on them. These questions belong to computational complexity theory, and are an important additional level of classification of machines beyond the set of functions they can compute Aaronson (2015).

Suppose that Olympia's pre-programmed transformations consist in the serial enumeration of the value of some trivial function: say $f(x) = 1$ if the leftmost digit in the decimal representation of x is even, and 0 otherwise. Consider another machine, Klapaucius, which on the special input lists sequentially the values of the function $g(x) = 1$ if the leftmost digit in the decimal representation of the Ackerman function $A(x, x)$ is even and 0 otherwise. Both f and g are Turing Computable. Yet due to the recursive depth of the Ackermann function, ordinary machines computing g require massive amounts of time and space for even small values of x. If Olympia and Klapaucius were TMs, then Klapaucius would, for any particular x, require far more time and space than Olympia to finish. As O-machines, Olympia and Klapaucius would require exactly the same amount of time and space to compute any particular value of x. Again, Olympia has very different properties than an ordinary TM, and is best classed with her more powerful brethren.

To the second objection, I respond that being isomorphic to a Turing Machine is plainly insufficient to count as a Turing Machine. The criterion, in particular, does not rule out machines that *simulate* TMs while belonging to a computationally more powerful class. One of the powers of some machines in $\mathcal{C}$ is that they can behave just like TMs. Olympia is one such machine. Though she may stoop to act as if she were a common TM, her high origins cannot be concealed.

We ought to care about which architecture a machine belongs to. If that is not a Turing architecture—and especially if the power of the architecture outstrips that of the Turing architecture—then the machine is not a TM. The architecture of a machine in turn determines which algorithms it can run and the time and space that those algorithms will require. Sprevak puts it well when he notes that

> ...it is not true that a universal computer can run any program. The programs that a computer (universal or otherwise) can run depend on that ma-

> chine's architecture. Certain architectures can run some programs and not others. Programs are at the algorithmic level, and that level is tied to the implementation on particular machines. (Sprevak (2007) 759)

The power of some machines might be such that they can emulate the input-output characteristics *and* the superficial transitions of a Turing Machine, while nevertheless not running the *program* that TMs run. They do not run the same program because they belong to a different architecture. In Olympia's case, that is an O-machine architecture.

The preceding two responses relied on the notions of a computational *architecture* and of *altering* a machine to perform another computation within the same architecture. The two notions are related. A computational architecture, as I'm using the term, defines the basic operations and resources by which a computational system may be manipulated (Pylyshyn (1984) 93). From a programming perspective, the architecture provides the functional primitives from which more complex processes must be built. This means that the basic operations defined by the computational architecture are also the *basic loci of intervention* upon a computational system. An architecture that provides both multiplication and addition as primitive operations can be directly intervened upon in a different way than one that provides only addition as basic and derives multiplication.

Considerations about computational architectures are *explanatorily* critical insofar as we take an interventionist stance on explanation. This is, I suspect, a departure from accounts of computation that focus solely on isomorphism between machine tables and implementing machinery (e.g. Chalmers (2011)). Explanatory claims are claims about causal influence. Claims about causal influence require a well-defined sense in which one could manipulate the system in question from one state to another (Woodward (2003) 115ff). To say that something belongs to a computational architecture, then, is to define the basic computational interventions that could be made upon it.

Changing Olympia's special input i or the transformation sequence S is, I claim, a basic computational intervention. This is an intervention that is not available to Turing Machines. That's what distinguishes Olympia from, say, a simple TM that simply marches along an input tape computing the identity function. (That is why a TM computing the identity function on a tape wherepon is written the solution to the halting problem does not count as solving the halting problem: changes to *input* do not count as manipulations on the computational structure of a TM.) Of course, even a basic computational intervention might require massive changes in the implementing machinery—a change from addition to subtraction at the program level might do all sorts of things at the chip level. The point is merely that *computational* interventions on Olympia could change it trivially to one that computes the halting function

Third and finally, Olympia could actually exist. This means that O-machines could exist. One might worry that this violates the Church-Turing thesis. But this would be to misunderstand the claim. There is nothing (save perhaps sufficient space) prohibiting our universe from containing Trurl. Similarly, there is nothing preventing our universe from containing an infinite set of stone tablets upon which a fiery hand has tediously engraved the solution to the halting problem. The absence of machines like Trurl in our immediate vicinity is explained by the fact that we don't know how to construct them. If the Church-Turing thesis is correct, that failure is a deep limitation upon us. Maudlin's detailed description of Olympia, however, shows one way in which such a machine might be implemented, and there's nothing impossible about that. Similarly, the computational interventions made possible by an architecture are possible in some idealized sense of intervention. These may not be the sorts of things we as finite beings could ever accomplish, or even know how to begin. But such idealizations are familiar when we discuss computation.

4 Computation and Finite Architectures

A final consideration. Olympia is an O-machine, not a Turing Machine. She is therefore irrelevant to any thesis that is specific to the properties of TMs. Of course, both Olympia and Turing Machines are idealizations. Like standard TMs, Olympia has a "(potentially) infinite" tape (Maudlin (1989) p416). This is a necessary qualification. Even if the computation that is actually run requires only a finite amount of space, satisfying the appropriate input counterfactuals may well require an infinite amount of space. (P may fail to halt on some inputs, after all.) Hence both Olympia and TMs must have an infinite tape, and for the same reasons.

Were we to actually build Olympia, we could supply her with only a finite tape. Does the distinction between Olympia and TMs still hold for their finite analogs? I argue that it does. Although finite machines can compute only Turing-computable functions, we can still make architectural distinctions between different types of finite machines. One appropriate test is counterfactual: given a machine of thus-and-such construction, what would be the power of an otherwise identical machine augmented with an infinite tape? Some limited machines would become TMs when so augmented. Others would have an O-machine architecture. (Note that this is not the same as saying they would become able to compute non-TM-computable functions; the claim is only that the expansion would have a certain architecture, not necessarily that it would have new computational properties.)

These two possibilities are worth distinguishing. Again, the architecture of a machine—finite or infinite—depends both on what it does and on how other relevantly similar machines might behave. Making this distinction, I suggest, might also help clarify the common intuition that certain finite computational architectures are somehow second-class compared to finite TMs. I have in mind the armature that forms the core of Olympia and its brutish cousins like Block's Giant Lookup Table.

Maudlin claims that the armature clearly doesn't compute. That is the core of the argument that computation does not depend on actual activity, since the difference between Olympia and the armature is (on input i at least) a difference in wholly inert machinery. But we needn't criticize the armature so harshly. I see no harm in saying that it computes. We can admit O-machines as computers too; if nothing else, they have an important place in computational theory. The lone armature can be treated as a degenerate member of $\mathcal{C}$, after all. It is simply one where S and i are arbitrarily complex and

P is the null program with no states other than 'Halt.' Similarly, lookup tables can't be wholly barred from computation: they play important roles in both assembly language programming (Duntemann (2011)) and chip design (Kaeslin (2008)).

Instead, we ought to say that simple "cheats" like Maudlin's finite armature or Block's GLUT are uninteresting because they are finite versions of O-Machine architectures. Because O-machines can compute any function from integers to integers, it is entirely unsurprising to discover that they can compute some particular such function. When we care about computation, then, we typically care only about computations done by TMs and the like.

Finite lookup devices, even if they are limited to turing-computable functions, don't belong to this class of interesting architectures. At best they might form building blocks from which more interesting architectures can be developed. If consciousness depends on computation, it surely depends on computations performed by architectures in the more interesting class to which TMs belong. This is something that *Maudlin* should be prepared to accept, note, as he explicitly rules out the possibility that machines like the armature could support properties like conscious awareness (p420). I have shown that Olympia, for all of her added complexity, remains architectural kin to the armature. Thus, Maudlin's creation poses no threat to the claim that TM-like architectures might support more interesting properties. [2]

[2]Thanks to [**Suppressed for Review**]. With apologies to Lem (1976).

References

Aaronson, S. (2015). Why philosophers should care about computational complexity. In Copeland, B. J., Posy, C., and Shagrir, O., editors, *Computability: Gödel, Turing, Church, and Beyond*. MIT Press, Cambridge.

Bartlett, G. (2012). Computational theories of conscious experience: Between a rock and a hard place. *Erkenntnis*, 76(2):195–209.

Chalmers, D. J. (2011). A computational foundation for the study of cognition. *Journal of Cognitive Science*, 12(323–357).

Copeland, B. J., editor (2004). *The Essential Turing*. Oxford University Press, Oxford.

Duntemann, J. (2011). *Assembly language step-by-step: Programming with Linux*. John Wiley & Sons, New York.

Kaeslin, H. (2008). *Digital Integrated Circuit Design: From VLSI Architectures to CMOS Fabrication*. Cambridge University Press, New York.

Klein, C. (2008). Dispositional implementation solves the superfluous structure problem. *Synthese*, 165(1):141–153.

Lem, S. (1976). *The Cyberiad*. Avon Books, New York. Trans. Michael Kandel.

Maudlin, T. (1989). Computation and consciousness. *The Journal of Philosophy*, 86(8):407–432.

Pylyshyn, Z. W. (1984). *Computation and cognition*. Cambridge University Press, Cambridge.

Sprevak, M. (2007). Chinese rooms and program portability. *The British Journal for the Philosophy of Science*, 58(4):755–776.

Turing, A. (1938). Systems of logic based on ordinals. In Copeland (2004), pages 125–204.

Woodward, J. (2003). *Making Things Happen*. Oxford University Press, New York.